

 免費電子書

學習

C++

Free unaffiliated eBook created from
Stack Overflow contributors.

#C++

.....	1
1: C ++	2
.....	2
.....	2
Examples	2
.....	2
.....	2
.....	3
.....	3
C/	3
.....	4
.....	4
.....	4
.....	4
.....	4
.....	5
.....	5
.....	6
.....	6
.....	6
.....	6
.....	7
C ++	8
.....	8
2: Arithmetic Metaprogramming	10
.....	10
Examples	10
Olog n	10
3: C ++ 11	12
.....	12
.....	12
.....	12

.....	12
-	12
-	13
.....	13
Examples	13
.....	13
.....	14
4: C ++ Streams	16
.....	16
Examples	16
.....	16
.....	17
.....	17
.....	17
.....	17
.....	17
.....	17
.....	17
.....	18
iostream	18
.....	18
.....	18
.....	18
.....	19
.....	19
STL	19
.....	19
.....	20
5: C ++	21
.....	21
Examples	21

.....	21
6: C ++	22
Examples	22
.....	22
.....	22
.....	22
.....	22
.....	23
/	23
.....	24
.....	24
.....	24
.....	25
7: C ++	26
.....	26
Examples	26
.....	26
.....	26
.....	26
8: C ++	28
.....	28
Examples	28
C ++	28
9: c ++/	30
.....	30
Examples	30
.....	30
10: C ++	31
Examples	31
C ++	31
11: C ++	34

15: constexpr	46
.....	46
.....	46
Examples.....	46
constexpr.....	46
constexpr.....	47
if.....	49
16: Const	50
.....	50
.....	50
Examples.....	50
.....	50
Const.....	51
Const.....	53
Const.....	54
const CV-Qualified Member Functions.....	54
const.....	55
17: const	58
.....	58
.....	58
Examples.....	58
Const.....	58
Const.....	58
Const.....	59
constconst getter.....	59
18: C	62
.....	62
Examples.....	62
.....	62
.....	62
.....	62

19: decltype	63
.....	63
Examples	63
.....	63
.....	63
20: ISO C ++	64
.....	64
.....	64
Examples	64
.....	64
C ++ 11	64
.....	64
.....	64
.....	65
.....	65
.....	65
.....	65
.....	65
.....	65
.....	65
.....	65
.....	65
.....	65
.....	66
C ++ 14	66
.....	66
.....	66
/	66
C ++ 17	66
.....	66
.....	67
C ++ 03	67
.....	67

C ++ 98.....	67
C89 / C90.....	67
.....	67
C ++ 20.....	68
.....	68
.....	68
21: Lambda.....	69
.....	69
.....	69
.....	69
Examples.....	69
lambda.....	69
.....	72
.....	72
.....	73
.....	74
.....	74
lambdas.....	75
.....	76
lambda.....	76
lambdaC ++ 03.....	78
lambdas.....	79
std::function.....	79
.....	79
Y.....	80
lambdas.....	80
22: Pimpl.....	83
.....	83
Examples.....	83
Pimpl.....	83
23: RAI.....	85
.....	85

Examples.....	85
.....	85
/ ScopeExit.....	85
ScopeSuccessc ++ 17.....	86
ScopeFailc ++ 17.....	87
24: RTTI.....	89
Examples.....	89
.....	89
dynamic_cast.....	89
typeid.....	89
c ++.....	90
25: SFINAE.....	91
Examples.....	91
enable_if.....	91
.....	91
void_t.....	92
decltype.....	93
SFINAE.....	94
enable_if_all / enable_if_any.....	95
is_detected.....	96
.....	97
26: static_assert.....	99
.....	99
.....	99
.....	99
Examples.....	99
static_assert.....	99
27: std :: function.....	100
Examples.....	100
.....	100
std :: functionstd :: bind.....	100
std :: function with lambdastd :: bind.....	101

`function`	101
std :: function	102
std :: tuple	104
28: std :: setstd :: multiset	106
.....	106
.....	106
Examples	106
.....	106
.....	107
.....	107
.....	108
.....	108
Lambda	109
.....	109
setmultiset	109
.....	110
29: Typedef	111
.....	111
.....	111
Examples	111
typedef	111
typedef	111
typedef	112
“”	112
30: ODR	113
Examples	113
.....	113
.....	113
ODR	114
31:	116
Examples	116
.....	116

.....	117
.....	118
.....	119
32: C ++C ++ 11C ++ 14C ++ 17C ++	120
Examples	120
.....	120
33:	121
.....	121
std :: shared_mutexstd :: shared_timed_mutex	121
std :: shared_mutexMSVC14.1	121
std :: shared_timed_mutexMSVC14.1	122
std :: shared_mutex/std :: shared_timed_mutex2	126
Examples	129
std :: unique_lockstd :: shared_lockstd :: lock_guard	129
std :: try_to_lockstd :: adopt_lockstd :: defer_lock	130
std ::	131
std :: scoped_lockC ++ 17	131
.....	131
std ::	131
34:	133
.....	133
.....	133
Examples	133
.....	133
35:	135
.....	135
Examples	135
.....	135
C	135
std :: bitset	135
.....	135

C	135
std :: bitset	135
.....	136
C	136
std :: bitset	136
.....	136
C	136
std :: bitset	136
nx.....	136
C	136
std :: bitset	136
.....	137
C	137
std :: bitset	137
.....	137
C	137
.....	137
2.....	138
.....	138
36:	140
.....	140
Examples.....	140
- AND.....	140
- OR.....	140
^ -	141
- NOT.....	142
<< -	143
>> -	143
37: OpenMP	145
.....	145
.....	145

Examples.....	145
OpenMP.....	145
OpenMP.....	146
OpenMPParallel For Loop.....	146
OpenMP/.....	147
38: std :: unordered_map.....	148
.....	148
.....	148
Examples.....	148
.....	148
.....	148
39:	149
Examples.....	149
.....	149
.....	149
40:	151
.....	151
.....	151
.....	151
Examples.....	151
.....	151
.....	151
.....	151
41:	153
Examples.....	153
.....	153
Rethrow.....	154
Try Blocks.....	155
.....	155
.....	155
const.....	156
.....	157

std :: uncaught_exceptions.....	158
.....	159
42:	162
.....	162
Examples.....	162
C ++ 11.....	162
.....	162
43:	164
Examples.....	164
.....	164
.....	166
44:	168
Examples.....	168
.....	168
prvalue.....	168
x.....	168
.....	169
glvalue.....	169
.....	169
45:	171
.....	171
Examples.....	171
/.....	171
.....	171
46:	173
.....	173
.....	173
Examples.....	173
.....	173
.....	175
std :: integer_sequence.....	176
.....	177

.....	177
C ++ 11.....	178
T.....	179
IF-THEN-ELSE.....	180
/.....	180
47:	182
.....	182
.....	182
Examples.....	182
.....	182
.....	183
.....	184
48:	186
.....	186
.....	186
.....	186
.....	186
.....	186
.....	186
.....	186
.....	186
Examples.....	186
.....	186
.....	186
.....	186
.....	187
49:	188
.....	188
Examples.....	188
.....	188
50:	189
Examples.....	189
.....	189
switch.....	189

.....	190
Scoped enums.....	191
C ++ 11.....	191
51:	193
Examples.....	193
gccgprof.....	193
gperf2dot.....	193
gccGoogle Perf ToolsCPU.....	195
52:	198
.....	198
Examples.....	198
.....	198
53:	199
.....	199
.....	199
Examples.....	199
.....	199
.....	200
cv-qualifier.....	200
54:	203
.....	203
.....	203
Examples.....	203
.....	203
55:	205
Examples.....	205
.....	205
56:	206
Examples.....	206
.....	206
.....	206

57:	207
Examples	207
.....	207
C ++	207
58:	209
.....	209
.....	209
Examples	209
.....	209
operatorFunctors	209
59:	211
Examples	211
.....	211
lambdas	211
60:	213
.....	213
.....	213
.....	213
Examples	213
.....	213
.....	214
.....	215
.....	215
.....	216
ADL	216
.....	217
/	218
.....	219
.....	219
.....	220
.....	220
61:	222

.....	222
Examples.....	222
.....	222
.....	223
Singeton.....	224
-	224
62:	225
Examples.....	225
INT.....	225
.....	225
.....	225
char16_t.....	225
char32_t.....	225
.....	225
.....	226
.....	226
.....	226
.....	226
wchar_t.....	227
63:	228
Examples.....	228
.....	228
.....	229
.....	230
64: CRTP	231
.....	231
Examples.....	231
CRTP.....	231
CRTP.....	232
65:	234
.....	234
Examples.....	234
.....	

.....	234
nullptr.....	234
.....	234
.....	235
66:	237
.....	237
.....	237
Examples.....	237
.....	237
.....	237
.....	238
.....	238
EXTERN.....	239
67:	240
.....	240
Examples.....	240
.....	240
68:	242
Examples.....	242
TCP.....	242
TCP.....	245
69:	247
Examples.....	247
Char.....	247
.....	247
char	247
.....	247
char16_t	248
char32_t	248
bool	248
wchar_t	248
.....	

.....	249
.....	249
.....	250
.....	251
.....	251
.....	251
70:	252
.....	252
Examples.....	252
.....	252
.....	254
.....	254
.....	254
.....	254
.....	254
71:	255
.....	255
.....	255
Examples.....	255
.....	255
.....	255
72:	257
.....	257
Examples.....	257
[[[]]].....	257
[[[]]].....	258
[[deprecated]][[deprecated“reason”]].....	258
[[nodiscard]].....	259
[[maybe_unused]].....	259
73:	261
.....

Examples.....261

.....261

.....261

.....262

74:263

.....263

.....263

.....263

Examples.....263

regex_matchregex_search.....263

regex_replace.....264

regex_token_iterator.....264

regex_iterator.....264

.....265

.....265

.....266

75: /GCC.....267

Examples.....267

!***!.....267

.....267

.....267

^***!.....268

***.....268

76:270

.....270

Examples.....270

.....270

77:271

.....271

Examples.....271

.....271

std :: tuple.....	271
std :: array.....	272
std :: pair.....	272
struct.....	273
.....	274
.....	275
std :: vector.....	276
.....	276
78:	278
.....	278
.....	278
.....	278
Examples.....	278
.....	278
.....	280
.....	282
.....	283
Do-while.....	283
.....	284
-	285
79:	287
.....	287
Examples.....	287
.....	287
.....	287
.....	288
80:	289
.....	289
Examples.....	289
.....	289
.....	289
.....	290
.....	

81:	292
.....	292
.....	292
.....	292
Examples	292
.....	292
.....	292
.....	293
.....	293
.....	294
.....	294
.....	294
/	294
/	295
.....	295
82:	296
.....	296
Examples	296
.....	296
.....	296
.....	297
lambdaC ++ 11	298
.....	299
std :: map	300
.....	301
83:	302
Examples	302
.....	302
84:	303
.....	303

Examples.....	303
.....	303
.....	304
std :: vector.....	304
2D.....	305
std :: vector.....	306
.....	307
85: I / O.....	309
.....	309
Examples.....	309
.....	309
.....	310
.....	311
.....	312
.....	313
.....	313
ASCIIstd :: string.....	314
.....	315
`struct`.....	315
.....	316
.....	317
.....	317
86:	319
.....	319
.....	319
Examples.....	319
std :: shared_ptr.....	319
std :: weak_ptr.....	321
std :: unique_ptr.....	322
C.....	324
auto_ptr.....	325
shared_ptr.....	326

std :: shared_ptr.....	327
value_ptr.....	328
87:	331
.....	331
Examples.....	331
.....	331
.....	332
.....	332
88:	334
.....	334
Examples.....	334
std :: futurestd :: promise.....	334
.....	334
std :: packaged_taskstd :: future.....	335
std :: future_errorstd :: future_errc.....	335
std :: futurestd :: async.....	336
.....	337
89:	339
Examples.....	339
.....	339
.....	339
.....	339
.....	340
90:	341
.....	341
.....	341
Examples.....	341
.....	341
voidreturn.....	341
.....	342
.....	342
.....	342
.....	342

..... 343

..... 344

..... 344

..... 345

..... 345

..... 345

..... 346

..... 346

`std``posix` 346

..... 347

..... 347

..... 347

const 347

..... 348

..... 349

..... 349

..... 349

..... 349

[[noreturn]] 349

..... 350

..... 350

91: **352**

..... 352

Examples 352

TU 352

..... 352

* 353

reinterpret_cast 353

..... 353

..... 354

..... 354

..... 355

92: **356**

.....	356
.....	356
Examples.....	356
CMake.....	356
GNU make.....	357
.....	357
.....	357
.....	358
.....	358
SCons.....	359
.....	359
.....	359
NMAKEMicrosoft.....	359
.....	359
AutotoolsGNU.....	359
.....	359
93:	361
Examples.....	361
std :: for_each.....	361
std :: next_permutation.....	361
std ::.....	362
std ::.....	363
std ::.....	364
std :: count_if.....	365
std :: find_if.....	366
std :: min_element.....	367
std :: nth_element.....	368
94:	370
.....	370
.....	370
.....	370

Examples.....	372
.....	372
.....	373
.....	373
.....	374
.....	374
.....	375
.....	376
.....	376
auto.....	377
unique_ptr.....	377
.....	378
.....	378
.....	381
95:	382
.....	382
Examples.....	382
.....	382
.....	382
autoconstreferences.....	383
.....	383
lambdaC ++ 14.....	383
.....	384
96:	386
.....	386
.....	386
Examples.....	387
.....	387
.....	393
.....	394
97:	396
.....	396

Examples.....	396
.....	396
.....	396
.....	396
.....	397
.....	397
.....	397
.....	397
.....	397
.....	398
.....	398
.....	399
if..else	399
breakcontinuegotoexit	400
98:	404
Examples.....	404
.....	404
99:	405
Examples.....	405
.....	405
.....	406
.....	406
.....	406
.....	407
.....	409
100: C ++	412
.....	412
.....	412
Examples.....	412
.....	412
.....	414
.....	416
Fluent APIBuilder.....	417
.....	418
.....	

419	
101:	420
Examples	420
.....	420
.....	420
.....	420
.....	421
.....	421
102: std :: integer_sequence	423
.....	423
Examples	423
std :: tuple	423
.....	424
.....	424
103: std :: string	425
.....	425
.....	425
.....	425
Examples	426
.....	426
.....	426
.....	426
.....	427
.....	427
.....	428
[]n	428
n	428
.....	429
.....	429
.....	429
constchar *	429
.....

430	
/.....	431
.....	431
std :: wstring.....	432
std :: string_view.....	433
.....	434
/.....	434
.....	435
.....	436
std :: string.....	437
104: std :: iomanip.....	438
Examples.....	438
std ::.....	438
std :: setprecision.....	438
std :: setfill.....	438
std :: setiosflags.....	439
105: std ::.....	441
.....	441
Examples.....	441
.....	441
106: std ::Modifiers.....	442
.....	442
.....	442
Examples.....	442
.....	442
.....	442
107: std ::.....	445
Examples.....	445
.....	445
108: std ::.....	448
Examples.....	448
.....	

.....	448
vs.....	448
vs Sentinel.....	448
vs std::pair<bool, T>.....	448
.....	448
.....	449
.....	450
value_or.....	450
109: std ::	451
.....	451
Examples.....	451
.....	451
std :: mapstd :: multimap.....	452
.....	453
.....	453
std :: mapstd :: multimap.....	454
std :: mapstd :: multimap.....	455
.....	456
.....	456
.....	456
.....	456
.....	456
std :: map.....	456
.....	457
110: std ::	458
Examples.....	458
.....	458
.....	458
111: std ::	460
.....	460
.....	460

Examples.....	460
std :: vector.....	460
.....	461
std :: vector.....	462
.....	462
.....	463
const	463
.....	464
.....	464
.....	464
.....	466
std :: vectorC.....	467
/.....	467
.....	468
.....	468
.....	468
.....	468
.....	468
.....	468
.....	468
.....	469
lambda	469
.....	469
.....	469
std :: vector.....	470
std :: vector.....	471
.....	472
.....	472
.....	474
.....	474
.....	475
.....	

.....	477
.....	477

112: std :: **479**

.....	479
-------	-----

Examples 479

std :: variant.....	479
---------------------	-----

.....	479
-------	-----

`std :: variant`	481
------------------------	-----

113: std :: **482**

.....	482
-------	-----

.....	482
-------	-----

Examples 482

std :: array.....	482
-------------------	-----

.....	483
-------	-----

.....	485
-------	-----

.....	485
-------	-----

.....	485
-------	-----

114: **487**

Examples 487

.....	487
-------	-----

.....	487
-------	-----

.....	489
-------	-----

std :: moveOn ² On.....	489
------------------------------------	-----

.....	492
-------	-----

.....	492
-------	-----

115: **494**

.....	494
-------	-----

.....	494
-------	-----

.....	494
-------	-----

Examples 494

.....	494
-------	-----

.....	494
std :: thread.....	495
.....	497
std :: asyncstd :: thread.....	498
.....	498
.....	498
.....	498
.....	499
.....	499
.....	500
.....	502
.....	503
116:	505
.....	505
Examples.....	505
std :: shared_lock.....	505
std :: call_oncestd :: once_flag.....	505
.....	506
std :: condition_variable_anystd :: cv_status.....	507
117:	508
.....	508
.....	508
Examples.....	508
GCC.....	508
.....	509
Visual C ++.....	509
Visual Studio - Hello World.....	512
Clang.....	519
.....	519
C ++.....	520
Code :: Blocks.....	521
118:	526

.....	526
.....	526
.....	526
Examples.....	526
Unix.....	526
CC ++.....	526
119:	528
.....	528
.....	528
Examples.....	528
C ++ 11override with virtual.....	528
.....	529
.....	530
.....	530
.....	531
120:	534
Examples.....	534
.....	534
vec.hhstd :: vector.....	535
File expr.hhvector plusvector inner product.....	537
main.ccsrc.....	540
.....	542
121: Elision	545
Examples.....	545
.....	545
.....	545
.....	546
.....	546
.....	547
elision.....	547
122:	549
.....	549

.....	549
.....	549
Examples.....	549
.....	549
.....	550
.....	550
123:	552
Examples.....	552
.....	552
decltype.....	552
.....	553
.....	553
.....	553
124:	555
.....	555
Examples.....	555
.....	555
.....	555
125:	557
Examples.....	557
.....	557
.....	558
.....	558
.....	559
126:	560
.....	560
Examples.....	560
1.....	560
2.....	561
3.....	561
127:	563
Examples.....	563

.....	563
.....	563
.....	563
.....	563
.....	563
.....	564
128:	565
Examples	565
C	565
.....	565
.....	566
.....	566
.....	566
.....	567
.....	567
.....	567
.....	568
.....	568
.....	569
Map Iterator	569
.....	570
.....	570
129:	572
.....	572
.....	572
Examples	572
.....	572
Lambda	572
130:	573
.....	573
Examples	573
.....

.....	575
.....	575
Pointer CV-Qualifiers.....	576
.....	579
131:	580
.....	580
Examples.....	580
.....	580
ANDOR.....	581
&& 	581
.....	582
132:	583
.....	583
.....	583
Examples.....	583
.....	583
.....	584
.....	585
.....	586
.....	586
.....	588
.....	588
NOT.....	589
I / O.....	589
.....	590
.....	594
133:	596
.....	596
Examples.....	596
.....	596
.....	596

.....	597
.....	598
.....	599
.....	600
constnessvolatility.....	601
134:	603
Examples.....	603
std :: recursive_mutex.....	603
135:	604
.....	604
Examples.....	604
.....	604
.....	606
136:	608
.....	608
.....	608
.....	608
Examples.....	610
ASM.....	610
.....	610
noexcept.....	611
.....	611
sizeof.....	612
.....	612
137:	617
.....	617
Examples.....	617
.....	617
.....	617
.....	618
138:	619

.....	619
.....	619
Examples.....	619
.....	619
.....	620
.....	620
.....	622
Const.....	624
139:	626
.....	626
.....	626
Examples.....	626
.....	626
.....	626
.....	628
.....	631
.....	631
X-.....	632
#pragma.....	634
.....	634
140:	636
Examples.....	636
.....	636
.....	636
141:	638
.....	638
Examples.....	638
.....	638
.....	638
.....	639
.....	639
142: /	641

.....	641
.....	641
Examples.....	641
.....	641
.....	641
.....	642
.....	644
.....	645
.....	646
.....	647
.....	647
.....	648
.....	648
/.....	649
.....	653
.....	656
.....	661
/.....	662
143:	664
.....	664
Examples.....	664
.....	664
.....	664
.....	665
144:	667
.....	667
.....	667
Examples.....	667
.....	667
Lambda.....	667
.....	667
145:	669
.....	

Examples.....	669
.....	669
vtable.....	670
`std :: function`.....	672
T.....	675
std :: any.....	676
146:	681
.....	681
Examples.....	681
.....	681
.....	681
.....	681
.....	681
std :: is_same.....	682
.....	683
.....	684
147:	685
.....	685
.....	685
.....	685
Examples.....	686
.....	686
.....	686
.....	686
.....	687
.....	687
.....	688
.....	688
.....	689
T.....	689
C.....	690

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [cplusplus](#)

It is an unofficial and free C++ ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official C++.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: C ++

“Hello World”。 C ++<iostream> std::cout。

C ++。 [Compiling and Building C ++](#)。

C ++ 98	ISO / IEC 14882:1998	1998-09-01
C ++ 03	ISO / IEC 14882:2003	2003-10-16
C ++ 11	ISO / IEC 14882:2011	2011-09-01
C ++ 14	ISO / IEC 14882:2014	
C ++ 17	TBD	201711
C ++ 20	TBD	202011

Examples

Hello World!

```
#include <iostream>

int main()
{
    std::cout << "Hello World!" << std::endl;
}
```

[Coliru](#)。

- #include <iostream> C ++<iostream>。
iostream。 std。
// O。
• int main() { ... }main。 main。 C ++mainint。
int。 main。
EXIT_SUCCESS。。
returnmain0。 return 0;。

void

- `std::cout << "Hello World!" << std::endl;`“Hello World”

- `std` :: ◦

- `::stdout` ◦ - [Microsoft](#) ◦

- `std::cout` `iostream` `stdout` ◦

- `<<` ◦

- `<<` `stream` `<<` `content` `content` ◦ `std::cout << "Foo" << " Bar";`“FooBar”◦

- `"Hello World!"` “”◦ `iostream`◦

- `std::endl` / `\n`◦ `iostream`◦ ◦

- `std::endl` ◦ “”◦

```
std::cout << "Hello World!\n";
```

`\n` ◦

- `;` ◦ `C ++` / ◦

`C ++` ◦ ◦

`C ++`

`//`

```
int main()
{
    // This is a single-line comment.
    int a; // this also is a single-line comment
    int i; // this is another single-line comment
}
```

C/

`/**/` ◦ `C ++` ◦ “C” `C ++` `C`

```
int main()
{
    /*
     * This is a block comment.
     */
}
```

```
int a;
}
```

◦ */

```
int main()
{
    /* A block comment with the symbol /*
       Note that the compiler is not affected by the second /*
       however, once the end-block-comment symbol is reached,
       the comment ends.
    */
    int a;
}
```

C ++C ◦ /* ◦

◦

```
void SomeFunction(/* argument 1 */ int a, /* argument 2 */ int b);
```

-
- /
 -
 -
 - / ◦

-
-

◦ ◦ ◦

◦ ◦ “” ◦

◦

◦

◦ ◦

- ◦

◦

- C ++ ◦

◦ ◦ ◦

- ◦ ◦
-

◦

```
int add2(int i); // The function is of the type (int) -> (int)
```

```
int add2(int i)
```

- int ◦
- add2 ◦
- 1
 - int◦
 - i◦

```
;
```

```
int add2(int); // Omitting the function arguments' name is also permitted.
```

C ++C ++◦ - ◦ C ++

```
int add2(int i); // The compiler will note that add2 is a function (int) -> int
int add2(int j); // As add2 already has a definition of (int) -> int, the compiler
                  // will regard this as an error.
```

void ◦ ◦

```
void do_something(); // The function takes no parameters, and does not return anything.
                    // Note that it can still affect variables it has access to.
```

◦ main2 add2

```
#include <iostream>

int add2(int i); // Declaration of add2

// Note: add2 is still missing a DEFINITION.
// Even though it doesn't appear directly in code,
// add2's definition may be LINKED in from another object file.

int main()
{
    std::cout << add2(2) << "\n"; // add2(2) will be evaluated at this point,
                                // and the result is printed.
    return 0;
}
```

add2 (2) ◦

*◦

add2

```
int add2(int i)           // Data that is passed into (int i) will be referred to by the name i
{                         // while in the function's curly brackets or "scope."

    int j = i + 2;       // Definition of a variable j as the value of i+2.
    return j;            // Returning or, in essence, substitution of j for a function call to
                          // add2.
}
```



◦

```
int add2(int i)           // Code contained in this definition will be evaluated
{                         // when add2() is called with one parameter.
    int j = i + 2;
    return j;
}

int add2(int i, int j)    // However, when add2() is called with two parameters, the
{                         // code from the initial declaration will be overloaded,
    int k = i + j + 2 ;   // and the code in this declaration will be evaluated
    return k;             // instead.
}
```

add2 ◦ C++ ◦ ◦



◦

```
int multiply(int a, int b = 7); // b has default value of 7.
int multiply(int a, int b)
{
    return a * b;              // If multiply() is called with one parameter, the
}                               // value will be multiplied by the default, 7.
```

multiply() ◦ b7. ◦

```
int multiply(int a = 10, int b = 20); // This is legal
int multiply(int a = 10, int b);      // This is illegal since int a is in the former
```



C++name_of_function(value1, value2, value3) ◦ ◦

! + - * %<< ◦ +C++ ◦

C++; ◦ C++

```
3+3
```

```
operator+(3, 3)
```

operator°

C ++CC ++。 ""C ++ C ++。

C ++。

```
int main()
{
    foo(2); // error: foo is called, but has not yet been declared
}

void foo(int x) // this later definition is not known in main
{
}
```

main()foo()°

```
void foo(int x) {} //Declare the foo function and body first

int main()
{
    foo(2); // OK: foo is completely defined beforehand, so it can be called here.
}
```

""

```
void foo(int); // Prototype declaration of foo, seen by main
                // Must specify return type, name, and argument list types

int main()
{
    foo(2); // OK: foo is known, called even though its body is not yet defined
}

void foo(int x) //Must match the prototype
{
    // Define body of foo here
}
```

void foo int °

```
// foo.h
void foo(int); // prototype declaration
```

```
// foo.cpp --> foo.o
#include "foo.h" // foo's prototype declaration is "hidden" in here
void foo(int x) { } // foo's body definition
```

foo.omain.o

```
// main.cpp --> main.o
```

```
#include "foo.h" // foo's prototype declaration is "hidden" in here
int main() { foo(2); } // foo is valid to call because its prototype declaration was
beforehand.
// the prototype and body definitions of foo are linked through the object files
```

“” ° °

C ++

C ++ °

° °

C ++ “”C ° C ++

1. C ++ #define °
2. C ++ °
3. °
4. °

- ° “” ° “”C ++ °

C ++ ° C ++ °

- ° [1] [http //faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html](http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html)

°

°

° ° °

```
#define ZERO 0
```

```
#include <something>
```

- ° something ° [hello world](#)

```
#include <iostream>
```

°

CC ++ C ++C °

```
#define something something_else
```

- ° somethingsomething_else ° C ++ °

```
something_else somethingsomething°
```

```
#if #else#endif°
```

```
#if something==true
//code
#else
//more code
#endif

#ifdef thing_that_you_want_to_know_if_is_defined
//code
#endif
```

truefalse ° °

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/206/c-plusplus>

2: Arithmetic Metaprogramming

C++

Examples

Olog n

o

```
template <int base, unsigned int exponent>
struct power
{
    static const int halfvalue = power<base, exponent / 2>::value;
    static const int value = halfvalue * halfvalue * power<base, exponent % 2>::value;
};

template <int base>
struct power<base, 0>
{
    static const int value = 1;
    static_assert(base != 0, "power<0, 0> is not allowed");
};

template <int base>
struct power<base, 1>
{
    static const int value = base;
};
```

```
std::cout << power<2, 9>::value;
```

C++ 14

```
template <int base, int exponent>
struct powerDouble
{
    static const int exponentAbs = exponent < 0 ? (-exponent) : exponent;
    static const int halfvalue = powerDouble<base, exponentAbs / 2>::intermediateValue;
    static const int intermediateValue = halfvalue * halfvalue * powerDouble<base, exponentAbs
% 2>::intermediateValue;

    constexpr static double value = exponent < 0 ? (1.0 / intermediateValue) :
intermediateValue;
};

template <int base>
struct powerDouble<base, 0>
{
    static const int intermediateValue = 1;
    constexpr static double value = 1;
};
```

```
    static_assert(base != 0, "powerDouble<0, 0> is not allowed");
};

template <int base>
struct powerDouble<base, 1>
{
    static const int intermediateValue = base;
    constexpr static double value = base;
};

int main()
{
    std::cout << powerDouble<2,-3>::value;
}
```

Arithmetic Metaprogramming <https://riptutorial.com/zh-TW/cplusplus/topic/10907/arithmetic-metaprogramming>

3: C ++ 11

◦ ◦ ◦

◦ C ++ 11◦

◦ `std::atomic<t>`◦ `t`◦

◦ ◦ `std::atomic<unsigned>``std::atomic<std::vector<foo> *``std::atomic<std::pair<bool, char>>`◦

• ◦

• ◦

• ◦ ◦

• - - ◦ ◦

• `std::memory_order`◦

std :: memory_order	
std::memory_order_relaxed	
std::memory_order_release → std::memory_order_acquire	load-acquirestore-releasestore-releasestore-release store-release
std::memory_order_consume	memory_order_acquire
std::memory_order_acq_rel	load-acquirestore-release
std::memory_order_seq_cst	

- ◦

◦ `std::memory_order_seq_cst`◦

◦ ◦ ◦ ◦

◦ `std::memory_order_relaxed`◦ ◦ ◦

-
`std::memory_order_release``std::memory_order_acquire`◦ *load-acquire*◦

load-acquirestore-releasestore-release load-acquire◦

- - `std::memory_order_acq_rel`◦ ◦

◦ - - ◦

◦ ◦

-

release-acquire std::memory_order_consume ◦ - ◦

Fences ◦ ◦

◦ ◦

Examples

```
int x, y;
bool ready = false;

void init()
{
    x = 2;
    y = 3;
    ready = true;
}
void use()
{
    if (ready)
        std::cout << x + y;
}
```

init()use() ◦ use()5◦

- CPU_{init()}

```
void init()
{
    ready = true;
    x = 2;
    y = 3;
}
```

- CPU_{use()}

```
void use()
{
    int local_x = x;
    int local_y = y;
    if (ready)
        std::cout << local_x + local_y;
}
```

- C++◦

init()use() ◦ use()ready==truexy◦

C++

```
int x, y;
std::atomic<bool> ready{false};

void init()
{
    x = 2;
    y = 3;
    ready.store(true, std::memory_order_release);
}
void use()
{
    if (ready.load(std::memory_order_acquire))
        std::cout << x + y;
}
```

init() ◦ true ◦ ready ◦

use() - ◦ ready ◦

CPU

init() S use() ◦ use() ready==true x==2 y==3 ◦

x CPU y use() ◦

```
int x, y;
std::atomic<bool> ready{false};

void init()
{
    x = 2;
    y = 3;
    atomic_thread_fence(std::memory_order_release);
    ready.store(true, std::memory_order_relaxed);
}
void use()
{
    if (ready.load(std::memory_order_relaxed))
    {
        atomic_thread_fence(std::memory_order_acquire);
        std::cout << x + y;
    }
}
```

xy std::cout ◦

◦

```
void block_and_use()
{
    while (!ready.load(std::memory_order_relaxed))
        ;
    atomic_thread_fence(std::memory_order_acquire);
}
```

```
std::cout << x + y;  
}
```

block_and_use() ready° °

C ++ 11 <https://riptutorial.com/zh-TW/cplusplus/topic/7975/c-plusplus-11>

4: C ++ Streams

```
std::istream_iterator<int> ifs;
std::copy(ifs, std::istream_iterator<int>(),
...ifs);
```

Examples

```
std::ostringstream operator<<
```

```
#include <sstream>
#include <string>

using namespace std;

int main()
{
    ostringstream ss;
    ss << "the answer to everything is " << 42;
    const string result = ss.str();
}
```

```
ostringstream ss;
```

o

```
ss << "the answer to everything is " << 42;
```

```
const string result = ss.str();
```

```
result<<"the answer to everything is 42" o
```

o

```
class foo
{
    // All sort of stuff here.
};

ostream &operator<<(ostream &os, const foo &f);
```

foo

```
foo f;
```

```
ostringstream ss;
ss << f;
const string result = ss.str();
```

resultfoo°

ifstream° C ++°

°

ifstreamoperator bool() true° ifstream::operator >>EOF

```
std::ifstream ifs("1.txt");
std::string s;
while(ifs >> s) {
    std::cout << s << std::endl;
}
```

ifstream::operator >>° std::getlineifstream::operator >>° getline

```
while(std::getline(ifs, s)) {
    std::cout << s << std::endl;
}
```

std::getline°

°

```
s.resize(100);
std::copy(std::istreambuf_iterator<char>(ifs), std::istreambuf_iterator<char>(),
    s.begin());
```

std::back_inserter

```
std::copy(std::istreambuf_iterator<char>(ifs), std::istreambuf_iterator<char>(),
    std::back_inserter(s));
```

```
std::vector v(std::istreambuf_iterator<char>(ifs),
    std::istreambuf_iterator<char>());
```

ifs

```
std::ifstream ifs("1.txt", std::ios::binary);
```

```
std::ofstream ofs("out.file");
std::copy(std::istreambuf_iterator<char>(ifs), std::istreambuf_iterator<char>(),
    std::ostream_iterator<char>(ofs));
ofs.close();
```

° Boost.Asio

```
boost::asio::ip::tcp::iostream stream;
stream.connect("example.com", "http");
std::copy(std::istreambuf_iterator<char>(ifs), std::istreambuf_iterator<char>(),
          std::ostream_iterator<char>(stream));
stream.close();
```

STL

```
int arr[100];
std::copy(std::istream_iterator<char>(ifs), std::istream_iterator<char>(), arr);
```

◦ 100◦

iostream

std::ostream_iterator◦ STL◦ std::ostream_iterator◦

```
std::vector<int> v = {1,2,3,4};
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " ! "));
```

1 ! 2 ! 3 ! 4 !

std::ostream_iterator◦ std::cout**3**

```
std::cout << std::setprecision(3);
std::fixed(std::cout);
```

floatstd::ostream_iterator int

```
std::vector<int> v = {1,2,3,4};
std::copy(v.begin(), v.end(), std::ostream_iterator<float>(std::cout, " ! "));
```

1.000 ! 2.000 ! 3.000 ! 4.000 !

std::vector<int> ◦

std::generate std::generate_nstd::transform◦

```
std::vector<int> v = {1,2,3,4,8,16};
```

“x is even”

```
std::boolalpha(std::cout); // print booleans alphabetically
std::transform(v.begin(), v.end(), std::ostream_iterator<bool>(std::cout, " "),
[](int val) {
    return (val % 2) == 0;
});
```

```
std::transform(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " "),
[](int val) {
    return val * val;
});
```

N

```
const int N = 10;
std::generate_n(std::ostream_iterator<int>(std::cout, " "), N, std::rand);
```

◦

```
int v[] = {1,2,3,4,8,16};
std::transform(v, std::end(v), std::ostream_iterator<int>(std::cout, " "),
[](int val) {
    return val * val;
});
```

STL

istream_iterator **STL**◦

```
std::vector<int> v(100);
std::copy(std::istream_iterator<int>(ifs), std::istream_iterator<int>(),
v.begin());
```

```
std::vector<int> v;
std::copy(std::istream_iterator<int>(ifs), std::istream_iterator<int>(),
std::back_inserter(v));
```

◦

istream::operator>>while◦

```
1.12 3.14 foo
2.1 2.2 barr
```

```
std::string s;
double a, b;
while(ifs >> a >> b >> s) {
```

```
std::cout << a << " " << b << " " << s << std::endl;
}
```

std::istream_iterator ◦ std::transform ◦ **3.14**

```
std::vector<double> v(100);
std::transform(std::istream_iterator<int>(ifs), std::istream_iterator<int>(),
v.begin(),
[](int val) {
    return val * 3.14;
});
```

C ++ Streams <https://riptutorial.com/zh-TW/cplusplus/topic/7660/c-plusplus-streams>

5: C ++

C ++。

Examples

。。

§12.7.1。

```
struct W { int j; };
struct X : public virtual W { };
struct Y {
    int *p;
    X x;
    Y() : p(&x.j) { // undefined, x is not yet constructed
    }
};
```

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/9885/c-plusplus>

6: C ++

Examples

1. sizeof(T) >= 1.

```
class Base {};  
  
class Derived : public Base  
{  
public:  
    int i;  
};
```

Derived Base. sizeof(Derived) == sizeof(int). C ++.

Derived. . .

C ++ . .

- . . .
- 199
-
-
- /
-
-
- -
- API

. .

.

```
void func(const A *a); // Some random function  
  
// useless memory allocation + deallocation for the instance  
auto a1 = std::make_unique<A>();  
func(a1.get());  
  
// making use of a stack object prevents  
auto a2 = A{};  
func(&a2);
```

C ++ 14

C++ 14.

```
std::map<std::string, std::unique_ptr<A>> lookup;
// Slow insertion/lookup
// Within this function, we will traverse twice through the map lookup an element
// and even a thirth time when it wasn't in
const A *lazyLookupSlow(const std::string &key) {
    if (lookup.find(key) != lookup.cend())
        lookup.emplace_back(key, std::make_unique<A>());
    return lookup[key].get();
}

// Within this function, we will have the same noticeable effect as the slow variant while
// going at double speed as we only traverse once through the code
const A *lazyLookupSlow(const std::string &key) {
    auto &value = lookup[key];
    if (!value)
        value = std::make_unique<A>();
    return value.get();
}
```

unique

```
std::vector<std::string> stableUnique(const std::vector<std::string> &v) {
    std::vector<std::string> result;
    std::set<std::string> checkUnique;
    for (const auto &s : v) {
        // As insert returns if the insertion was successful, we can deduce if the element was
        // already in or not
        // This prevents an insertion, which will traverse through the map for every unique
        // element
        // As a result we can almost gain 50% if v would not contain any duplicates
        if (checkUnique.insert(s).second)
            result.push_back(s);
    }
    return result;
}
```

/

std::setstd::vector ◦ ◦

```
std::vector<std::string> stableUnique(const std::vector<std::string> &v) {
    std::vector<std::string> result;
    // By reserving 'result', we can ensure that no copying or moving will be done in the
    // vector
    // as it will have capacity for the maximum number of elements we will be inserting
    // If we make the assumption that no allocation occurs for size zero
    // and allocating a large block of memory takes the same time as a small block of memory
    // this will never slow down the program
    // Side note: Compilers can even predict this and remove the checks the growing from the
    // generated code
    result.reserve(v.size());
    std::set<std::string> checkUnique;
    for (const auto &s : v) {
        // See example above
    }
}
```

```

        if (checkUnique.insert(s).second)
            result.push_back(s);
    }
    return result;
}

```

◦

```

// This variant of stableUnique contains a complexity of N log(N)
// N > number of elements in v
// log(N) > insert complexity of std::set
std::vector<std::string> stableUnique(const std::vector<std::string> &v) {
    std::vector<std::string> result;
    std::set<std::string> checkUnique;
    for (const auto &s : v) {
        // See Optimizing by executing less code
        if (checkUnique.insert(s).second)
            result.push_back(s);
    }
    return result;
}

```

N.std :: string◦

```

// This variant of stableUnique contains a complexity of N
// N > number of elements in v
// 1 > insert complexity of std::unordered_set
std::vector<std::string> stableUnique(const std::vector<std::string> &v) {
    std::vector<std::string> result;
    std::unordered_set<std::string> checkUnique;
    for (const auto &s : v) {
        // See Optimizing by executing less code
        if (checkUnique.insert(s).second)
            result.push_back(s);
    }
    return result;
}

```

std::string /◦ ◦

◦ ◦

```

#include <cstring>

class string final
{
    constexpr static auto SMALL_BUFFER_SIZE = 16;

    bool _isAllocated{false}; //< Remember if we allocated memory
    char *_buffer{nullptr}; //< Pointer to the buffer we are using
    char _smallBuffer[SMALL_BUFFER_SIZE]= {'\0'}; //< Stack space used for SMALL OBJECT
    OPTIMIZATION

public:
    ~string()

```

```

{
    if (_isAllocated)
        delete [] _buffer;
}

explicit string(const char *cStyleString)
{
    auto stringSize = std::strlen(cStyleString);
    _isAllocated = (stringSize > SMALL_BUFFER_SIZE);
    if (_isAllocated)
        _buffer = new char[stringSize];
    else
        _buffer = &_amp;smallBuffer[0];
    std::strcpy(_buffer, &cStyleString[0]);
}

string(string &&rhs)
: _isAllocated(rhs._isAllocated)
, _buffer(rhs._buffer)
, _smallBuffer(rhs._smallBuffer) //< Not needed if allocated
{
    if (_isAllocated)
    {
        // Prevent double deletion of the memory
        rhs._buffer = nullptr;
    }
    else
    {
        // Copy over data
        std::strcpy(_smallBuffer, rhs._smallBuffer);
        _buffer = &_amp;smallBuffer[0];
    }
}
// Other methods, including other constructors, copy constructor,
// assignment operators have been omitted for readability
};

```

newdelete° °

_bufferbool_isAllocatedintel 648° °

° ° C++ 11 standard librarystd::basic_string<>std::function<>°

°

° POD°

C++ <https://riptutorial.com/zh-TW/cplusplus/topic/4474/c-plusplus>

7: C ++

。

C ++ “” “”。

。

Examples

[Google Test](#) [Google C ++](#) [gtest](#)。

```
// main.cpp

#include <gtest/gtest.h>
#include <iostream>

// Google Test test cases are created using a C++ preprocessor macro
// Here, a "test suite" name and a specific "test name" are provided.
TEST(module_name, test_name) {
    std::cout << "Hello world!" << std::endl;
    // Google Test will also provide macros for assertions.
    ASSERT_EQ(1+1, 2);
}

// Google Test can be run manually from the main() function
// or, it can be linked to the gtest_main library for an already
// set-up main() function primed to accept Google Test test cases.
int main(int argc, char** argv) {
    ::testing::InitGoogleTest(&argc, argv);

    return RUN_ALL_TESTS();
}

// Build command: g++ main.cpp -lgtest
```

[CatchTDD](#) [BDD](#)。

Catch

```
SCENARIO( "vectors can be sized and resized", "[vector]" ) {
    GIVEN( "A vector with some items" ) {
        std::vector v( 5 );

        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 5 );

        WHEN( "the size is increased" ) {
            v.resize( 10 );

            THEN( "the size and capacity change" ) {
```

```

        REQUIRE( v.size() == 10 );
        REQUIRE( v.capacity() >= 10 );
    }
}
WHEN( "the size is reduced" ) {
    v.resize( 0 );

    THEN( "the size changes but not capacity" ) {
        REQUIRE( v.size() == 0 );
        REQUIRE( v.capacity() >= 5 );
    }
}
WHEN( "more capacity is reserved" ) {
    v.reserve( 10 );

    THEN( "the capacity changes but not the size" ) {
        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 10 );
    }
}
WHEN( "less capacity is reserved" ) {
    v.reserve( 0 );

    THEN( "neither size nor capacity are changed" ) {
        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 5 );
    }
}
}
}
}

```

```

Scenario: vectors can be sized and resized
  Given: A vector with some items
  When: more capacity is reserved
  Then: the capacity changes but not the size

```

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/9928/c-plusplus>

8: C ++

C ++UTF-8UTF-16。 ANSI / ASCII。

。

Unicode。

Examples

C ++

```
#include <iostream>
#include <string>

int main()
{
    const char * C_String = "This is a line of text w";
    const char * C_Problem_String = "This is a line of text Ź";
    std::string Std_String("This is a second line of text w");
    std::string Std_Problem_String("This is a second line of ex Ź");

    std::cout << "String Length: " << Std_String.length() << '\n';
    std::cout << "String Length: " << Std_Problem_String.length() << '\n';

    std::cout << "CString Length: " << strlen(C_String) << '\n';
    std::cout << "CString Length: " << strlen(C_Problem_String) << '\n';
    return 0;
}
```

WindowsOSXGCCMSVC 。

Microsoft MSVC

```
31
31
CString24
CString24
```

MSVC“”。

Unicode。

GNC / GCC

```
31
36
CString24
CString26
```

LinuxGCC 。

Unicode“” 。

- CC ++ **C ++char** 。

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/5270/c-plusplus>

9: C++/

<iostream>

stream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

cin ◦ cin ◦ cout ◦

```
cin >> value
```

cin - input stream
'>>' - extraction operator
value - variable (destination)

cin ◦ **ENTER** ◦

```
cout << "Enter a value: "
```

cout - output stream
'<<' - insertion operator
"Enter a value: " - string to be displayed

cout

std::stream::stream ◦

std::endl ◦ ◦ '\n' ◦ ◦

Examples

```
#include <iostream>

int main()
{
    int value;
    std::cout << "Enter a value: " << std::endl;
    std::cin >> value;
    std::cout << "The square of entered value is: " << value * value << std::endl;
    return 0;
}
```

C++/ <https://riptutorial.com/zh-TW/cplusplus/topic/10683/c-plusplus->

10: C ++

Examples

C ++

```
class listNode
{
    public:
    int data;
    listNode *next;
    listNode(int val):data(val),next(NULL){}
};
```

List

```
class List
{
    public:
    listNode *head;
    List():head(NULL){}
    void insertAtBegin(int val);
    void insertAtEnd(int val);
    void insertAtPos(int val);
    void remove(int val);
    void print();
    ~List();
};
```

```
void List::insertAtBegin(int val)//inserting at front of list
{
    listNode *newnode = new listNode(val);
    newnode->next=this->head;
    this->head=newnode;
}
```

```
void List::insertAtEnd(int val) //inserting at end of list
{
    if(head==NULL)
    {
        insertAtBegin(val);
        return;
    }
    listNode *newnode = new listNode(val);
    listNode *ptr=this->head;
    while(ptr->next!=NULL)
    {
        ptr=ptr->next;
    }
    ptr->next=newnode;
}
```

```

void List::insertAtPos(int pos,int val)
{
    listNode *newnode=new listNode(val);
    if(pos==1)
    {
        //as head
        newnode->next=this->head;
        this->head=newnode;
        return;
    }
    pos--;
    listNode *ptr=this->head;
    while(ptr!=NULL && --pos)
    {
        ptr=ptr->next;
    }
    if(ptr==NULL)
    return;//not enough elements
    newnode->next=ptr->next;
    ptr->next=newnode;
}

```

```

void List::remove(int toBeRemoved)//removing an element
{
    if(this->head==NULL)
    return; //empty
    if(this->head->data==toBeRemoved)
    {
        //first node to be removed
        listNode *temp=this->head;
        this->head=this->head->next;
        delete(temp);
        return;
    }
    listNode *ptr=this->head;
    while(ptr->next!=NULL && ptr->next->data!=toBeRemoved)
    ptr=ptr->next;
    if(ptr->next==NULL)
    return;//not found
    listNode *temp=ptr->next;
    ptr->next=ptr->next->next;
    delete(temp);
}

```

```

void List::print()//printing the list
{
    listNode *ptr=this->head;
    while(ptr!=NULL)
    {
        cout<<ptr->data<<" ";
        ptr=ptr->next;
    }
    cout<<endl;
}

```

```

List::~List()
{
    listNode *ptr=this->head,*next=NULL;
}

```

```
while (ptr!=NULL)
{
    next=ptr->next;
    delete (ptr);
    ptr=next;
}
}
```

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/7485/c-plusplus>

11: C ++

Examples

Fibonacci

Fibonacci

```
int get_term_fib(int n)
{
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return get_term_fib(n - 1) + get_term_fib(n - 2);
}
```

n ° °

```
int get_term_fib(int n, int prev = 0, int curr = 1)
{
    if (n == 0)
        return prev;
    if (n == 1)
        return curr;
    return get_term_fib(n - 1, curr, prev + curr);
}
```

Fibonacci

memoization

° °

memoization

```
#include <map>

int fibonacci(int n)
{
    static std::map<int, int> values;
    if (n==0 || n==1)
        return n;
    std::map<int,int>::iterator iter = values.find(n);
    if (iter == values.end())
    {
        return values[n] = fibonacci(n-1) + fibonacci(n-2);
    }
    else
    {
        return iter->second;
    }
}
```

```
}
```

\$ On\$. \$ O1\$.

◦ ◦

```
#include <map>

int fibonacci(int n, std::map<int, int> values)
{
    if (n==0 || n==1)
        return n;
    std::map<int,int>::iterator iter = values.find(n);
    if (iter == values.end())
    {
        return values[n] = fibonacci(n-1) + fibonacci(n-2);
    }
    else
    {
        return iter->second;
    }
}
```

◦ ◦ ;◦

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/5693/c-plusplus>

12: C ++

-
-
-

Examples

- ◦
- ◦ ◦
- ◦

```
int func(int f, int b) {
    //new variables are created and values from the outside copied
    //f has a value of 0
    //inner_b has a value of 1
    f = 1;
    //f has a value of 1
    b = 2;
    //inner_b has a value of 2
    return f+b;
}

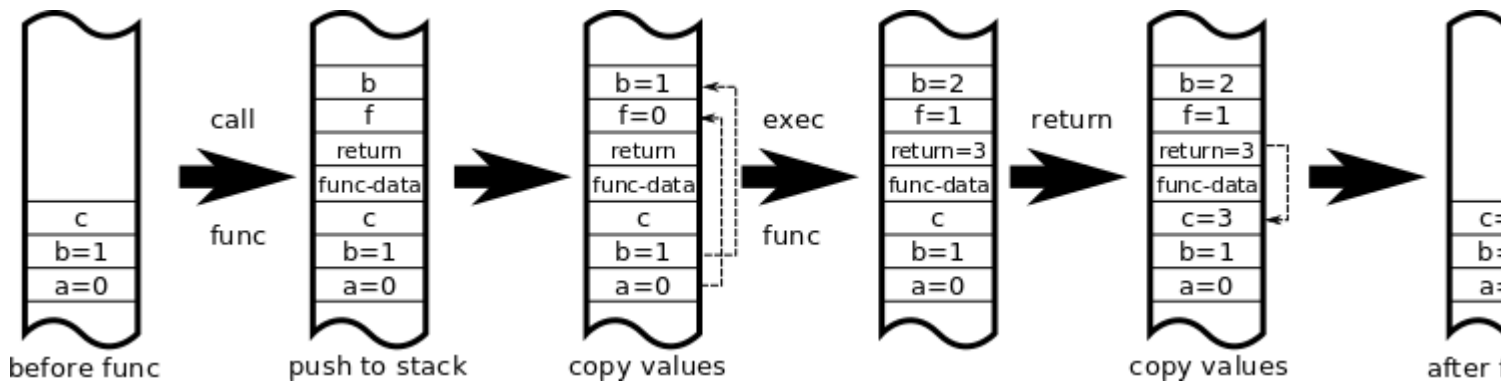
int main(void) {
    int a = 0;
    int b = 1; //outer_b
    int c;

    c = func(a,b);
    //the return value is copied to c

    //a has a value of 0
    //outer_b has a value of 1 <--- outer_b and inner_b are different variables
    //c has a value of 3
}
```

main ◦ ◦ finner_b b ◦ a<->fb<->b ◦

varibale b ◦ ◦



“”。

C ++^{“”} <https://riptutorial.com/zh-TW/cplusplus/topic/10669/c-plusplus---->

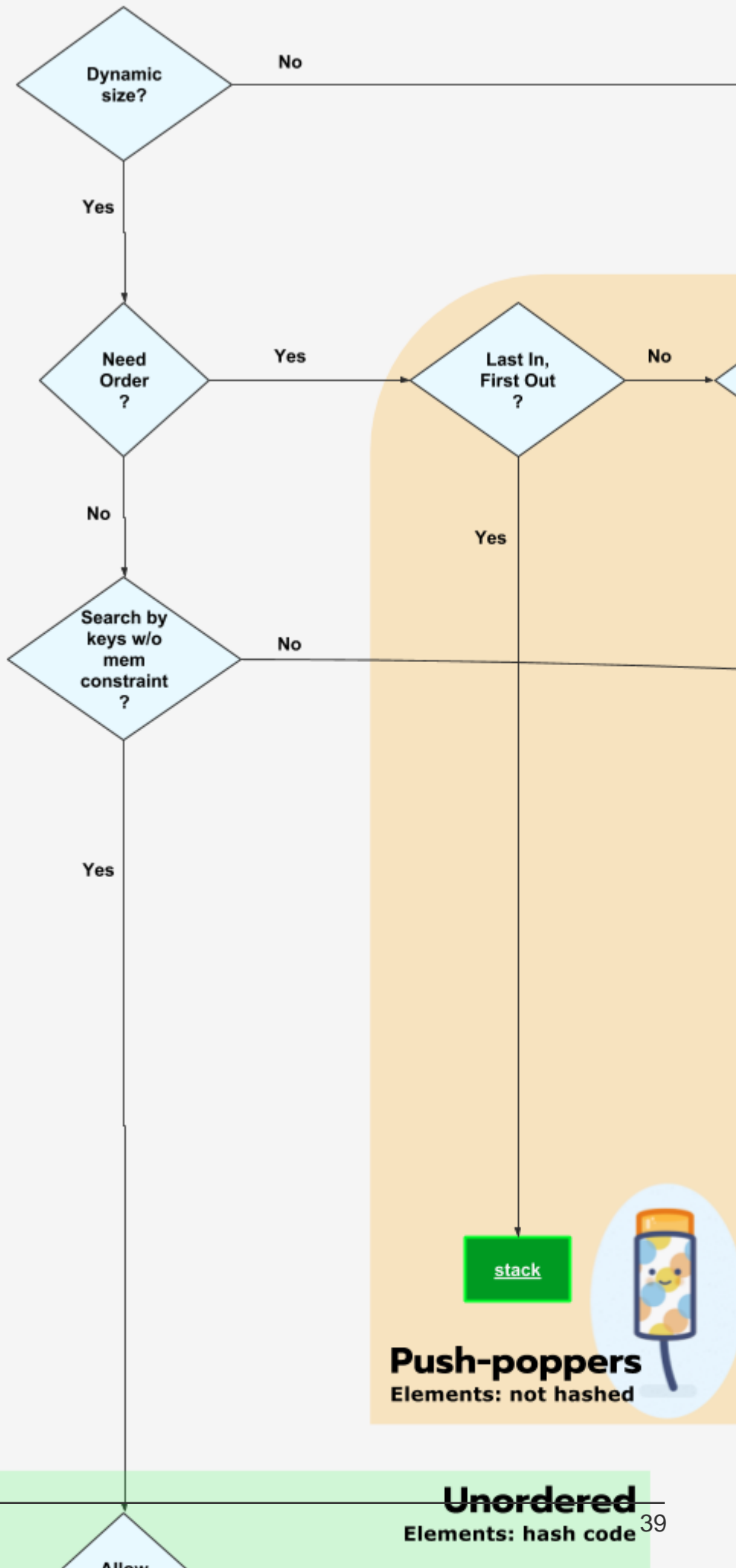
13: C ++

C ++。 ◦ C ++intMyClass。

Examples

C ++

C ++Container。



stack



Push-poppers
Elements: not hashed

Unordered
Elements: hash code

14: C ++

C ++。 。 。

/

-
- UBSanTSanMSanESan
- CFI
-

Examples

C ++segfault - valgrind

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}
```

-g

```
g++ -g -o main main.cpp
```

```
$ ./main
Segmentation fault (core dumped)
$
```

valgrind

```
$ valgrind ./main
==8515== Memcheck, a memory error detector
==8515== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==8515== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==8515== Command: ./main
==8515==
==8515== Conditional jump or move depends on uninitialised value(s)
==8515==    at 0x400813: fail() (main.cpp:7)
==8515==    by 0x40083F: main (main.cpp:13)
==8515==
==8515== Invalid read of size 4
```

```

==8515==      at 0x400819: fail() (main.cpp:8)
==8515==      by 0x40083F: main (main.cpp:13)
==8515== Address 0x0 is not stack'd, malloc'd or (recently) free'd
==8515==
==8515== Process terminating with default action of signal 11 (SIGSEGV): dumping core
==8515== Access not within mapped region at address 0x0
==8515==      at 0x400819: fail() (main.cpp:8)
==8515==      by 0x40083F: main (main.cpp:13)
==8515== If you believe this happened as a result of a stack
==8515== overflow in your program's main thread (unlikely but
==8515== possible), you can try to increase the size of the
==8515== main thread stack using the --main-stacksize= flag.
==8515== The main thread stack size used in this run was 8388608.
==8515==
==8515== HEAP SUMMARY:
==8515==      in use at exit: 72,704 bytes in 1 blocks
==8515== total heap usage: 1 allocs, 0 frees, 72,704 bytes allocated
==8515==
==8515== LEAK SUMMARY:
==8515==      definitely lost: 0 bytes in 0 blocks
==8515==      indirectly lost: 0 bytes in 0 blocks
==8515==      possibly lost: 0 bytes in 0 blocks
==8515==      still reachable: 72,704 bytes in 1 blocks
==8515==      suppressed: 0 bytes in 0 blocks
==8515== Rerun with --leak-check=full to see details of leaked memory
==8515==
==8515== For counts of detected and suppressed errors, rerun with: -v
==8515== Use --track-origins=yes to see where uninitialised values come from
==8515== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
$

```

```

==8515== Invalid read of size 4
==8515==      at 0x400819: fail() (main.cpp:8)
==8515==      by 0x40083F: main (main.cpp:13)
==8515== Address 0x0 is not stack'd, malloc'd or (recently) free'd

```

4. ◦ main.cpp8fail()main.cppmain 13.◦

main.cpp8

```
std::cout << *p3 << std::endl;
```

```

==8515== Conditional jump or move depends on uninitialised value(s)
==8515==      at 0x400813: fail() (main.cpp:7)
==8515==      by 0x40083F: main (main.cpp:13)

```

7

```
if (p3) {
```

p3p2◦ p3

```
int *p3 = p1;
```

Valgrind--track-origins=yes

```
valgrind --track-origins=yes ./main
```

valgrindvalgrind。。

```
==8517== Conditional jump or move depends on uninitialised value(s)
==8517==    at 0x400813: fail() (main.cpp:7)
==8517==    by 0x40083F: main (main.cpp:13)
==8517== Uninitialised value was created by a stack allocation
==8517==    at 0x4007F6: fail() (main.cpp:3)
```

73

```
int *p1;
```

。

GDBSegfault

。

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p2 << std::endl;
    }
}

int main() {
    fail();
}
```

```
g++ -g -o main main.cpp
```

gdb

```
gdb ./main
```

gdb shell。 run。

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/opencog/code-snippets/stackoverflow/a.out

Program received signal SIGSEGV, Segmentation fault.
```

```
0x0000000000400850 in fail () at debugging_with_gdb.cc:11
11         std::cout << *p2 << std::endl;
```

11. p2. .

```
(gdb) print p2
$1 = (int *) 0x0
```

p20x0NULL. NULL. .

.

```
int main() {
    int value;
    std::vector<int> vectorToSort;
    vectorToSort.push_back(42); vectorToSort.push_back(13);
    for (int i = 52; i; i = i - 1)
    {
        vectorToSort.push_back(i * 2);
    }
    /// Optimized for sorting small vectors
    if (vectorToSort.size() == 1);
    else
    {
        if (vectorToSort.size() <= 2)
            std::sort(vectorToSort.begin(), std::end(vectorToSort));
    }
    for (value : vectorToSort) std::cout << value << ' ';
    return 0; }
```

```
std::vector<int> createSemiRandomData() {
    std::vector<int> data;
    data.push_back(42);
    data.push_back(13);
    for (int i = 52; i; --i)
        vectorToSort.push_back(i * 2);
    return data;
}

/// Optimized for sorting small vectors
void sortVector(std::vector &v) {
    if (vectorToSort.size() == 1)
        return;
    if (vectorToSort.size() > 2)
        return;

    std::sort(vectorToSort.begin(), vectorToSort.end());
}

void printVector(const std::vector<int> &v) {
    for (auto i : v)
        std::cout << i << ' ';
}

int main() {
    auto vectorToSort = createSemiRandomData();
    sortVector(std::ref(vectorToSort));
}
```

```
printVector(vectorToSort);  
  
return 0;  
}
```

-
- ◦
- main() **3** ◦

bug ◦

/

- " ◦ ◦
- IDE ◦

/◦ v.begin() **VS** std::end(v) ◦

- - optimized
 - sortVector()
 - std::ref() sortVector()

◦ ◦ **PS2** ◦

◦ ◦

if if (var); ◦

- clang++ -Wall -Weverything -Werror ...
- g++ -Wall -Weverything -Werror ...
- cl.exe /W4 /WX ...

◦ while (staticAtomicBool);while (localBool);while (localBool);◦

◦ ◦

◦ **C ++ 98**

- clang++ -Wall -Weverything -Werror -Wno-errorstoaccept ...
- g++ -Wall -Weverything -Werror -Wno-errorstoaccept ...
- cl.exe /W4 /WX /wd<no of warning>...

◦ ◦ ◦

- clang-tidy
 - -
 -
 -
 - - make_unique
 - - nullptr
 -
 -

Clang◦

- visual studio
- crazy QtClang

C ++◦ ◦

bug◦ ◦

◦ ◦ ◦

◦ ◦

/◦ ◦

bug◦ [CPI](#)◦ ◦

CPU<1◦

[clang](#)-fsanitize=safe-stack◦ [GCC](#)◦

◦ ◦

[C ++](#) <https://riptutorial.com/zh-TW/cplusplus/topic/9814/c-plusplus>

15: constexpr

constexpr C++ 17 if.

constexpr C++ 11 C++ 11. C++ 11. C++ 14constexpr.

Examples

constexpr

constexpr const.

#define

constexpr #define. constexpr.

C++ 11

```
int main()
{
    constexpr int N = 10 + 2;
    cout << N;
}
```

```
cout << 12;
```

.

```
#define N 10 + 2
```

```
int main()
{
    cout << N;
}
```

```
cout << 10 + 2;
```

```
cout << 10 + 2; . . .
```

```
#define
```

```
cout << N * 2;
```

```
cout << 10 + 2 * 2; // 14
```

constexpr24.

const

const ◦ constexpr ◦ constexpr ◦ const ◦

```
int main()
{
    const int size1 = 10;
    const int size2 = abs(10);

    int arr_one[size1];
    int arr_two[size2];
}
```

GCC ◦ ◦ size210 ◦

const ◦ const ◦ #define ◦

C++ 11

```
int main()
{
    constexpr int size = 10;

    int arr[size];
}
```

constexpr ◦

C++ 11

```
constexpr int size = abs(10);
```

abs constexpr ◦

constexpr ◦

C++ 11

```
constexpr bool FailFatal = true;
constexpr float PI = 3.14f;
constexpr char* site = "StackOverflow";
```

auto

C++ 11

```
constexpr auto domain = ".COM"; // const char * const domain = ".COM"
constexpr auto PI = 3.14; // constexpr double
```

constexpr

constexpr ◦

C++ 11

```
constexpr int Sum(int a, int b)
{
    return a + b;
}
```

constexpr

C++ 11

```
int main()
{
    constexpr int S = Sum(10,20);

    int Array[S];
    int Array2[Sum(20,30)]; // 50 array size, compile time
}
```

constexpr S constexpr **const** Sum constexpr °

constexpr

C++ 11

```
int a = 20;
auto sum = Sum(a, abs(-20));
```

Sum constexpr ° °

const

C++ 11

```
int a = 20;
constexpr auto sum = Sum(a, abs(-20));
```

constexpr ° Sum constexpr **RconstL** constexpr °

constexpr °

C++ 11

```
constexpr int Sum(int a, int b)
{
    int a1 = a; // ERROR
    return a + b;
}
```

a1 **constexpr** constexpr ° constexpr a

C++ 11

```
constexpr int Sum(int a, int b)
{
    constexpr int a1 = a;    // ERROR
    ..
}
```

C++ 11

```
constexpr int Sum(int a, int b)
{
    return abs(a) + b; // or abs(a) + abs(b)
}
```

abs(a) abs(10) abs constexpr int

C++ 11

```
constexpr int Abs(int v)
{
    return v >= 0 ? v : -v;
}

constexpr int Sum(int a, int b)
{
    return Abs(a) + b;
}
```

Abs constexpr Abs Sum constexpr Sum(-10, 20) 30

if

C++ 17

if constexpr

```
template<class T, class ... Rest>
void g(T &&p, Rest &&...rs)
{
    // ... handle p
    if constexpr (sizeof...(rs) > 0)
        g(rs...); // never instantiated with an empty argument list
}
```

return

if constexpr #ifndef #ifdef #ifndef if constexpr #ifndef

```
if constexpr(false) {
    foobar; // error; foobar has not been declared
    std::vector<int> v("hello, world"); // error; no matching constructor
}
```

constexpr <https://riptutorial.com/zh-TW/cplusplus/topic/3899/constexpr>

16: Const

- `class ClassOne {public: bool non_modifying_member_function() const { /* ... */ }};`
- `int ClassTwo::non_modifying_member_function() const { /* ... */ }`
- `void ClassTwo::modification_member_function() { /* ... */ }`
- `char non_param_modding_fun() const { ClassOne one; const ClassTwo * two { /* ... */ } }`
- `float parameter_modifying_function() const { ClassTwo one; ClassOne * two { /* ... */ } }`
- `short ClassThree::non_modding_non_param_modding_fun() const { ClassOne one; const { /* ... */ } }`

`const` ◦ `Const` ◦ `Correct Function Parameters` ◦ `Const` ◦ `Correct Function Parameters` ◦

`const` ◦ `const` ◦ `const` ◦

`volatile` ◦ `const` ◦

ISO_CPP

C++

Examples

`const` ◦ ◦ `const` ◦ `const` ◦

`const` ◦ [CV-qualifiers](#) / `const` ◦

```
class ConstCorrectClass {
    int x;

public:
    int getX() const { return x; } // Function is const: Doesn't modify instance.
    void setX(int i) { x = i; }    // Not const: Modifies instance.
};

// Parameter is const: Doesn't modify parameter.
int const_correct_reader(const ConstCorrectClass& c) {
    return c.getX();
}

// Parameter isn't const: Modifies parameter.
void const_correct_writer(ConstCorrectClass& c) {
    c.setX(42);
}

const ConstCorrectClass invariant; // Instance is const: Can't be modified.
ConstCorrectClass          variant; // Instance isn't const: Can be modified.

// ...

const_correct_reader(invariant); // Good.    Calling non-modifying function on const instance.
const_correct_reader(variant);   // Good.    Calling non-modifying function on modifiable
instance.
```

```
const_correct_writer(variant); // Good. Calling modifying function on modifiable instance.
const_correct_writer(invariant); // Error. Calling modifying function on const instance.
```

const;constconstconstconstconst **CV**◦

Const

const **-correct**this **CV**const **mutable**const ;const **cv-qualified**const ◦ **CV**const T*T*const T* ◦ const ◦

constconstConst Correct Function Parameters◦

this **cv-qualifiers**

```
// Assume class Field, with member function "void insert_value(int);".

class ConstIncorrect {
    Field fld;

public:
    ConstIncorrect(Field& f); // Modifies.

    Field& getField();        // Might modify. Also exposes member as non-const reference,
                               // allowing indirect modification.
    void setField(Field& f); // Modifies.

    void doSomething(int i); // Might modify.
    void doNothing();        // Might modify.
};

ConstIncorrect::ConstIncorrect(Field& f) : fld(f) {} // Modifies.
Field& ConstIncorrect::getField() { return fld; }    // Doesn't modify.
void ConstIncorrect::setField(Field& f) { fld = f; } // Modifies.
void ConstIncorrect::doSomething(int i) {           // Modifies.
    fld.insert_value(i);
}
void ConstIncorrect::doNothing() {}                // Doesn't modify.

class ConstCorrectCVQ {
    Field fld;

public:
    ConstCorrectCVQ(Field& f); // Modifies.

    const Field& getField() const; // Doesn't modify. Exposes member as const reference,
                                    // preventing indirect modification.
    void setField(Field& f); // Modifies.

    void doSomething(int i); // Modifies.
    void doNothing() const; // Doesn't modify.
};

ConstCorrectCVQ::ConstCorrectCVQ(Field& f) : fld(f) {}
Field& ConstCorrectCVQ::getField() const { return fld; }
void ConstCorrectCVQ::setField(Field& f) { fld = f; }
void ConstCorrectCVQ::doSomething(int i) {
    fld.insert_value(i);
}
}
```

```

void ConstCorrectCVQ::doNothing() const {}

// This won't work.
// No member functions can be called on const ConstIncorrect instances.
void const_correct_func(const ConstIncorrect& c) {
    Field f = c.getField();
    c.do_nothing();
}

// But this will.
// getField() and doNothing() can be called on const ConstCorrectCVQ instances.
void const_correct_func(const ConstCorrectCVQ& c) {
    Field f = c.getField();
    c.do_nothing();
}

```

Const Correct Function Parameters **const -correct** ◦

```

class ConstCorrect {
    Field fld;

public:
    ConstCorrect(const Field& f); // Modifies instance. Doesn't modify parameter.

    const Field& getField() const; // Doesn't modify. Exposes member as const reference,
                                   // preventing indirect modification.
    void setField(const Field& f); // Modifies instance. Doesn't modify parameter.

    void doSomething(int i); // Modifies. Doesn't modify parameter (passed by value).
    void doNothing() const; // Doesn't modify.
};

ConstCorrect::ConstCorrect(const Field& f) : fld(f) {}
Field& ConstCorrect::getField() const { return fld; }
void ConstCorrect::setField(const Field& f) { fld = f; }
void ConstCorrect::doSomething(int i) {
    fld.insert_value(i);
}
void ConstCorrect::doNothing() const {}

```

const **const; constainers** const ◦

```

class ConstCorrectContainer {
    int arr[5];

public:
    // Subscript operator provides read access if instance is const, or read/write access
    // otherwise.
    int& operator[](size_t index) { return arr[index]; }
    const int& operator[](size_t index) const { return arr[index]; }

    // ...
};

```

const ◦

Const

const ~~-correct~~ const ~~CV~~-this const T*T^o

```
struct Example {
    void func()          { std::cout << 3 << std::endl; }
    void func() const { std::cout << 5 << std::endl; }
};

void const_incorrect_function(Example& one, Example* two) {
    one.func();
    two->func();
}

void const_correct_function(const Example& one, const Example* two) {
    one.func();
    two->func();
}

int main() {
    Example a, b;
    const_incorrect_function(a, &b);
    const_correct_function(a, &b);
}

// Output:
3
3
5
5
```

const const ~~-correct~~ const ~~-incorrect~~ const ~~-correct~~ const ~~-incorrect~~ const const^o [const ~~-incorrect~~ const const^o

```
// Read value from vector, then compute & return a value.
// Caches return values for speed.
template<typename T>
const T& bad_func(std::vector<T>& v, Helper<T>& h) {
    // Cache values, for future use.
    // Once a return value has been calculated, it's cached & its index is registered.
    static std::vector<T> vals = {};

    int v_ind = h.get_index(); // Current working index for v.
    int vals_ind = h.get_cache_index(v_ind); // Will be -1 if cache index isn't registered.

    if (vals.size() && (vals_ind != -1) && (vals_ind < vals.size()) && !(h.needs_recalc())) {
        return vals[h.get_cache_index(v_ind)];
    }

    T temp = v[v_ind];

    temp -= h.poll_device();
    temp *= h.obtain_random();
    temp += h.do_tedious_calculation(temp, v[h.get_last_handled_index()]);

    // We're feeling tired all of a sudden, and this happens.
    if (vals_ind != -1) {
        vals[vals_ind] = temp;
    }
}
```

```

    } else {
        v.push_back(temp); // Oops. Should've been accessing vals.
        vals_ind = vals.size() - 1;
        h.register_index(v_ind, vals_ind);
    }

    return vals[vals_ind];
}

// Const correct version. Is identical to above version, so most of it shall be skipped.
template<typename T>
const T& good_func(const std::vector<T>& v, Helper<T>& h) {
    // ...

    // We're feeling tired all of a sudden, and this happens.
    if (vals_ind != -1) {
        vals[vals_ind] = temp;
    } else {
        v.push_back(temp); // Error: discards qualifiers.
        vals_ind = vals.size() - 1;
        h.register_index(v_ind, vals_ind);
    }

    return vals[vals_ind];
}

```

Const

const◦ const const◦

const CV-Qualified Member Functions

- const
 - ◦ mutable◦
 - mutable◦
- const
 - ◦
 - ◦

```

// ConstMemberFunctions.h

class ConstMemberFunctions {
    int val;
    mutable int cache;
    mutable bool state_changed;

public:
    // Constructor clearly changes logical state. No assumptions necessary.
    ConstMemberFunctions(int v = 0);

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val(). It may or may not call squared_calc() or bad_func().
    int calc() const;

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val(). It may or may not call calc() or bad_func().

```

```

int squared_calc() const;

// We can assume this function doesn't change logical state, and doesn't call
// set_val(). It may or may not call calc() or squared_calc().
void bad_func() const;

// We can assume this function changes logical state, and may or may not call
// calc(), squared_calc(), or bad_func().
void set_val(int v);
};

```

const

```

// ConstMemberFunctions.cpp

ConstMemberFunctions::ConstMemberFunctions(int v /* = 0 */)
: cache(0), val(v), state_changed(true) {}

// Our assumption was correct.
int ConstMemberFunctions::calc() const {
    if (state_changed) {
        cache = 3 * val;
        state_changed = false;
    }

    return cache;
}

// Our assumption was correct.
int ConstMemberFunctions::squared_calc() const {
    return calc() * calc();
}

// Our assumption was incorrect.
// Function fails to compile, due to `this` losing qualifiers.
void ConstMemberFunctions::bad_func() const {
    set_val(863);
}

// Our assumption was correct.
void ConstMemberFunctions::set_val(int v) {
    if (v != val) {
        val = v;
        state_changed = true;
    }
}

```

const

- const
 - ◦
 - /◦
- const
 - ◦
 - /◦

◦

```

// function_parameter.h

// We can assume that c isn't modified (and c.set_val() isn't called), and isn't passed
// to non_qualified_function_parameter(). If passed to one_const_one_not(), it is the first
// parameter.
void const_function_parameter(const ConstMemberFunctions& c);

// We can assume that c is modified and/or c.set_val() is called, and may or may not be passed
// to any of these functions. If passed to one_const_one_not, it may be either parameter.
void non_qualified_function_parameter(ConstMemberFunctions& c);

// We can assume that:
// l is not modified, and l.set_val() won't be called.
// l may or may not be passed to const_function_parameter().
// r is modified, and/or r.set_val() may be called.
// r may or may not be passed to either of the preceding functions.
void one_const_one_not(const ConstMemberFunctions& l, ConstMemberFunctions& r);

// We can assume that c isn't modified (and c.set_val() isn't called), and isn't passed
// to non_qualified_function_parameter(). If passed to one_const_one_not(), it is the first
// parameter.
void bad_parameter(const ConstMemberFunctions& c);

```

const

```

// function_parameter.cpp

// Our assumption was correct.
void const_function_parameter(const ConstMemberFunctions& c) {
    std::cout << "With the current value, the output is: " << c.calc() << '\n'
               << "If squared, it's: " << c.squared_calc()
               << std::endl;
}

// Our assumption was correct.
void non_qualified_function_parameter(ConstMemberFunctions& c) {
    c.set_val(42);
    std::cout << "For the value 42, the output is: " << c.calc() << '\n'
               << "If squared, it's: " << c.squared_calc()
               << std::endl;
}

// Our assumption was correct, in the ugliest possible way.
// Note that const correctness doesn't prevent encapsulation from intentionally being broken,
// it merely prevents code from having write access when it doesn't need it.
void one_const_one_not(const ConstMemberFunctions& l, ConstMemberFunctions& r) {
    // Let's just punch access modifiers and common sense in the face here.
    struct Machiavelli {
        int val;
        int unimportant;
        bool state_changed;
    };
    reinterpret_cast<Machiavelli&>(r).val = l.calc();
    reinterpret_cast<Machiavelli&>(r).state_changed = true;

    const_function_parameter(l);
    const_function_parameter(r);
}

// Our assumption was incorrect.

```

```
// Function fails to compile, due to `this` losing qualifiers in c.set_val().
void bad_parameter(const ConstMemberFunctions& c) {
    c.set_val(18);
}
```

`const` Machiavelli◦

```
class DealBreaker : public ConstMemberFunctions {
public:
    DealBreaker(int v = 0);

    // A foreboding name, but it's const...
    void no_guarantees() const;
}

DealBreaker::DealBreaker(int v /* = 0 */) : ConstMemberFunctions(v) {}

// Our assumption was incorrect.
// const_cast removes const-ness, making the compiler think we know what we're doing.
void DealBreaker::no_guarantees() const {
    const_cast<DealBreaker*>(this)->set_val(823);
}

// ...

const DealBreaker d(50);
d.no_guarantees(); // Undefined behaviour: d really IS const, it may or may not be modified.
```

`const const`◦

Const <https://riptutorial.com/zh-TW/cplusplus/topic/7217/const>

17: const

- `const myVariable = initial; //const;`
- `const Type myReference = myVariable; //const`
- `const Type * myPointer = myVariable; //const.`
- `* const myPointer = myVariable; //const.`
- `const Type * const myPointer = myVariable; //constconst.`

`const`¹ ◦ `const`.

`1const_cast`

Examples

Const

◦

```
// a is const int, so it can't be changed
const int a = 15;
a = 12;           // Error: can't assign new value to const variable
a += 1;          // Error: can't assign new value to const variable
```

```
int &b = a;        // Error: can't bind non-const reference to const variable
const int &c = a; // OK; c is a const reference

int *d = &a;      // Error: can't bind pointer-to-non-const to const variable
const int *e = &a // OK; e is a pointer-to-const

int f = 0;
e = &f;          // OK; e is a non-const pointer-to-const,
                // which means that it can be rebound to new int* or const int*

*e = 1           // Error: e is a pointer-to-const which means that
                // the value it points to can't be changed through dereferencing e

int *g = &f;
*g = 1;         // OK; this value still can be changed through dereferencing
                // a pointer-not-to-const
```

Const

```
int a = 0, b = 2;

const int* pA = &a; // pointer-to-const. `a` can't be changed through this
int* const pB = &a; // const pointer. `a` can be changed, but this pointer can't.
const int* const pC = &a; // const pointer-to-const.

//Error: Cannot assign to a const reference
*pA = b;
```

```

pA = &b;

*pB = b;

//Error: Cannot assign to const pointer
pB = &b;

//Error: Cannot assign to a const reference
*pC = b;

//Error: Cannot assign to const pointer
pC = &b;

```

Const

const

```

class MyClass
{
private:
    int myInt_;
public:
    int myInt() const { return myInt_; }
    void setMyInt(int myInt) { myInt_ = myInt; }
};

```

const thisconst MyClass *MyClass * ◦ ;◦ setMyIntconst ◦

const ◦ const MyClassconst◦

staticconst ◦ ;◦ staticconst◦

constconst getter◦

C++const◦ getter◦

FooBar

```

class Foo
{
public:
    Bar& GetBar(/* some arguments */)
    {
        /* some calculations */
        return bar;
    }

    const Bar& GetBar(/* some arguments */) const
    {
        /* some calculations */
        return bar;
    }

    // ...

```

```
};
```

constconstconstconst.

- **constconst.** **constconst.** **'const_cast'**const.

```
struct Foo
{
    Bar& GetBar(/*arguments*/)
    {
        return const_cast<Bar&>(const_cast<const Foo*>(this)->GetBar(/*arguments*/));
    }

    const Bar& GetBar(/*arguments*/) const
    {
        /* some calculations */
        return foo;
    }
};
```

constconst GetBar**const**GetBar const_cast<const Foo*>(this) ◦ **constconstconstconst.**

```
#include <iostream>

class Student
{
public:
    char& GetScore(bool midterm)
    {
        return const_cast<char&>(const_cast<const Student*>(this)->GetScore(midterm));
    }

    const char& GetScore(bool midterm) const
    {
        if (midterm)
        {
            return midtermScore;
        }
        else
        {
            return finalScore;
        }
    }

private:
    char midtermScore;
    char finalScore;
};

int main()
{
    // non-const object
    Student a;
    // We can assign to the reference. Non-const version of GetScore is called
    a.GetScore(true) = 'B';
    a.GetScore(false) = 'A';

    // const object
```



```
const Student b(a);  
// We still can call GetScore method of const object,  
// because we have overloaded const version of GetScore  
std::cout << b.GetScore(true) << b.GetScore(false) << '\n';  
}
```

const <https://riptutorial.com/zh-TW/cplusplus/topic/2386/const>

18: C

C++。

Examples

C++ C++。

```
int class = 5
```

。

C void* C++ C++C

```
void* ptr;  
int* intptr = ptr;
```

。

C++ goto switch C++

```
goto foo;  
int skipped = 1;  
foo;
```

。

[C https://riptutorial.com/zh-TW/cplusplus/topic/9645/c](https://riptutorial.com/zh-TW/cplusplus/topic/9645/c)

19: decltype

decltype°

Examples

°

```
int a = 10;

// Assume that type of variable 'a' is not known here, or it may
// be changed by programmer (from int to long long, for example).
// Hence we declare another variable, 'b' of the same type using
// decltype keyword.
decltype(a) b; // 'decltype(a)' evaluates to 'int'
```

“a”

```
float a=99.0f;
```

bfloat °

```
std::vector<int> intVector;
```

° auto ° °

```
vector<int>::iterator iter;
```

decltype(intVector)°

```
decltype(intVector)::iterator iter;
```

```
decltype(intVector.begin()) iter;
```

beginvector<int>::iterator °

const_iterator cbegin

```
decltype(intVector.cbegin()) iter; // vector<int>::const_iterator
```

decltype <https://riptutorial.com/zh-TW/cplusplus/topic/9930/decltype>

20: ISO C ++

1998 C ++。 C ++ C ++。 。 。

C ++ ""。

C ++。 Bjarne Stroustrup 90。 C ++ C ++。 。

C。 。 ANSI 1989。 ISO / ISO。 。

C ++。 ISO 22 WG 21。 1995。 C ++。 1995 STL WG 21 C ++。 3。 ISO C ++ / IEC 14882。 14882 1998。

。 C ++。 2003 14882 2003。 C ++;。

C ++ 0x2008 2009。 - 14882 2011。

WG 21。 C ++ 11 C ++。 3 14882 2014。

。 C ++ 17 C ++ 20。

Examples

ISO / ISO <http://www.iso.org>。 C ++。

- C ++ 20 C ++ 2a [HTML](#)
- C ++ 17 C ++ 1z [20173N4659](#)。
- C ++ 14 C ++ 1y [201411N4296](#)
- C ++ 11 C ++ 0x [20112N3242](#)
- C ++ 03
- C ++ 98

C ++ 11

C ++ 11 C ++。 [isocpp](#)。

-
- - [decltype](#)
 -
 -
 -
 - [Rvalue](#)
 - [Lambda](#)
 - [no](#)
 - [constexpr](#)

- [nullptr](#) -
-
-
-

- = default= delete
-
-
-
-
-
- final
-

-
- long long -
-
-
- POD

-
-
-
-

-
-
-

- `__cplusplus` ++ 11
-
-
-
- [static_assert](#)
-
-
-
- C99



- `unique_ptr`
- `shared_ptr`
- `weak_ptr`
- ABI
-
-
-
-
-
-
-
-
-

-
- unordered_*
- std::
- Modifiers

-
-
-
-
-
-
-
-

C++ 14

C++ 14 C++ 11。 C++ 11。 [isocpp](#)。

-
- -
 - decltype
 - [lambda](#)
 - [lambdas](#)
 -
 - constexpr
 - [\[\[deprecated\]\]](#)
 -

-
- - std::types
 - [std::make_unique](#)
 - _t
 - get<string>(t)
 - greater<>(x)
 - [std::quoted](#)

-
- [std::gets](#) C++ 11 C++ 14
 - [std::random_shuffle](#)

C++ 17

C++ 17。 C++ 1z。

-
-

- `auto`
-
-
-
-
- `[[fallthrough]]` `[[nodiscard]]` `[[maybe_unused]]`
- `static_assert`
- `ifswitchswitch`
-
- `if constexpr`
-
-

-
- `std::optional`
 - `std::variant`
 - `std::string_view`
 - `merge()` `extract()`
 - `<filesystem>` ◦
 - `<algorithm>` ◦
 - `<cmath>` ◦
 - `map` `<>unordered_map` `<>set` `<>unordered_set` `<>`

C ++ 03

C ++ 03 C ++ 98 ◦ ◦

-
-

C ++ 98

C ++ 98 C ++ ◦ C C ++ C ◦

C89 / C90

- `const`
-
- C-language C99
-
-
- C-language C99
-
-
-
-

-
-

C ++ 20

C ++ 20C ++C ++ 17。 [ISO cpp](#)。

2020C ++。

————

。

————

。

[ISO C ++](#) <https://riptutorial.com/zh-TW/cplusplus/topic/2742/iso-c-plusplus>

21: Lambda

- [*default-capture* *capture-list*] *argument-list* *mutable* *throw-specification* - > *return-type* { *lambda-body* } // lambda.
- [*capture-list*] *argument-list* { *lambda-body* } //lambda.
- [=] *argument-list* { *lambda-body* } //.
- [] *argument-list* { *lambda-body* } //.
- [*capture-list*] { *lambda-body* } //.

	<code>◦ = & ◦ <i>lambda</i> ◦ ◦</code>
	<code><i>lambda-body</i> ◦ ◦ & ◦ <i>this</i> ◦ ◦</code>
	<code>lambda ◦</code>
	<code>const ◦ mutableconst ◦ ◦</code>
	<code>lambda ◦ noexceptthrow (std::exception) ◦</code>
	<code>lambda ◦ <i>lambda-body</i> [[noreturn]] ◦</code>
- >	<code>lambda ◦ ◦</code>
	<code>lambda ◦</code>

C ++ 17 `constexpr lambda` `lambda` ◦ `lambda` `constexpr` `lambda` `constexpr` `constexpr`

```
//Explicitly define this lambda as constexpr
[] () constexpr {
    //Do stuff
}
```

Examples

lambda

`lambda` ◦ `lambda` `prvalue` ◦

“lambda” [lambda](#) Alonzo Church2030. [LambdaLISP LISP](#). [lambdaLISPC ++ lambda](#).

`lambda` ◦ ◦ `lambda` ◦

`lambda` [] () {}

```
[](){} // An empty lambda, which does and returns nothing
```

[] ◦ **lambda** ◦ **lambda** ◦ **lambda;lambda** ◦

```
int a = 0; // Define an integer variable
auto f = []() { return a*9; }; // Error: 'a' cannot be accessed
auto f = [a]() { return a*9; }; // OK, 'a' is "captured" by value
auto f = [&a]() { return a++; }; // OK, 'a' is "captured" by reference
// Note: It is the responsibility of the programmer
// to ensure that a is not destroyed before the
// lambda is called.
auto b = f(); // Call the lambda function. a is taken from the capture list
and not passed here.
```

() ◦ **lambda** **mutable** ◦ **lambda** **as**

```
auto call_foo = [x]() { x.foo(); };
auto call_foo2 = [x]{ x.foo(); };
```

C++ 14

auto ◦ **lambda** **as**

```
auto sort_cpp11 = [](std::vector<T>::const_reference lhs, std::vector<T>::const_reference rhs)
{ return lhs < rhs; };
auto sort_cpp14 = [](const auto &lhs, const auto &rhs) { return lhs < rhs; };
```

{}

lambda

lambda **operator()**

```
int multiplier = 5;
auto timesFive = [multiplier](int a) { return a * multiplier; };
std::out << timesFive(2); // Prints 10

multiplier = 15;
std::out << timesFive(2); // Still prints 2*5 == 10
```

lambda ◦

```
[]() { return true; };
```

bool ◦

```
[]() -> bool { return true; };
```

Lambda

lambda^o operator() const^o

```
auto func = [c = 0]() { ++c; std::cout << c; }; // fails to compile because ++c
// tries to mutate the state of
// the lambda.
```

mutable^ocloseoperator() const

```
auto func = [c = 0]() mutable { ++c; std::cout << c; };
```

mutable^o

```
auto func = [c = 0]() mutable -> int { ++c; std::cout << c; return c; };
```

lambda

C ++ 11

C ++ 11

```
// Generic functor used for comparison
struct islessthan
{
    islessthan(int threshold) : _threshold(threshold) {}

    bool operator()(int value) const
    {
        return value < _threshold;
    }
private:
    int _threshold;
};

// Declare a vector
const int arr[] = { 1, 2, 3, 4, 5 };
std::vector<int> vec(arr, arr+5);

// Find a number that's less than a given input (assume this would have been function input)
int threshold = 10;
std::vector<int>::iterator it = std::find_if(vec.begin(), vec.end(), islessthan(threshold));
```

C ++ 11

C ++ 11

```
// Declare a vector
std::vector<int> vec{ 1, 2, 3, 4, 5 };

// Find a number that's less than a given input (assume this would have been function input)
int threshold = 10;
auto it = std::find_if(vec.begin(), vec.end(), [threshold](int value) { return value <
```

```
threshold; });
```

return lambda as return

```
// Returns bool, because "value > 10" is a comparison which yields a Boolean result
auto l = [](int value) {
    return value > 10;
}
```

return lambda

```
// error: return types must match if lambda has unspecified return type
auto l = [](int value) {
    if (value < 10) {
        return 1;
    } else {
        return 1.5;
    }
};
```

```
// The return type is specified explicitly as 'double'
auto l = [](int value) -> double {
    if (value < 10) {
        return 1;
    } else {
        return 1.5;
    }
};
```

auto Lambdas

```
auto copy = [](X& x) { return x; }; // 'copy' returns an X, so copies its input
auto ref = [](X& x) -> X& { return x; }; // 'ref' returns an X&, no copy
```

lambda lambda

```
int a = 0;

[a]() {
    return a; // Ok, 'a' is captured by value
};
```

C++ 14

```
auto p = std::unique_ptr<T>(...);

[p]() { // Compile error; `unique_ptr` is not copy-constructible
    return p->createWidget();
};
```

C++ 14 lambda

C++ 14

```

auto p = std::make_unique<T>(...);

[p = std::move(p)]() {
    return p->createWidget();
};

```

lambda \circ **lambda**_{operator()} **const** \circ

const

```

int a = 0;

[a]() {
    a = 2; // Illegal, 'a' is accessed via `const`

    decltype(a) a1 = 1;
    a1 = 2; // valid: variable 'a1' is not const
};

```

const **lambda**_{mutable}

```

int a = 0;

[a]() mutable {
    a = 2; // OK, 'a' can be modified
    return a;
};

```

lambda_a \circ **lambda**_{aa} \circ

```

int a = 5 ;
auto plus5Val = [a] (void) { return a + 5 ; } ;
auto plus5Ref = [&a] (void) {return a + 5 ; } ;

a = 7 ;
std::cout << a << ", value " << plus5Val() << ", reference " << plus5Ref() ;
// The result will be "7, value 10, reference 12"

```

C++ 14

Lambdas \circ **lambdas**

```

auto p = std::make_unique<T>(...);

auto lamb = [p = std::move(p)]() //Overrides capture-by-value of `p`.
{
    p->SomeFunc();
};

```

lambda_p \circ **lambda**_{make_unique} \circ **lambda** \circ

```

auto lamb_copy = lamb; //Illegal
auto lamb_move = std::move(lamb); //legal.

```

lamb_move◦

std::function<>◦ `std::function` **lambda** shared_ptr

```
auto shared_lambda = [](auto&& f){
    return [spf = std::make_shared<std::decay_t<decltype(f)>>(decltype(f)(f))]
        (auto&&...args)->decltype(auto) {
        return (*spf)(decltype(args)(args)...);
    };
};
auto lamb_shared = shared_lambda(std::move(lamb_move));
```

move-only lambda `lambda` std::function◦

auto◦

```
int a = 0;

auto lamb = [&v = a](int add) //Note that `a` and `v` have different names
{
    v += add; //Modifies `a`
};

lamb(20); //`a` becomes 20.
```

Generalize◦

```
auto lamb = [p = std::make_unique<T>(...)]()
{
    p->SomeFunc();
}
```

lambda `lambda`◦ `lambda`◦

&◦ `lambda` `lambda`◦ `lambda`

```
// Declare variable 'a'
int a = 0;

// Declare a lambda which captures 'a' by reference
auto set = [&a]() {
    a = 1;
};

set();
assert(a == 1);
```

mutable a const ◦

◦ `lambda` ◦ `lambda`◦

lambda lambda

```
int a = 1;
int b = 2;

// Default capture by value
[=]() { return a + b; }; // OK; a and b are captured by value

// Default capture by reference
[&]() { return a + b; }; // OK; a and b are captured by reference
```

o

```
int a = 0;
int b = 1;

[=, &b]() {
    a = 2; // Illegal; 'a' is capture by value, and lambda is not 'mutable'
    b = 2; // OK; 'b' is captured by reference
};
```

lambdas

C++ 14

Lambda lambda

```
auto twice = [](auto x){ return x+x; };

int i = twice(2); // i == 4
std::string s = twice("hello"); // s == "hellohello"
```

operator() C++ lambda

```
struct _unique_lambda_type
{
    template<typename T>
    auto operator() (T x) const {return x + x;}
};
```

lambda

```
[](auto x, int y) {return x + y;}
```

xyint o

lambda auto& auto&& rvalue

```
auto lamb1 = [](int &&x) {return x + 5;};
auto lamb2 = [](auto &&x) {return x + 5;};
int x = 10;
lamb1(x); // Illegal; must use `std::move(x)` for `int&&` parameters.
```

```
lambda2(x); // Legal; the type of `x` is deduced as `int&`.
```

Lambda

```
auto lam = [](auto&&... args){return f(std::forward<decltype(args)>(args)...)};
```

```
auto lam = [](auto&&... args){return f(decltype(args)(args)...)};
```

auto&&""。

lambda。

```
boost::variant<int, double> value;  
apply_visitor(value, [&](auto&& e){  
    std::cout << e;  
});
```

;

```
mutex_wrapped<std::ostream&> os = std::cout;  
os.write([&](auto&& os){  
    os << "hello world\n";  
});
```

std::ostream&std::ostream&;。 ; autofor(:)auto。

lambdalambda

```
auto sorter = [](int lhs, int rhs) -> bool {return lhs < rhs;};
```

```
using func_ptr = bool (*)(int, int);  
func_ptr sorter_func = sorter; // implicit conversion
```

```
func_ptr sorter_func2 = +sorter; // enforce implicit conversion
```

lambdaoperator()。 lambda。 lambda。

lambdaAPIC ++。

C ++ 14

lambda。 。

```
auto sorter = [](auto lhs, auto rhs) { return lhs < rhs; };  
using func_ptr = bool (*)(int, int);  
func_ptr sorter_func = sorter; // deduces int, int  
// note however that the following is ambiguous  
// func_ptr sorter_func2 = +sorter;
```


lambda

lambda

```
class Foo
{
private:
    int i;

public:
    Foo(int val) : i(val) {}

    // definition of a member function
    void Test()
    {
        auto lamb = [](Foo &foo, int val)
        {
            // modification of a private member variable
            foo.i = val;
        };

        // lamb is allowed to access a private member, because it is a friend of Foo
        lamb(*this, 30);
    }
};
```

lambda

Lambdas

```
class Foo
{
private:
    int i;

public:
    Foo(int val) : i(val) {}

    void Test()
    {
        // capture the this pointer by value
        auto lamb = [this](int val)
        {
            i = val;
        };

        lamb(30);
    }
};
```

this this->

this this lambda this lambda outlambda

lambda thismutable const const

```
[&] this ◦ ◦ this ◦
```

C++ 17

Lambda `this ◦ *this`

```
class Foo
{
private:
    int i;

public:
    Foo(int val) : i(val) {}

    void Test()
    {
        // capture a copy of the object given by the this pointer
        auto lamb = [*this](int val) mutable
        {
            i = val;
        };

        lamb(30); // does not change this->i
    }
};
```

lambda C++ 03

C++ Lambda ◦ lambda C++ 03

```
// Some dummy types:
struct T1 {int dummy;};
struct T2 {int dummy;};
struct R {int dummy;};

// Code using a lambda function (requires C++11)
R use_lambda(T1 val, T2 ref) {
    // Use auto because the type of the lambda is unknown.
    auto lambda = [val, &ref](int arg1, int arg2) -> R {
        /* lambda-body */
        return R();
    };
    return lambda(12, 27);
}

// The functor class (valid C++03)
// Similar to what the compiler generates for the lambda function.
class Functor {
    // Capture list.
    T1 val;
    T2& ref;

public:
    // Constructor
    inline Functor(T1 val, T2& ref) : val(val), ref(ref) {}

    // Functor body
```

```

R operator()(int arg1, int arg2) const {
    /* lambda-body */
    return R();
}
};

// Equivalent to use_lambda, but uses a functor (valid C++03).
R use_functor(T1 val, T2 ref) {
    Functor functor(val, ref);
    return functor(12, 27);
}

// Make this a self-contained example.
int main() {
    T1 t1;
    T2 t2;
    use_functor(t1,t2);
    use_lambda(t1,t2);
    return 0;
}

```

lambda mutable functor call-operator const

```

R operator()(int arg1, int arg2) /*non-const*/ {
    /* lambda-body */
    return R();
}

```

lambdas

Euclid gcd() lambda

```

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a%b);
}

```

lambda lambda lambda this this lambda

`std::function`

lambda `std::function`

```

std::function<int(int, int)> gcd = [&](int a, int b){
    return b == 0 ? a : gcd(b, a%b);
};

```

gcd gcd lambda lambda lambda

```

auto gcd_self = std::make_shared<std::unique_ptr< std::function<int(int, int)> >>();
* gcd_self = std::make_unique<std::function<int(int, int)>>(
    [gcd_self](int a, int b){
        return b == 0 ? a : (**gcd_self)(b, a%b);
    });
};

```

/◦ lambda◦

Y

```
template <class F>
struct y_combinator {
    F f; // the lambda will be stored here

    // a forwarding operator():
    template <class... Args>
    decltype(auto) operator()(Args&&... args) const {
        // we pass ourselves to f, then the arguments.
        // the lambda should take the first argument as `auto&& recurse` or similar.
        return f(*this, std::forward<Args>(args)...);
    }
};

// helper function that deduces the type of the lambda:
template <class F>
y_combinator<std::decay_t<F>> make_y_combinator(F&& f) {
    return {std::forward<F>(f)};
}

// (Be aware that in C++17 we can do better than a `make_` function)
```

gcd

```
auto gcd = make_y_combinator(
    [](auto&& gcd, int a, int b){
        return b == 0 ? a : gcd(b, a%b);
    }
);
```

y_combinator lambda◦ lambda◦

lambda“recurse”◦ recurse◦

y_combinator“recurse”y_combinator◦ y_combinator lambda◦

```
auto foo = make_y_combinator( [&](auto&& recurse, some arguments) {
    // write body that processes some arguments
    // when you want to recurse, call recurse(some other arguments)
});
```

lambda◦

lambdas

C++ 14

◦

```
template<std::size_t...Is>
void print_indexes( std::index_sequence<Is...> ) {
    using discard=int[];
```

```

(void)discard{0, ((void) (
    std::cout << Is << '\n' // here Is is a compile-time constant.
),0)...};
}
template<std::size_t I>
void print_indexes_upto() {
    return print_indexes( std::make_index_sequence<I>{} );
}

```

print_indexes_uptoprint_indexes_upto° ° °

lambdas°

lambda

```

template<std::size_t I>
using index_t = std::integral_constant<std::size_t, I>;
template<std::size_t I>
constexpr index_t<I> index{};

template<class=void, std::size_t...Is>
auto index_over( std::index_sequence<Is...> ) {
    return [] (auto&& f) {
        using discard=int[];
        (void)discard{0, (void(
            f( index<Is> )
        ),0)...};
    };
}

template<std::size_t N>
auto index_over(index_t<N> = {}) {
    return index_over( std::make_index_sequence<N>{} );
}

```

C++ 17

index_over()

```

template<class=void, std::size_t...Is>
auto index_over( std::index_sequence<Is...> ) {
    return [] (auto&& f) {
        ((void) (f(index<Is>)), ...);
    };
}

```

“inline”

```

template<class Tup, class F>
void for_each_tuple_element(Tup&& tup, F&& f) {
    using T = std::remove_reference_t<Tup>;
    using std::tuple_size;
    auto from_zero_to_N = index_over< tuple_size<T>{} >();

    from_zero_to_N(
        [&] (auto i) {
            using std::get;

```

```
    f( get<i>( std::forward<Tup>(tup) ) );  
  }  
);  
}
```

auto iindex_overstd::integral_constant<std::size_t, ???>constexprstd::size_tthis std::get<i>◦

```
template<std::size_t I>  
void print_indexes_upto() {  
  index_over(index<I>) ([](auto i){  
    std::cout << i << '\n'; // here i is a compile-time constant  
  });  
}
```

◦

◦

Lambda <https://riptutorial.com/zh-TW/cplusplus/topic/572/lambda>

22: Pimpl

PIMPL ointerIMPL ementation cpp◦

◦ #include◦

pimpl◦

Examples

Pimpl

C ++ 11

```
// widget.h

#include <memory> // std::unique_ptr
#include <experimental/propagate_const>

class Widget
{
public:
    Widget();
    ~Widget();
    void DoSomething();

private:
    // the pImpl idiom is named after the typical variable name used
    // ie, pImpl:
    struct Impl; // forward declaration
    std::experimental::propagate_const<std::unique_ptr< Impl >> pImpl; // ptr to actual
    implementation
};
```

```
// widget.cpp

#include "widget.h"
#include "reallycomplextypes.h" // no need to include this header inside widget.h

struct Widget::Impl
{
    // the attributes needed from Widget go here
    ReallyComplexType rct;
};

Widget::Widget() :
    pImpl(std::make_unique<Impl>())
{}

Widget::~~Widget() = default;

void Widget::DoSomething()
{
    // do the stuff here with pImpl
```

```
}
```

```
pImplWidget/◦ Widget◦
```

```
pImpl“◦ Widget“pImpl◦
```

```
unique_ptr Impl~Widget () ◦ =default=default Impl◦
```

Pimpl <https://riptutorial.com/zh-TW/cplusplus/topic/2143/pimpl>

23: RAII

RAII resource acquisition initialization. SBRMRRIDRAII. C++ - .

◦ ◦

RAII. exit. ◦ pid.

unixexec-familyfork-exec. RAIIexec. exec. fork.

Examples

```
std::mutex mtx;

void bad_lock_example() {
    mtx.lock();
    try
    {
        foo();
        bar();
        if (baz()) {
            mtx.unlock(); // Have to unlock on each exit point.
            return;
        }
        quux();
        mtx.unlock(); // Normal unlock happens here.
    }
    catch(...) {
        mtx.unlock(); // Must also force unlock in the presence of
        throw; // exceptions and allow the exception to continue.
    }
}
```

◦ unlock()unlock()◦ ◦

RAII

```
std::mutex mtx;

void good_lock_example() {
    std::lock_guard<std::mutex> lk(mtx); // constructor locks.
                                        // destructor unlocks. destructor call
                                        // guaranteed by language.

    foo();
    bar();
    if (baz()) {
        return;
    }
    quux();
}
```

lock_guard lock_guardlock() unlock() ◦ lock_guardmutex ◦ - ;◦

/ ScopeExit

```
template<typename Function>
class Finally final
{
public:
    explicit Finally(Function f) : f(std::move(f)) {}
    ~Finally() { f(); } // (1) See below

    Finally(const Finally&) = delete;
    Finally(Finally&&) = default;
    Finally& operator =(const Finally&) = delete;
    Finally& operator =(Finally&&) = delete;
private:
    Function f;
};
// Execute the function f when the returned object goes out of scope.
template<typename Function>
auto onExit(Function &&f) { return Finally<std::decay_t<Function>>{std::forward<Function>(f)};
}
```

```
void foo(std::vector<int>& v, int i)
{
    // ...

    v[i] += 42;
    auto autoRollBackChange = onExit([&](){ v[i] -= 42; });

    // ... code as recursive call `foo(v, i + 1)`
}
```

1

- ~Finally() noexcept { f(); } std::terminate
- ~Finally() noexcept(noexcept(f())) { f(); } **terminate**◦
- ~Finally() noexcept { try { f(); } catch (...) { /* ignore exception (might log it) */ } }
std::terminate ◦

ScopeSuccessc ++ 17

C ++ 17

int std::uncaught_exceptions() ◦ bool std::uncaught_exception()◦

```
#include <exception>
#include <iostream>

template <typename F>
class ScopeSuccess
{
private:
    F f;
    int uncaughtExceptionCount = std::uncaught_exceptions();
public:
    explicit ScopeSuccess(const F& f) : f(f) {}
}
```

```

ScopeSuccess(const ScopeSuccess&) = delete;
ScopeSuccess& operator =(const ScopeSuccess&) = delete;

// f() might throw, as it can be caught normally.
~ScopeSuccess() noexcept(noexcept(f())) {
    if (uncaughtExceptionCount == std::uncaught_exceptions()) {
        f();
    }
}
};

struct Foo {
    ~Foo() {
        try {
            ScopeSuccess logSuccess{[]() {std::cout << "Success 1\n";}};
            // Scope succeeds,
            // even if Foo is destroyed during stack unwinding
            // (so when 0 < std::uncaught_exceptions())
            // (or previously std::uncaught_exception() == true)
        } catch (...) {
        }
        try {
            ScopeSuccess logSuccess{[]() {std::cout << "Success 2\n";}};

            throw std::runtime_error("Failed"); // returned value
                                                // of std::uncaught_exceptions increases
        } catch (...) { // returned value of std::uncaught_exceptions decreases
        }
    }
};

int main()
{
    try {
        Foo foo;

        throw std::runtime_error("Failed"); // std::uncaught_exceptions() == 1
    } catch (...) { // std::uncaught_exceptions() == 0
    }
}

```

```
Success 1
```

ScopeFailc ++ 17

C ++ 17

```
int std::uncaught_exceptions() ◦ bool std::uncaught_exception()◦
```

```

#include <exception>
#include <iostream>

template <typename F>
class ScopeFail
{
private:
    F f;

```

```

    int uncaughtExceptionCount = std::uncaught_exceptions();
public:
    explicit ScopeFail(const F& f) : f(f) {}
    ScopeFail(const ScopeFail&) = delete;
    ScopeFail& operator =(const ScopeFail&) = delete;

    // f() should not throw, else std::terminate is called.
    ~ScopeFail() {
        if (uncaughtExceptionCount != std::uncaught_exceptions()) {
            f();
        }
    }
};

struct Foo {
    ~Foo() {
        try {
            ScopeFail logFailure{[](){std::cout << "Fail 1\n";}};
            // Scope succeeds,
            // even if Foo is destroyed during stack unwinding
            // (so when 0 < std::uncaught_exceptions())
            // (or previously std::uncaught_exception() == true)
        } catch (...) {
        }
        try {
            ScopeFail logFailure{[](){std::cout << "Failure 2\n";}};

            throw std::runtime_error("Failed"); // returned value
                                                // of std::uncaught_exceptions increases
        } catch (...) { // returned value of std::uncaught_exceptions decreases
        }
    }
};

int main()
{
    try {
        Foo foo;

        throw std::runtime_error("Failed"); // std::uncaught_exceptions() == 1
    } catch (...) { // std::uncaught_exceptions() == 0
    }
}

```

Failure 2

RAII <https://riptutorial.com/zh-TW/cplusplus/topic/1320/raii->

24: RTTI

Examples

`.name()` `std::type_info` `typeid`

```
#include <iostream>
#include <typeinfo>

int main()
{
    int speed = 110;

    std::cout << typeid(speed).name() << '\n';
}
```

```
int
```

dynamic_cast

`dynamic_cast<>()`

BCclass A

```
class A { public: virtual ~A(){} };

class B: public A
{ public: void work4B(){} };

class C: public A
{ public: void work4C(){} };

void non_polymorphic_work(A* ap)
{
    if (B* bp =dynamic_cast<B*>(ap))
        bp->work4B();
    if (C* cp =dynamic_cast<C*>(ap))
        cp->work4C();
}
```

typeid

`typeid` `const std::type_info` **CV**

```
struct Base {
    virtual ~Base() = default;
};
struct Derived : Base {};
Base* b = new Derived;
assert(typeid(*b) == typeid(Derived{})); // OK
```

`typeid` ◦ **cv-qualification** ◦ `typeid(Derived)` `typeid(Derived{})`

```
assert(typeid(*b) == typeid(Derived{})); // OK
```

`typeid` **info** ◦

```
struct Base {
    // note: no virtual destructor
};
struct Derived : Base {};
Derived d;
Base& b = d;
assert(typeid(b) == typeid(Base)); // not Derived
assert(typeid(std::declval<Base>()) == typeid(Base)); // OK because unevaluated
```

C ++

dynamic_cast ◦

static_cast ◦

reinterpret_cast ◦ ◦

const_cast `const / volatile` ◦ `const API` ◦

RTTI <https://riptutorial.com/zh-TW/cplusplus/topic/3129/rtti->

25: SFINAE

Examples

enable_if

std::enable_ifSFINAE。

```
template <bool Cond, typename Result=void>
struct enable_if { };

template <typename Result>
struct enable_if<true, Result> {
    using type = Result;
};
```

enable_if<true, R>::typeRenable_if<false, T>::typeenable_if<false, T>::typeenable_iftype。

std::enable_if

```
int negate(int i) { return -i; }

template <class F>
auto negate(F f) { return -f(); }
```

negate(1)。

```
int negate(int i) { return -i; }

template <class F, class = typename std::enable_if<!std::is_arithmetic<F>::value>::type>
auto negate(F f) { return -f(); }
```

negate<int>!std::is_arithmetic<int>::valuefalse。 SFINAE。 negate(1) -。

std::enable_ifSFINAE SFINAE。 std::size size(arg)

```
// for containers
template<typename Cont>
auto size1(Cont const& cont) -> decltype( cont.size() );

// for arrays
template<typename Elt, std::size_t Size>
std::size_t size1(Elt const(&arr)[Size]);

// implementation omitted
template<typename Cont>
struct is_sizeable;

// for containers
template<typename Cont, std::enable_if_t<std::is_sizeable<Cont>::value, int> = 0>
```

```

auto size2(Cont const& cont);

// for arrays
template<typename Elt, std::size_t Size>
std::size_t size2(Elt const(&arr)[Size]);

```

is_sizeable **SFINAE**.

◦ `incr(i, 3)` `i += 3` `INT_MAX` `int` ◦

```

// handle signed types
template<typename Int>
auto incr1(Int& target, Int amount)
-> std::void_t<int[static_cast<Int>(-1) < static_cast<Int>(0)]>;

// handle unsigned types by just doing target += amount
// since unsigned arithmetic already behaves as intended
template<typename Int>
auto incr1(Int& target, Int amount)
-> std::void_t<int[static_cast<Int>(0) < static_cast<Int>(-1)]>;

template<typename Int, std::enable_if_t<std::is_signed<Int>::value, int> = 0>
void incr2(Int& target, Int amount);

template<typename Int, std::enable_if_t<std::is_unsigned<Int>::value, int> = 0>
void incr2(Int& target, Int amount);

```

`std::enable_if` **API** ◦ `is_sizeable<Cont>::value` `cont.size()` `size_t` `is_sizeable` ◦ `std::is_signed` `incr1` ◦

void_t

C++ 11

`void_t` ◦ `void_t` ◦

`std::void_t` **C++ 17**

```

template <class...> using void_t = void;

```

```

template <class...>
struct make_void { using type = void; };

template <typename... T>
using void_t = typename make_void<T...>::type;

```

`void_t` ◦ `foo()`

```

template <class T, class=void>
struct has_foo : std::false_type {};

template <class T>
struct has_foo<T, void_t<decltype(std::declval<T>().foo())>> : std::true_type {};

```


has_foo<T>::value has_foo<T, void>has_foo<T, void> ◦

- T foo() void ◦ has_foo<T>::valuetrue
- T ◦ has_foo<T>::valuefalse ◦

```
template<class T, class=void>
struct can_reference : std::false_type {};

template<class T>
struct can_reference<T, std::void_t<T&>> : std::true_type {};
```

std::declvaldecltype ◦

void ◦

```
struct details {
    template<template<class...>class Z, class=void, class...Ts>
    struct can_apply:
        std::false_type
    {};
    template<template<class...>class Z, class...Ts>
    struct can_apply<Z, std::void_t<Z<Ts...>>, Ts...>:
        std::true_type
    {};
};

template<template<class...>class Z, class...Ts>
using can_apply = details::can_apply<Z, void, Ts...>;
```

std::void_tcan_apply ◦ can_apply

```
template<class T>
using ref_t = T&;

template<class T>
using can_reference = can_apply<ref_t, T>;    // Is T& well formed for T?
```

```
template<class T>
using dot_foo_r = decltype(std::declval<T&>().foo());

template<class T>
using can_dot_foo = can_apply< dot_foo_r, T >;    // Is T.foo() well formed for T?
```

◦

std**C ++ 17**can_apply ◦

void_tWalter Brown ◦ CppCon 2016 ◦

decltype

C ++ 11

decltype

```
namespace details {
    using std::to_string;

    // this one is constrained on being able to call to_string(T)
    template <class T>
    auto convert_to_string(T const& val, int )
        -> decltype(to_string(val))
    {
        return to_string(val);
    }

    // this one is unconstrained, but less preferred due to the ellipsis argument
    template <class T>
    std::string convert_to_string(T const& val, ... )
    {
        std::ostringstream oss;
        oss << val;
        return oss.str();
    }
}

template <class T>
std::string convert_to_string(T const& val)
{
    return details::convert_to_string(val, 0);
}
```

to_string() convert_to_string() details::convert_to_string() ◦ 0int0.....

to_string() convert_to_string() decltype(to_string(val)) ◦ ◦ operator<<(std::ostream&, T) ◦ oss << val◦

SFINAE

SFINAES ubstitution **F** ailure **I** s **N** ot **A** n **E** rror◦ ◦

- ◦

```
template <class T>
auto begin(T& c) -> decltype(c.begin()) { return c.begin(); }

template <class T, size_t N>
T* begin(T (&arr)[N]) { return arr; }

int vals[10];
begin(vals); // OK. The first function template substitution fails because
              // vals.begin() is ill-formed. This is not an error! That function
              // is just removed from consideration as a viable overload candidate,
              // leaving us with the array overload.
```

◦

```
template <class T>
void add_one(T& val) { val += 1; }
```

```
int i = 4;
add_one(i); // ok

std::string msg = "Hello";
add_one(msg); // error. msg += 1 is ill-formed for std::string, but this
              // failure is NOT in the immediate context of substituting T
```

enable_if_all / enable_if_any

C++ 11

```
template<typename ...Args> void func(Args &&...args) { //... };
```

C++ 17 `enable_if` SFINAE `Args Args` ◦ C++ 17 `std::conjunction` `std::disjunction` ◦

```
/// C++17: SFINAE constraints on all of the parameters in Args.
template<typename ...Args,
        std::enable_if_t<std::conjunction_v<custom_conditions_v<Args>...>>* = nullptr>
void func(Args &&...args) { //... };
```

```
/// C++17: SFINAE constraints on any of the parameters in Args.
template<typename ...Args,
        std::enable_if_t<std::disjunction_v<custom_conditions_v<Args>...>>* = nullptr>
void func(Args &&...args) { //... };
```

C++ 17. ◦

`std::conjunction` `std::disjunction` ◦ `std::enable_if` `enable_if_all` `enable_if_any` ◦ ◦

`enable_if_all` `enable_if_any`

`seq_and` `seq_or` `std::conjunction` `std::disjunction`

```
/// Helper for prior to C++14.
template<bool B, class T, class F >
using conditional_t = typename std::conditional<B,T,F>::type;

/// Emulate C++17 std::conjunction.
template<bool...> struct seq_or: std::false_type {};
template<bool...> struct seq_and: std::true_type {};

template<bool B1, bool... Bs>
struct seq_or<B1,Bs...>:
    conditional_t<B1,std::true_type,seq_or<Bs...>> {};

template<bool B1, bool... Bs>
struct seq_and<B1,Bs...>:
    conditional_t<B1,seq_and<Bs...>,std::false_type> {};
```

```

template<bool... Bs>
using enable_if_any = std::enable_if<seq_or<Bs...>::value>;

template<bool... Bs>
using enable_if_all = std::enable_if<seq_and<Bs...>::value>;

```

```

template<bool... Bs>
using enable_if_any_t = typename enable_if_any<Bs...>::type;

template<bool... Bs>
using enable_if_all_t = typename enable_if_all<Bs...>::type;

```

```

// SFINAE constraints on all of the parameters in Args.
template<typename ...Args,
         enable_if_all_t<custom_conditions_v<Args>...>* = nullptr>
void func(Args &&...args) { //... };

// SFINAE constraints on any of the parameters in Args.
template<typename ...Args,
         enable_if_any_t<custom_conditions_v<Args>...>* = nullptr>
void func(Args &&...args) { //... };

```

is_detected

`type_traitSFINAE` `detected_or` `detected_t` `is_detected` ◦

`typename Default` `template <typename...>` `Op` `typename ... Args`

- `is_detected` `std::true_type` `std::false_type` `Op<Args...>`
- `detected_t` `Op<Args...>` `nonesuch` `Op<Args...>`
- `detected_or` `value_t` `is_detected` `type` `Op<Args...>` `Default` `Op<Args...>`

`std::void_t` **SFINAE**

C++ 17

```

namespace detail {
    template <class Default, class AlwaysVoid,
             template<class...> class Op, class... Args>
    struct detector
    {
        using value_t = std::false_type;
        using type = Default;
    };

    template <class Default, template<class...> class Op, class... Args>
    struct detector<Default, std::void_t<Op<Args...>>, Op, Args...>
    {
        using value_t = std::true_type;
        using type = Op<Args...>;
    };
} // namespace detail

```

```

// special type to indicate detection failure
struct nonesuch {
    nonesuch() = delete;
    ~nonesuch() = delete;
    nonesuch(nonesuch const&) = delete;
    void operator=(nonesuch const&) = delete;
};

template <template<class...> class Op, class... Args>
using is_detected =
    typename detail::detector<nonesuch, void, Op, Args...>::value_t;

template <template<class...> class Op, class... Args>
using detected_t = typename detail::detector<nonesuch, void, Op, Args...>::type;

template <class Default, template<class...> class Op, class... Args>
using detected_or = detail::detector<Default, void, Op, Args...>;

```

```

typename <typename T, typename ...Ts>
using foo_type = decltype(std::declval<T>().foo(std::declval<Ts>()...));

struct C1 {};

struct C2 {
    int foo(char) const;
};

template <typename T>
using has_foo_char = is_detected<foo_type, T, char>;

static_assert(!has_foo_char<C1>::value, "Unexpected");
static_assert(has_foo_char<C2>::value, "Unexpected");

static_assert(std::is_same<int, detected_t<foo_type, C2, char>>::value,
              "Unexpected");

static_assert(std::is_same<void, // Default
              detected_or<void, foo_type, C1, char>>::value,
              "Unexpected");
static_assert(std::is_same<int, detected_or<void, foo_type, C2, char>>::value,
              "Unexpected");

```

enable_if<>°

°

decltype°

" - random_access_tag° °

```

#include <algorithm>
#include <iterator>

namespace detail
{
    // this gives us infinite types, that inherit from each other
    template<std::size_t N>

```

```

struct pick : pick<N-1> {};
template<>
struct pick<0> {};

// the overload we want to be preferred have a higher N in pick<N>
// this is the first helper template function
template<typename T>
auto stable_sort(T& t, pick<2>)
    -> decltype( t.stable_sort(), void() )
{
    // if the container have a member stable_sort, use that
    t.stable_sort();
}

// this helper will be second best match
template<typename T>
auto stable_sort(T& t, pick<1>)
    -> decltype( t.sort(), void() )
{
    // if the container have a member sort, but no member stable_sort
    // it's customary that the sort member is stable
    t.sort();
}

// this helper will be picked last
template<typename T>
auto stable_sort(T& t, pick<0>)
    -> decltype( std::stable_sort(std::begin(t), std::end(t)), void() )
{
    // the container have neither a member sort, nor member stable_sort
    std::stable_sort(std::begin(t), std::end(t));
}
}

// this is the function the user calls. it will dispatch the call
// to the correct implementation with the help of 'tags'.
template<typename T>
void stable_sort(T& t)
{
    // use an N that is higher than any used above.
    // this will pick the highest overload that is well formed.
    detail::stable_sort(t, detail::pick<10>{});
}

```

◦

tag-dispatch◦

SFINAE <https://riptutorial.com/zh-TW/cplusplus/topic/1169/sfinae-->

26: static_assert

- `static_assert bool_constexpr message`
- `static_assert bool_constexpr / *C ++ 17 * /`

<code>bool_constexpr</code>	
	<code>bool_constexprfalse</code>

◦

Examples

static_assert

◦ `static_assert()` ◦

```
template<typename T>
T mul10(const T t)
{
    static_assert( std::is_integral<T>::value, "mul10() only works for integral types" );
    return (t << 3) + (t << 1);
}
```

`static_assert()` `bool constexpr` ◦ ◦ `C ++ 17`;

C ++ 17

```
template<typename T>
T mul10(const T t)
{
    static_assert( std::is_integral<T>::value );
    return (t << 3) + (t << 1);
}
```

- `constexpr`
- ◦
 - `constexpr`
 -
- `C ++`
- `sizeof(T) sizeof(T) 32int`
-

`static_assert()` `SFINAE` `/std::enable_if<>` ◦ `/` ◦ `SFINAE` ◦

`static_assert` <https://riptutorial.com/zh-TW/cplusplus/topic/3822/static-assert>

27: std :: function

Examples

```
#include <iostream>
#include <functional>
std::function<void(int , const std::string&)> myFuncObj;
void theFunc(int i, const std::string& s)
{
    std::cout << s << ": " << i << std::endl;
}
int main(int argc, char *argv[])
{
    myFuncObj = theFunc;
    myFuncObj(10, "hello world");
}
```

std :: functionstd :: bind

- std::bindstd::function◦

```
class A
{
public:
    std::function<void(int, const std::string&)> m_CbFunc = nullptr;
    void foo()
    {
        if (m_CbFunc)
        {
            m_CbFunc(100, "event fired");
        }
    }
};

class B
{
public:
    B()
    {
        auto aFunc = std::bind(&B::eventHandler, this, std::placeholders::_1,
std::placeholders::_2);
        anObjA.m_CbFunc = aFunc;
    }
    void eventHandler(int i, const std::string& s)
    {
        std::cout << s << ": " << i << std::endl;
    }

    void DoSomethingOnA()
    {
        anObjA.foo();
    }

    A anObjA;
};
```



```
};

int main(int argc, char *argv[])
{
    B anObjB;
    anObjB.DoSomethingOnA();
}
```

std :: function with lambda std :: bind

```
#include <iostream>
#include <functional>

using std::placeholders::_1; // to be used in std::bind example

int stdf_foobar (int x, std::function<int(int)> moo)
{
    return x + moo(x); // std::function moo called
}

int foo (int x) { return 2+x; }

int foo_2 (int x, int y) { return 9*x + y; }

int main()
{
    int a = 2;

    /* Function pointers */
    std::cout << stdf_foobar(a, &foo) << std::endl; // 6 ( 2 + (2+2) )
    // can also be: stdf_foobar(2, foo)

    /* Lambda expressions */
    /* An unnamed closure from a lambda expression can be
     * stored in a std::function object:
     */
    int capture_value = 3;
    std::cout << stdf_foobar(a,
                            [capture_value](int param) -> int { return 7 + capture_value *
param; })
                << std::endl;
    // result: 15 == value + (7 * capture_value * value) == 2 + (7 + 3 * 2)

    /* std::bind expressions */
    /* The result of a std::bind expression can be passed.
     * For example by binding parameters to a function pointer call:
     */
    int b = stdf_foobar(a, std::bind(foo_2, _1, 3));
    std::cout << b << std::endl;
    // b == 23 == 2 + ( 9*2 + 3 )
    int c = stdf_foobar(a, std::bind(foo_2, 5, _1));
    std::cout << c << std::endl;
    // c == 49 == 2 + ( 9*5 + 2 )

    return 0;
}
```

`function`

std::function< std::function<value semantics> [1]>> callable.

function<...> noexcept C++

```
//Header file
using MyPredicate = std::function<bool(const MyValue &, const MyValue &)>;

void SortMyContainer(MyContainer &C, const MyPredicate &pred);

//Source file
void SortMyContainer(MyContainer &C, const MyPredicate &pred)
{
    std::sort(C.begin(), C.end(), pred);
}
```

SortMyContainer SortMyContainer pred pred

function;/; function callable SortMyContainer; function;

o o

function allocator [.....] [2]

C++ 17

allocator function function o

function function function function

std::function

```
/*
 * This example show some ways of using std::function to call
 * a) C-like function
 * b) class-member function
 * c) operator()
 * d) lambda function
 *
 * Function call can be made:
 * a) with right arguments
 * b) argumens with different order, types and count
 */
#include <iostream>
#include <functional>
#include <iostream>
#include <vector>

using std::cout;
using std::endl;
using namespace std::placeholders;

// simple function to be called
double foo_fn(int x, float y, double z)
```

```

{
    double res = x + y + z;
    std::cout << "foo_fn called with arguments: "
              << x << ", " << y << ", " << z
              << " result is : " << res
              << std::endl;
    return res;
}

// structure with member function to call
struct foo_struct
{
    // member function to call
    double foo_fn(int x, float y, double z)
    {
        double res = x + y + z;
        std::cout << "foo_struct::foo_fn called with arguments: "
                  << x << ", " << y << ", " << z
                  << " result is : " << res
                  << std::endl;
        return res;
    }
    // this member function has different signature - but it can be used too
    // please not that argument order is changed too
    double foo_fn_4(int x, double z, float y, long xx)
    {
        double res = x + y + z + xx;
        std::cout << "foo_struct::foo_fn_4 called with arguments: "
                  << x << ", " << z << ", " << y << ", " << xx
                  << " result is : " << res
                  << std::endl;
        return res;
    }
    // overloaded operator() makes whole object to be callable
    double operator()(int x, float y, double z)
    {
        double res = x + y + z;
        std::cout << "foo_struct::operator() called with arguments: "
                  << x << ", " << y << ", " << z
                  << " result is : " << res
                  << std::endl;
        return res;
    }
};

int main(void)
{
    // typedefs
    using function_type = std::function<double(int, float, double)>;

    // foo_struct instance
    foo_struct fs;

    // here we will store all binded functions
    std::vector<function_type> bindings;

    // var #1 - you can use simple function
    function_type var1 = foo_fn;
    bindings.push_back(var1);
}

```

```

// var #2 - you can use member function
function_type var2 = std::bind(&foo_struct::foo_fn, fs, _1, _2, _3);
bindings.push_back(var2);

// var #3 - you can use member function with different signature
// foo_fn_4 has different count of arguments and types
function_type var3 = std::bind(&foo_struct::foo_fn_4, fs, _1, _3, _2, 0);
bindings.push_back(var3);

// var #4 - you can use object with overloaded operator()
function_type var4 = fs;
bindings.push_back(var4);

// var #5 - you can use lambda function
function_type var5 = [](int x, float y, double z)
{
    double res = x + y + z;
    std::cout << "lambda called with arguments: "
        << x << ", " << y << ", " << z
        << " result is : " << res
        << std::endl;
    return res;
};
bindings.push_back(var5);

std::cout << "Test stored functions with arguments: x = 1, y = 2, z = 3"
    << std::endl;

for (auto f : bindings)
    f(1, 2, 3);
}

```

```

Test stored functions with arguments: x = 1, y = 2, z = 3
foo_fn called with arguments: 1, 2, 3 result is : 6
foo_struct::foo_fn called with arguments: 1, 2, 3 result is : 6
foo_struct::foo_fn_4 called with arguments: 1, 3, 2, 0 result is : 6
foo_struct::operator() called with arguments: 1, 2, 3 result is : 6
lambda called with arguments: 1, 2, 3 result is : 6

```

std :: tuple

o

std :: tuple

```

#include <iostream>
#include <functional>
#include <tuple>
#include <iostream>

// simple function to be called
double foo_fn(int x, float y, double z)
{
    double res = x + y + z;
    std::cout << "foo_fn called. x = " << x << " y = " << y << " z = " << z
        << " res=" << res;
    return res;
}

```

```

}

// helpers for tuple unrolling
template<int ...> struct seq {};
template<int N, int ...S> struct gens : gens<N-1, N-1, S...> {};
template<int ...S> struct gens<0, S...>{ typedef seq<S...> type; };

// invocation helper
template<typename FN, typename P, int ...S>
double call_fn_internal(const FN& fn, const P& params, const seq<S...>)
{
    return fn(std::get<S>(params) ...);
}
// call function with arguments stored in std::tuple
template<typename Ret, typename ...Args>
Ret call_fn(const std::function<Ret(Args...)>& fn,
            const std::tuple<Args...>& params)
{
    return call_fn_internal(fn, params, typename gens<sizeof...(Args)>::type());
}

int main(void)
{
    // arguments
    std::tuple<int, float, double> t = std::make_tuple(1, 5, 10);
    // function to call
    std::function<double(int, float, double)> fn = foo_fn;

    // invoke a function with stored arguments
    call_fn(fn, t);
}

```

```
foo_fn called. x = 1 y = 5 z = 10 res=16
```

std :: function <https://riptutorial.com/zh-TW/cplusplus/topic/2294/std---function->

28: std :: setstd :: multiset

set multisetmultiset

C++ C++ 98;

Examples

o

- o o
- o o ; o
- o o

```
#include <iostream>
#include <set>

int main ()
{
    std::set<int> sut;
    std::set<int>::iterator it;
    std::pair<std::set<int>::iterator,bool> ret;

    // Basic insert
    sut.insert(7);
    sut.insert(5);
    sut.insert(12);

    ret = sut.insert(23);
    if (ret.second==true)
        std::cout << "# 23 has been inserted!" << std::endl;

    ret = sut.insert(23); // since it's a set and 23 is already present in it, this insert
    should fail
    if (ret.second==false)
        std::cout << "# 23 already present in set!" << std::endl;

    // Insert with hint for optimization
    it = sut.end();
    // This case is optimized for C++11 and above
    // For earlier version, point to the element preceding your insertion
    sut.insert(it, 30);

    // inserting a range of values
    std::set<int> sut2;
    sut2.insert(20);
    sut2.insert(30);
    sut2.insert(45);
    std::set<int>::iterator itStart = sut2.begin();
    std::set<int>::iterator itEnd = sut2.end();

    sut.insert (itStart, itEnd); // second iterator is excluded from insertion

    std::cout << std::endl << "Set under test contains:" << std::endl;
```

```

for (it = sut.begin(); it != sut.end(); ++it)
{
    std::cout << *it << std::endl;
}

return 0;
}

```

```

# 23 has been inserted!
# 23 already present in set!

```

Set under test contains:

```

5
7
12
20
23
30
45

```

◦ initializer_list

```

auto il = { 7, 5, 12 };
std::multiset<int> msut;
msut.insert(il);

```

setmultiset◦

◦ ◦ intcompare

```

#include <iostream>
#include <set>
#include <stdlib.h>

struct custom_compare final
{
    bool operator() (const std::string& left, const std::string& right) const
    {
        int nLeft = atoi(left.c_str());
        int nRight = atoi(right.c_str());
        return nLeft < nRight;
    }
};

int main ()
{
    std::set<std::string> sut({"1", "2", "5", "23", "6", "290"});
}

```

```

std::cout << "### Default sort on std::set<std::string> :" << std::endl;
for (auto &&data: sut)
    std::cout << data << std::endl;

std::set<std::string, custom_compare> sut_custom({"1", "2", "5", "23", "6", "290"},
                                                custom_compare{}); //< Compare object
optional as its default constructible.

std::cout << std::endl << "### Custom sort on set :" << std::endl;
for (auto &&data : sut_custom)
    std::cout << data << std::endl;

auto compare_via_lambda = [](auto &&lhs, auto &&rhs){ return lhs > rhs; };
using set_via_lambda = std::set<std::string, decltype(compare_via_lambda)>;
set_via_lambda sut_reverse_via_lambda({"1", "2", "5", "23", "6", "290"},
                                       compare_via_lambda);

std::cout << std::endl << "### Lambda sort on set :" << std::endl;
for (auto &&data : sut_reverse_via_lambda)
    std::cout << data << std::endl;

return 0;
}

```

```

### Default sort on std::set<std::string> :
1
2
23
290
5
6
### Custom sort on set :
1
2
5
6
23
290

### Lambda sort on set :
6
5
290
23
2
1

```

3std::set◦

- std::less<T>◦ operator< ◦ ◦

API23◦ operator< ◦

-

- ◦ std::pairfirst◦

- ◦ `constconst.....`
- ◦ `compare`

Lambda

[Lambda](#) ◦ ◦

`lambda` `lambda` `cpp` `decltype(lambda)` ◦ ◦

`std::set` ◦

`std::set` `compare` ◦

-
- 2 ◦

setmultiset

`std::set` `std::multiset`

`find()` ◦ `end()` ◦

```
std::set<int> sut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(3); // contains 3, 10, 15, 22

auto itS = sut.find(10); // the value is found, so *itS == 10
itS = sut.find(555); // the value is not found, so itS == sut.end()

std::multiset<int> msut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(15);
sut.insert(3); // contains 3, 10, 15, 15, 22

auto itMS = msut.find(10);
```

`count()` `set / multiset` `set` 01 ◦

```
int result = sut.count(10); // result == 1
result = sut.count(555); // result == 0

result = msut.count(10); // result == 1
result = msut.count(15); // result == 2
```

`std::multiset` ◦ `equal_range()` ◦ `std::pair` ◦ `multiset` ◦

```
auto eqr = msut.equal_range(15);
```

```
auto st = eqr.first; // point to first element '15'
auto en = eqr.second; // point to element '22'

eqr = msut.equal_range(9); // both eqr.first and eqr.second point to element '10'
```

set / multiset_{clear}

```
std::set<int> sut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(3);
sut.clear(); //size of sut is 0
```

erase^o

```
std::set<int> sut;
std::set<int>::iterator it;

sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(3);
sut.insert(30);
sut.insert(33);
sut.insert(45);

// Basic deletion
sut.erase(3);

// Using iterator
it = sut.find(22);
sut.erase(it);

// Deleting a range of values
it = sut.find(33);
sut.erase(it, sut.end());

std::cout << std::endl << "Set under test contains:" << std::endl;
for (it = sut.begin(); it != sut.end(); ++it)
{
    std::cout << *it << std::endl;
}
```

Set under test contains:

10

15

30

multiset^o multiset^o

std :: set **std :: multiset** <https://riptutorial.com/zh-TW/cplusplus/topic/9005/std----setstd----multiset>

29: Typedef

using typedef C ++ 11.

- `typedef type-specifier-seq init-declarator-list ;`
- `attribute-specifier-seq typedef decl-specifier-seq init-declarator-list ; //C ++ 11`
- `attribute-specifier-seq opt = type-id ; //C ++ 11`

Examples

typedef

typedef typedef ◦ typedef ◦

```
int T;          // T has type int
typedef int T; // T is an alias for int

int A[100];     // A has type "array of 100 ints"
typedef int A[100]; // A is an alias for the type "array of 100 ints"
```

◦

```
typedef int A[100];
// S is a struct containing an array of 100 ints
struct S {
    A data;
};
```

typedef ◦ ◦

```
struct S {
    int f(int);
};
typedef int I;
// ok: defines int S::f(int)
I S::f(I x) { return x; }
```

typedef

typedef ◦

```
void (*f)(int); // f has type "pointer to function of int returning void"
typedef void (*f)(int); // f is an alias for "pointer to function of int returning void"
```

◦

```
void (Foo::*pmf)(int); // pmf has type "pointer to member function of Foo taking int
// and returning void"
```

```
typedef void (Foo::*pmf)(int); // pmf is an alias for "pointer to member function of Foo
                               // taking int and returning void"
```

```
void (Foo::*Foo::f(const char*))(int);
int (&g())[100];
```

typedef

```
typedef void (Foo::pmf)(int); // pmf is a pointer to member function type
pmf Foo::f(const char*);     // f is a member function of Foo

typedef int (&ra)[100];      // ra means "reference to array of 100 ints"
ra g();                      // g returns reference to array of 100 ints
```

typedef

typedef° typedef°

```
int *x, (*p)(); // x has type int*, and p has type int(*)()
typedef int *x, (*p)(); // x is an alias for int*, while p is an alias for int(*)()
```

“”

C++ 11

using° °

```
using I = int;
using A = int[100]; // array of 100 ints
using FP = void(*) (int); // pointer to function of int returning void
using MP = void (Foo::*)(int); // pointer to member function of Foo of int returning void
```

usingusing typedef° °

typedef using° using“template typedef”°

Typedef <https://riptutorial.com/zh-TW/cplusplus/topic/9328/typedef>

30: ODR

Examples

◦ ◦

foo.h

```
#ifndef FOO_H
#define FOO_H
#include <iostream>
void foo() { std::cout << "foo"; }
void bar();
#endif
```

foo.cpp

```
#include "foo.h"
void bar() { std::cout << "bar"; }
```

main.cpp

```
#include "foo.h"
int main() {
    foo();
    bar();
}
```

foofoo.hfoo.cpp main.cpp ◦ foo◦ foo.hfoo.cppmain.cppfoo.h ◦ foo◦

.cpp◦

inline◦ ◦ ◦

◦

foo.h

```
#ifndef FOO_H
#define FOO_H
#include <iostream>
inline void foo() { std::cout << "foo"; }
void bar();
#endif
```

foo.cpp

```
#include "foo.h"
void bar() {
```

```
    // more complicated definition
}
```

main.cpp

```
#include "foo.h"
int main() {
    foo();
    bar();
}
```

foo.o main.o foo.cpp main.cpp foofoo.o

o

```
// in foo.h
class Foo {
    void bar() { std::cout << "bar"; }
    void baz();
};

// in foo.cpp
void Foo::baz() {
    // definition
}
```

Foo::baz.o

ODR

ODR^o func

- header.h

```
void overloaded(int);
inline void func() { overloaded('*'); }
```

- Foo.cpp

```
#include "header.h"

void foo()
{
    func(); // `overloaded` refers to `void overloaded(int)`
}
```

- bar.cpp

```
void overloaded(char); // can come from other include
#include "header.h"

void bar()
```

```
{  
    func(); // `overloaded` refers to `void overloaded(char)`  
}
```

ODR_{overloaded}

ODR <https://riptutorial.com/zh-TW/cplusplus/topic/4907/-odr->

31:

Examples

C++ 11

C++ 11. C++ 03 [Rule of Three](#) C++ 11 Five of Rule.

◦ ◦ ◦

```
class Person
{
    char* name;
    int age;

public:
    // Destructor
    ~Person() { delete [] name; }

    // Implement Copy Semantics
    Person(Person const& other)
        : name(new char[std::strlen(other.name) + 1])
        , age(other.age)
    {
        std::strcpy(name, other.name);
    }

    Person &operator=(Person const& other)
    {
        // Use copy and swap idiom to implement assignment.
        Person copy(other);
        swap(*this, copy);
        return *this;
    }

    // Implement Move Semantics
    // Note: It is usually best to mark move operators as noexcept
    //       This allows certain optimizations in the standard library
    //       when the class is used in a container.

    Person(Person&& that) noexcept
        : name(nullptr) // Set the state so we know it is undefined
        , age(0)
    {
        swap(*this, that);
    }

    Person& operator=(Person&& that) noexcept
    {
        swap(*this, that);
        return *this;
    }

    friend void swap(Person& lhs, Person& rhs) noexcept
    {
        std::swap(lhs.name, rhs.name);
    }
};
```



```

        std::swap(lhs.age, rhs.age);
    }
};

```

◦

```

Person& operator=(Person copy)
{
    swap(*this, copy);
    return *this;
}

```

“”◦◦

C++ 11

RAII◦ default◦

Rule of Three Person cstrings

```

class cstring {
private:
    char* p;

public:
    ~cstring() { delete [] p; }
    cstring(cstring const& );
    cstring(cstring&& );
    cstring& operator=(cstring const& );
    cstring& operator=(cstring&& );

    /* other members as appropriate */
};

```

Person

```

class Person {
    cstring name;
    int arg;

public:
    ~Person() = default;
    Person(Person const& ) = default;
    Person(Person&& ) = default;
    Person& operator=(Person const& ) = default;
    Person& operator=(Person&& ) = default;

    /* other members as appropriate */
};

```

Person; Person◦◦

```

struct Person {
    cstring name;
    int arg;
};

```

```
};
```

cstringdelete d/Person

R. Martinho Fernandes

C++ 03

Rule of Three

RAII

```
class Person
{
    char* name;
    int age;

public:
    Person(char const* new_name, int new_age)
        : name(new char[std::strlen(new_name) + 1])
        , age(new_age)
    {
        std::strcpy(name, new_name);
    }

    ~Person() {
        delete [] name;
    }
};
```

name

```
int main()
{
    Person p1("foo", 11);
    Person p2 = p1;
}
```

p1 p2 p1 C++ p1.name p2.name

main p2; p1 delete

Person

```
Person(Person const& other)
    : name(new char[std::strlen(other.name) + 1])
    , age(other.age)
{
    std::strcpy(name, other.name);
}

Person &operator=(Person const& other)
{
    // Use copy and swap idiom to implement assignment
```

```
Person copy(other);
swap(copy);           // assume swap() exchanges contents of *this and copy
return *this;
}
```

◦ ◦ *thiscopycopycopy ◦ copy*this◦

◦

```
SomeType t = ...;
t = t;
```

◦ Person◦

◦

```
SomeType &operator=(const SomeType &other)
{
    if(this != &other)
    {
        //Do assignment logic.
    }
    return *this;
}
```

◦ ◦ ◦

copy and swap idiom ◦ ◦ ◦

C++ 11

◦ `std::swap` / ◦

◦

<https://riptutorial.com/zh-TW/cplusplus/topic/1206/-->

32: C ++ C ++ 11 C ++ 14 C ++ 17 C ++

Examples

C ++ c

```
for(size_t i = 0; i < c.size(); ++i) c[i] = 0;
```

```
for(size_t i = 0; i <= c.size(); ++j) c[i] = 0;
      ^~~~~~^
```

```
for(iterator it = c.begin(); it != c.end(); ++it) (*it) = 0;
```

C ++ 11 for auto

```
for(auto& x : c) x = 0;
```

c x ° °

C ++ 11

```
for(auto begin = c.begin(), end = c.end(); begin != end; ++begin)
{
    // ...
}
```

auto begin = c.begin(), end = c.end(); ~~force~~beginendend° for/° C ++ 17

```
auto begin = c.begin();
auto end = c.end();
for(; begin != end; ++begin)
{
    // ...
}
```

beginend° /°

C ++ C ++ 11 C ++ 14 C ++ 17 C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/7134/c-plusplus-plusplus-11c-plusplus-14c-plusplus-17c-plusplus>

33:

std :: shared_mutexstd :: shared_timed_mutex

◦

RWLock◦

std :: shared_mutexshared_timed_mutex◦

std :: shared_timed_mutex'std :: shared_mutex + time method'◦

◦

std :: shared_mutexMSVC14.1

```
class shared_mutex
{
public:
typedef _Smtx_t * native_handle_type;

shared_mutex() _NOEXCEPT
    : _Myhandle(0)
    { // default construct
    }

~shared_mutex() _NOEXCEPT
    { // destroy the object
    }

void lock() _NOEXCEPT
    { // lock exclusive
    _Smtx_lock_exclusive(&_Myhandle);
    }

bool try_lock() _NOEXCEPT
    { // try to lock exclusive
    return (_Smtx_try_lock_exclusive(&_Myhandle) != 0);
    }

void unlock() _NOEXCEPT
    { // unlock exclusive
    _Smtx_unlock_exclusive(&_Myhandle);
    }

void lock_shared() _NOEXCEPT
    { // lock non-exclusive
    _Smtx_lock_shared(&_Myhandle);
    }

bool try_lock_shared() _NOEXCEPT
    { // try to lock non-exclusive
```

```

    return (_Smtx_try_lock_shared(&_Myhandle) != 0);
}

void unlock_shared() _NOEXCEPT
{
    // unlock non-exclusive
    _Smtx_unlock_shared(&_Myhandle);
}

native_handle_type native_handle() _NOEXCEPT
{
    // get native handle
    return (&_Myhandle);
}

shared_mutex(const shared_mutex&) = delete;
shared_mutex& operator=(const shared_mutex&) = delete;
private:
    _Smtx_t _Myhandle;
};

void __cdecl _Smtx_lock_exclusive(_Smtx_t * smtx)
{
    /* lock shared mutex exclusively */
    AcquireSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smtx));
}

void __cdecl _Smtx_lock_shared(_Smtx_t * smtx)
{
    /* lock shared mutex non-exclusively */
    AcquireSRWLockShared(reinterpret_cast<PSRWLOCK>(smtx));
}

int __cdecl _Smtx_try_lock_exclusive(_Smtx_t * smtx)
{
    /* try to lock shared mutex exclusively */
    return (TryAcquireSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smtx)));
}

int __cdecl _Smtx_try_lock_shared(_Smtx_t * smtx)
{
    /* try to lock shared mutex non-exclusively */
    return (TryAcquireSRWLockShared(reinterpret_cast<PSRWLOCK>(smtx)));
}

void __cdecl _Smtx_unlock_exclusive(_Smtx_t * smtx)
{
    /* unlock exclusive shared mutex */
    ReleaseSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smtx));
}

void __cdecl _Smtx_unlock_shared(_Smtx_t * smtx)
{
    /* unlock non-exclusive shared mutex */
    ReleaseSRWLockShared(reinterpret_cast<PSRWLOCK>(smtx));
}

```

std :: shared_mutex Windows Slim Reader / Write Locks [https://msdn.microsoft.com/ko-kr/library/windows/desktop/aa904937\(v=vs.85\).aspx](https://msdn.microsoft.com/ko-kr/library/windows/desktop/aa904937(v=vs.85).aspx)

std :: shared_timed_mutex。

std :: shared_timed_mutex MSVC14.1。

```

class shared_timed_mutex
{

```

```

typedef unsigned int _Read_cnt_t;
static constexpr _Read_cnt_t _Max_readers = _Read_cnt_t(-1);
public:
shared_timed_mutex() _NOEXCEPT
    : _Mymtx(), _Read_queue(), _Write_queue(),
      _Readers(0), _Writing(false)
{    // default construct
}

~shared_timed_mutex() _NOEXCEPT
{    // destroy the object
}

void lock()
{    // lock exclusive
unique_lock<mutex> _Lock(_Mymtx);
while (_Writing)
    _Write_queue.wait(_Lock);
_Writing = true;
while (0 < _Readers)
    _Read_queue.wait(_Lock);    // wait for writing, no readers
}

bool try_lock()
{    // try to lock exclusive
lock_guard<mutex> _Lock(_Mymtx);
if (_Writing || 0 < _Readers)
    return (false);
else
{    // set writing, no readers
_Writing = true;
return (true);
}
}

template<class _Rep,
class _Period>
bool try_lock_for(
    const chrono::duration<_Rep, _Period>& _Rel_time)
{    // try to lock for duration
return (try_lock_until(chrono::steady_clock::now() + _Rel_time));
}

template<class _Clock,
class _Duration>
bool try_lock_until(
    const chrono::time_point<_Clock, _Duration>& _Abs_time)
{    // try to lock until time point
auto _Not_writing = [this] { return (!_Writing); };
auto _Zero_readers = [this] { return (_Readers == 0); };
unique_lock<mutex> _Lock(_Mymtx);

if (!_Write_queue.wait_until(_Lock, _Abs_time, _Not_writing))
    return (false);

_Writing = true;

if (!_Read_queue.wait_until(_Lock, _Abs_time, _Zero_readers))
{    // timeout, leave writing state
_Writing = false;
_Lock.unlock();    // unlock before notifying, for efficiency
}
}

```

```

        _Write_queue.notify_all();
        return (false);
    }

    return (true);
}

void unlock()
{    // unlock exclusive
    {    // unlock before notifying, for efficiency
        lock_guard<mutex> _Lock(_Mymtx);

        _Writing = false;
    }

    _Write_queue.notify_all();
}

void lock_shared()
{    // lock non-exclusive
    unique_lock<mutex> _Lock(_Mymtx);
    while (_Writing || _Readers == _Max_readers)
        _Write_queue.wait(_Lock);
    ++_Readers;
}

bool try_lock_shared()
{    // try to lock non-exclusive
    lock_guard<mutex> _Lock(_Mymtx);
    if (_Writing || _Readers == _Max_readers)
        return (false);
    else
    {    // count another reader
        ++_Readers;
        return (true);
    }
}

template<class _Rep,
         class _Period>
bool try_lock_shared_for(
    const chrono::duration<_Rep, _Period>& _Rel_time)
{    // try to lock non-exclusive for relative time
    return (try_lock_shared_until(_Rel_time
        + chrono::steady_clock::now()));
}

template<class _Time>
bool _Try_lock_shared_until(_Time _Abs_time)
{    // try to lock non-exclusive until absolute time
    auto _Can_acquire = [this] {
        return (!_Writing && _Readers < _Max_readers); };

    unique_lock<mutex> _Lock(_Mymtx);

    if (!_Write_queue.wait_until(_Lock, _Abs_time, _Can_acquire))
        return (false);

    ++_Readers;
    return (true);
}

```



```

template<class _Clock,
        class _Duration>
bool try_lock_shared_until(
    const chrono::time_point<_Clock, _Duration>& _Abs_time)
{    // try to lock non-exclusive until absolute time
return (_Try_lock_shared_until(_Abs_time));
}

bool try_lock_shared_until(const xtime *_Abs_time)
{    // try to lock non-exclusive until absolute time
return (_Try_lock_shared_until(_Abs_time));
}

void unlock_shared()
{    // unlock non-exclusive
_Read_cnt_t _Local_readers;
bool _Local_writing;

    {    // unlock before notifying, for efficiency
lock_guard<mutex> _Lock(_Mymtx);
--_Readers;
_Local_readers = _Readers;
_Local_writing = _Writing;
}

    if (_Local_writing && _Local_readers == 0)
        _Read_queue.notify_one();
    else if (!_Local_writing && _Local_readers == _Max_readers - 1)
        _Write_queue.notify_all();
}

shared_timed_mutex(const shared_timed_mutex&) = delete;
shared_timed_mutex& operator=(const shared_timed_mutex&) = delete;
private:
mutex _Mymtx;
condition_variable _Read_queue, _Write_queue;
_Read_cnt_t _Readers;
bool _Writing;
};

class stl_condition_variable_win7 final : public stl_condition_variable_interface
{
public:
    stl_condition_variable_win7()
    {
        __crtInitializeConditionVariable(&m_condition_variable);
    }

    ~stl_condition_variable_win7() = delete;
    stl_condition_variable_win7(const stl_condition_variable_win7&) = delete;
    stl_condition_variable_win7& operator=(const stl_condition_variable_win7&) = delete;

    virtual void destroy() override {}

    virtual void wait(stl_critical_section_interface *lock) override
    {
        if (!stl_condition_variable_win7::wait_for(lock, INFINITE))
            std::terminate();
    }
}

```

```

    virtual bool wait_for(stl_critical_section_interface *lock, unsigned int timeout) override
    {
        return __crtSleepConditionVariableSRW(&m_condition_variable,
static_cast<stl_critical_section_win7 *>(lock)->native_handle(), timeout, 0) != 0;
    }

    virtual void notify_one() override
    {
        __crtWakeConditionVariable(&m_condition_variable);
    }

    virtual void notify_all() override
    {
        __crtWakeAllConditionVariable(&m_condition_variable);
    }

private:
    CONDITION_VARIABLE m_condition_variable;
};

```

std :: shared_timed_mutexstd :: condition_value。

- 。
- 。

```

STLSharedMutex READ :          486647
STLSharedMutex WRITE :         205986
TOTAL READ&WRITE :             692633

STLSharedTimedMutex READ :      140291
STLSharedTimedMutex WRITE :     178849
TOTAL READ&WRITE :             319140

```

1000/。

std :: shared_mutex/std :: shared_timed_mutex2 。

- /。
- 。
- 。

```

void useSTLSharedMutex()
{
    std::shared_mutex shared_mtx_lock;

    std::vector<std::thread> readThreads;
    std::vector<std::thread> writeThreads;

    std::list<int> data = { 0 };
    volatile bool exit = false;

    std::atomic<int> readProcessedCnt(0);
    std::atomic<int> writeProcessedCnt(0);
}

```

```

for (unsigned int i = 0; i < std::thread::hardware_concurrency(); i++)
{
    readThreads.push_back(std::thread([&data, &exit, &shared_mtx_lock,
&readProcessedCnt]() {
        std::list<int> mydata;
        int localProcessCnt = 0;

        while (true)
        {
            shared_mtx_lock.lock_shared();

            mydata.push_back(data.back());
            ++localProcessCnt;

            shared_mtx_lock.unlock_shared();

            if (exit)
                break;
        }

        std::atomic_fetch_add(&readProcessedCnt, localProcessCnt);

    }));

    writeThreads.push_back(std::thread([&data, &exit, &shared_mtx_lock,
&writeProcessedCnt]() {

        int localProcessCnt = 0;

        while (true)
        {
            shared_mtx_lock.lock();

            data.push_back(rand() % 100);
            ++localProcessCnt;

            shared_mtx_lock.unlock();

            if (exit)
                break;
        }

        std::atomic_fetch_add(&writeProcessedCnt, localProcessCnt);

    }));
}

std::this_thread::sleep_for(std::chrono::milliseconds(MAIN_WAIT_MILLISECONDS));
exit = true;

for (auto &r : readThreads)
    r.join();

for (auto &w : writeThreads)
    w.join();

std::cout << "STLSharedMutex READ :           " << readProcessedCnt << std::endl;
std::cout << "STLSharedMutex WRITE :           " << writeProcessedCnt << std::endl;
std::cout << "TOTAL READ&WRITE :           " << readProcessedCnt + writeProcessedCnt <<

```

```

std::endl << std::endl;
}

void useSTLSharedTimedMutex()
{
    std::shared_timed_mutex shared_mtx_lock;

    std::vector<std::thread> readThreads;
    std::vector<std::thread> writeThreads;

    std::list<int> data = { 0 };
    volatile bool exit = false;

    std::atomic<int> readProcessedCnt(0);
    std::atomic<int> writeProcessedCnt(0);

    for (unsigned int i = 0; i < std::thread::hardware_concurrency(); i++)
    {
        readThreads.push_back(std::thread([&data, &exit, &shared_mtx_lock,
&readProcessedCnt]() {
            std::list<int> mydata;
            int localProcessCnt = 0;

            while (true)
            {
                shared_mtx_lock.lock_shared();

                mydata.push_back(data.back());
                ++localProcessCnt;

                shared_mtx_lock.unlock_shared();

                if (exit)
                    break;
            }

            std::atomic_fetch_add(&readProcessedCnt, localProcessCnt);

        }));

        writeThreads.push_back(std::thread([&data, &exit, &shared_mtx_lock,
&writeProcessedCnt]() {

            int localProcessCnt = 0;

            while (true)
            {
                shared_mtx_lock.lock();

                data.push_back(rand() % 100);
                ++localProcessCnt;

                shared_mtx_lock.unlock();

                if (exit)
                    break;
            }

            std::atomic_fetch_add(&writeProcessedCnt, localProcessCnt);

```

```

    });
}

std::this_thread::sleep_for(std::chrono::milliseconds(MAIN_WAIT_MILLISECONDS));
exit = true;

for (auto &r : readThreads)
    r.join();

for (auto &w : writeThreads)
    w.join();

std::cout << "STLSharedTimedMutex READ :      " << readProcessedCnt << std::endl;
std::cout << "STLSharedTimedMutex WRITE :     " << writeProcessedCnt << std::endl;
std::cout << "TOTAL READ&WRITE :                   " << readProcessedCnt + writeProcessedCnt <<
std::endl << std::endl;
}

```

Examples

std::unique_lockstd::shared_lockstd::lock_guard

RAIItry

std::unique_lock

std::shared_lock std::shared_mutexstd::shared_locks ◦ C++ 14

std::lock_guardstd::unique_lockstd::shared_lock

```

#include <unordered_map>
#include <mutex>
#include <shared_mutex>
#include <thread>
#include <string>
#include <iostream>

class PhoneBook {
public:
    std::string getPhoneNo( const std::string & name )
    {
        std::shared_lock<std::shared_timed_mutex> l(_protect);
        auto it = _phonebook.find( name );
        if ( it != _phonebook.end() )
            return (*it).second;
        return "";
    }
    void addPhoneNo ( const std::string & name, const std::string & phone )
    {
        std::unique_lock<std::shared_timed_mutex> l(_protect);
        _phonebook[name] = phone;
    }

    std::shared_timed_mutex _protect;
    std::unordered_map<std::string, std::string> _phonebook;
};

```

std :: try_to_lockstd :: adopt_lockstd :: defer_lock

std :: unique_lock std::try_to_lock std::defer_lockstd::adopt_lock

1. std::try_to_lock

```
{
    std::atomic_int temp {0};
    std::mutex _mutex;

    std::thread t( [&]() {

        while( temp!= -1){
            std::this_thread::sleep_for(std::chrono::seconds(5));
            std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);

            if(lock.owns_lock()){
                //do something
                temp=0;
            }
        }
    });

    while ( true )
    {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);
        if(lock.owns_lock()){
            if (temp < INT_MAX){
                ++temp;
            }
            std::cout << temp << std::endl;
        }
    }
}
```

2. std::defer_lock

```
{
    std::unique_lock<std::mutex> lock1(_mutex1, std::defer_lock);
    std::unique_lock<std::mutex> lock2(_mutex2, std::defer_lock);
    lock1.lock()
    lock2.lock(); // deadlock here
    std::cout << "Locked! << std::endl;
    //...
}
```

```
{
    std::unique_lock<std::mutex> lock1(_mutex1, std::defer_lock);
    std::unique_lock<std::mutex> lock2(_mutex2, std::defer_lock);
    std::lock(lock1,lock2); // no deadlock possible
    std::cout << "Locked! << std::endl;
    //...
}
```

3. std::adopt_lock

```

{
    std::unique_lock<std::mutex> lock1(_mutex1, std::adopt_lock);
    std::unique_lock<std::mutex> lock2(_mutex2, std::adopt_lock);
    std::cout << "Locked! << std::endl;
    //...
}

```

`std::adopt_lock` ◦

`std::`

`std::mutex` ◦

```

std::atomic_int temp{0};
std::mutex _mutex;

std::thread t( [&]() {
    while( temp!= -1){
        std::this_thread::sleep_for(std::chrono::seconds(5));
        std::unique_lock<std::mutex> lock( _mutex);

        temp=0;
    }
});

while ( true )
{
    std::this_thread::sleep_for(std::chrono::milliseconds(1));
    std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);
    if ( temp < INT_MAX )
        temp++;
    cout << temp << endl;
}

```

std::scoped_lockC ++ 17

`std::scoped_lockRAIIstd::lock` ◦ `std::scoped_lock` ◦

```

{
    std::scoped_lock lock{ _mutex1, _mutex2};
    //do something
}

```

C ++ 1x

- [std::mutex](#) - ◦
- `std::timed_mutex` - `try_to_lock`
- [std::recursive_mutex](#) - ◦
- `std::shared_mutexstd::shared_timed_mutex` - ◦

std ::

std::lock° std::lock°

```
std::lock(_mutex1, _mutex2);
```

<https://riptutorial.com/zh-TW/cplusplus/topic/9895/>

34:

CC ++ ◦ ◦ ◦ sizeof() ◦

◦

```
struct foo {
    unsigned x;
    unsigned y;
}
static struct foo my_var;
```

```
my_var.y = 5;
```

sizeof (unsigned) == 4 **xy4** ◦

```
loada register1,#myvar      ; get the address of the structure
storei register1[4],#0x05   ; put the value '5' at offset 4, e.g., set y=5
```

xy ◦

```
struct foo {
    unsigned x : 4; /* Range 0-0x0f, or 0 through 15 */
    unsigned y : 4;
}
```

xy4 ◦ **18** ◦ **y**

```
loada  register1,#myvar      ; get the address of the structure
loadb  register2,register1[0] ; get the byte from memory
andb   register2,#0x0f      ; zero out y in the byte, leaving x alone
orb    register2,#0x50      ; put the 5 into the 'y' portion of the byte
stb    register1[0],register2 ; put the modified byte back into memory
```

- ◦ ◦

◦ ◦ - - ◦

Examples

```
struct FileAttributes
{
    unsigned int ReadOnly: 1;
    unsigned int Hidden: 1;
};
```

1 ◦ : 1: 1: 1 ◦ 8int64int ◦ unsigned ◦

“1”。

```
struct Date
{
    unsigned int Year : 13; // 2^13 = 8192, enough for "year" representation for long time
    unsigned int Month: 4; // 2^4 = 16, enough to represent 1-12 month values.
    unsigned int Day: 5; // 32
};
```

22sizeof4。

。。

```
Date d;

d.Year = 2016;
d.Month = 7;
d.Day = 22;

std::cout << "Year:" << d.Year << std::endl <<
    "Month:" << d.Month << std::endl <<
    "Day:" << d.Day << std::endl;
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2710/>

35:

`std::bitset<bitset>` ◦

```
#include <bitset>
```

`std::bitset` **csetbitset** ◦

•

Examples

C

OR | ◦

```
// Bit x will be set  
number |= 1LL << x;
```

std :: bitset

`set(x)` `set(x,true)` - x1 ◦

```
std::bitset<5> num(std::string("01100"));  
num.set(0); // num is now 01101  
num.set(2); // num is still 01101  
num.set(4,true); // num is now 11110
```

C

AND & ◦

```
// Bit x will be cleared  
number &= ~(1LL << x);
```

std :: bitset

`reset(x)` `set(x,false)` - x◦

```
std::bitset<5> num(std::string("01100"));
```

```
num.reset(2); // num is now 01000
num.reset(0); // num is still 01000
num.set(3,false); // num is now 00000
```

C

XOR \wedge ◦

```
// Bit x will be the opposite value of what it is currently
number  $\wedge$ = 1LL << x;
```

std :: bitset

```
std::bitset<4> num(std::string("0100"));
num.flip(2); // num is now 0000
num.flip(0); // num is now 0001
num.flip(); // num is now 1110 (flips all bits)
```

C

xAND &

```
(number >> x) & 1LL; // 1 if the 'x'th bit of 'number' is set, 0 otherwise
```

◦ number number >> x ◦

```
(number & (1LL << x)); // (1 << x) if the 'x'th bit of 'number' is set, 0 otherwise
```

◦

std :: bitset

```
std::bitset<4> num(std::string("0010"));
bool bit_val = num.test(1); // bit_val value is set to true;
```

nx

C

```
// Bit n will be set if x is 1 and cleared if x is 0.
number  $\wedge$ = (-x  $\wedge$  number) & (1LL << n);
```

std :: bitset

set(n, val) = nval ◦

```
std::bitset<5> num(std::string("00100"));
num.set(0,true); // num is now 00101
num.set(2,false); // num is now 00001
```

C

```
x = -1; // -1 == 1111 1111 ... 1111b
```

◦

std :: bitset

```
std::bitset<10> x;
x.set(); // Sets all bits to '1'
```

C

```
template <typename T>
T rightmostSetBitRemoved(T n)
{
    // static_assert(std::is_integral<T>::value && !std::is_signed<T>::value, "type should be
    unsigned"); // For c++11 and later
    return n & (n - 1);
}
```

- $n0 \ \& \ 0xFF..FF$
- $n0bxxxxxx10..00n - 10bxxxxxx011..11 \ n \ \& \ (n - 1)0bxxxxxx000..00$ ◦

◦

```
unsigned value = 1234;
unsigned bits = 0; // accumulates the total number of bits set in `n`

for (bits = 0; value; value >>= 1)
    bits += value & 1;
```

```
unsigned bits = 0; // accumulates the total number of bits set in `n`

for (; value; ++bits)
    value &= value - 1;
```

value°

Peter Wegner [CACM 3/322](#) - 1960CBrian W. KernighanDennis M. Ritchie°

12

```
unsigned popcount(std::uint64_t x)
{
    const std::uint64_t m1 = 0x5555555555555555; // binary: 0101...
    const std::uint64_t m2 = 0x3333333333333333; // binary: 00110011..
    const std::uint64_t m4 = 0x0f0f0f0f0f0f0f0f; // binary: 0000111100001111

    x -= (x >> 1) & m1; // put count of each 2 bits into those 2 bits
    x = (x & m2) + ((x >> 2) & m2); // put count of each 4 bits into those 4 bits
    x = (x + (x >> 4)) & m4; // put count of each 8 bits into those 8 bits
    return (x * h01) >> 56; // left 8 bits of x + (x<<8) + (x<<16) + (x<<24) + ...
}
```

°

CPUx86popcnt ° g ++

```
int __builtin_popcount (unsigned x);
```

2

$n \& (n - 1)$ 2

```
bool power_of_2 = n && !(n & (n - 1));
```

$n \&& 02$ °

° **a**01(1)0000101(0)00001 ° ° **1**°

```
/*
convert small letter to captial letter.
=====
    a: 01100001
   mask: 11011111 <-- (0xDF)  11(0)11111
      :-----
a&mask: 01000001 <-- A letter
*****/
```

A

```
#include <cstdio>

int main()
{
    char op1 = 'a'; // "a" letter (i.e. small case)
```

```
int mask = 0xDF; // choosing a proper mask

printf("a (AND) mask = A\n");
printf("%c & 0xDF = %c\n", op1, op1 & mask);

return 0;
}
```

```
$ g++ main.cpp -o test1
$ ./test1
a (AND) mask = A
a & 0xDF = A
```

<https://riptutorial.com/zh-TW/cplusplus/topic/3016/>

36:

o

```
int a = ~0;
int b = a << 1;
```

6432x86PIC。

>> XOR。Endian-ness。Endianness。

Examples

- AND

```
int a = 6;      // 0110b (0x06)
int b = 10;     // 1010b (0x0A)
int c = a & b;  // 0010b (0x02)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

a = 6, b = 10, c = 2

AND

```
TRUE AND TRUE = TRUE
TRUE AND FALSE = FALSE
FALSE AND FALSE = FALSE
```

a 0110 b 1010 AND "0010

```
int a = 0 1 1 0
int b = 1 0 1 0 &
-----
int c = 0 0 1 0
```

&=AND

```
int a = 5; // 0101b (0x05)
a &= 10; // a = 0101b & 1010b
```

| - OR

```
int a = 5;      // 0101b (0x05)
int b = 12;     // 1100b (0x0C)
int c = a | b;  // 1101b (0x0D)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```


a = 5, b = 12, c = 13

OR

```
true OR true = true
true OR false = true
false OR false = false
```

a 0101 b 1100 OR "1101

```
int a = 0 1 0 1
int b = 1 1 0 0 |
           -----
int c = 1 1 0 1
```

|=OR

```
int a = 5; // 0101b (0x05)
a |= 12; // a = 0101b | 1101b
```

^

```
int a = 5; // 0101b (0x05)
int b = 9; // 1001b (0x09)
int c = a ^ b; // 1100b (0x0C)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

a = 5, b = 9, c = 12

XOR

```
true OR true = false
true OR false = true
false OR false = false
```

XOR true OR true = false true AND/OR true = true XOR°

a 0101 b 1001 XOR 1100

```
int a = 0 1 0 1
int b = 1 0 0 1 ^
           -----
int c = 1 1 0 0
```

^=XOR

```
int a = 5; // 0101b (0x05)
a ^= 9; // a = 0101b ^ 1001b
```

XOR°

◦ `std::swap()` ◦

XOR

```
int a = 42;
int b = 64;

// XOR swap
a ^= b;
b ^= a;
a ^= b;

std::cout << "a = " << a << ", b = " << b << "\n";
```

◦

```
void doXORSwap(int& a, int& b)
{
    // Need to add a check to make sure you are not swapping the same
    // variable with itself. Otherwise it will zero the value.
    if (&a != &b)
    {
        // XOR swap
        a ^= b;
        b ^= a;
        a ^= b;
    }
}
```

◦ `xora ^ c = aca | c`

2015+

```
int cn=0b0111;
```

- NOT

```
unsigned char a = 234; // 1110 1010b (0xEA)
unsigned char b = ~a; // 0001 0101b (0x15)

std::cout << "a = " << static_cast<int>(a) <<
    ", b = " << static_cast<int>(b) << std::endl;
```

a = 234, b = 21

NOT ◦ 1 0 0 1 ◦ NOT

```
unsigned char a = 234; // 1110 1010b (0xEA)
unsigned char b = ~a; // 0001 0101b (0x15)
unsigned char c = a ^ ~0;
```

NOT

```

unsigned int i = ~0;
unsigned char c = ~0;

std::cout << "max uint = " << i << std::endl <<
    "max uchar = " << static_cast<short>(c) << std::endl;

```

NOT \sim 10 °

NOT ~ NOT ! ;NOTNOT (!1) != (~1)

<< -

```

int a = 1;        // 0001b
int b = a << 1;  // 0010b

std::cout << "a = " << a << ", b = " << b << std::endl;

```

a = 1, b = 2

a 1 05 0000 0101 45 << 4 80 0101 0000 ° 12

```

int a = 7;
while (a < 200) {
    std::cout << "a = " << a << std::endl;
    a <<= 1;
}

a = 7;
while (a < 200) {
    std::cout << "a = " << a << std::endl;
    a *= 2;
}

```

```

int a = 2147483647; // 0111 1111 1111 1111 1111 1111 1111 1111
int b = a << 1;    // 1111 1111 1111 1111 1111 1111 1111 1110

std::cout << "a = " << a << ", b = " << b << std::endl;

```

a = 2147483647, b = -2

shift °

```

int a = 1;
int b = a << -1; // undefined behavior
char c = a << 20; // undefined behavior

```

<<=

```

int a = 5; // 0101b
a <<= 1;   // a = a << 1;

```

>> -

```
int a = 2;      // 0010b
int b = a >> 1; // 0001b

std::cout << "a = " << a << ", b = " << b << std::endl;
```

a = 2, b = 1

a 1;

```
int a = -2;
int b = a >> 1; // the value of b will be depend on the compiler
```

```
int a = 1;
int b = a >> -1; // undefined behavior
```

>>=

```
int a = 2; // 0010b
a >>= 1;   // a = a >> 1;
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2572/>

37: OpenMP

OpenMP ++。 OpenMP [OpenMP](#) 。

。

OpenMP。 OpenMP API `omp_get_thread_num()` `omp.h`。

OpenMP `OpenMP` `pragma` 。

- GCC `-fopenmp`
- Clang `-fopenmp`
- MSVC `/openmp`

Examples

OpenMP

。

OpenMP。 `openMP` API `omp.h`

```
std::cout << "begin ";
// This pragma statement hints the compiler that the
// contents within the { } are to be executed in as
// parallel sections using openMP, the compiler will
// generate this chunk of code for parallel execution
#pragma omp parallel sections
{
    // This pragma statement hints the compiler that
    // this is a section that can be executed in parallel
    // with other section, a single section will be executed
    // by a single thread.
    // Note that it is "section" as opposed to "sections" above
    #pragma omp section
    {
        std::cout << "hello " << std::endl;
        /** Do something **/
    }
    #pragma omp section
    {
        std::cout << "world " << std::endl;
        /** Do something **/
    }
}
// This line will not be executed until all the
// sections defined above terminates
std::cout << "end" << std::endl;
```

2。 。



OpenMP

```
std::cout << "begin ";
// Start of parallel sections
#pragma omp parallel sections
{
    // Execute these sections in parallel
    #pragma omp section
    {
        ... do something ...
        std::cout << "hello ";
    }
    #pragma omp section
    {
        ... do something ...
        std::cout << "world ";
    }
    #pragma omp section
    {
        ... do something ...
        std::cout << "forever ";
    }
}
// end of parallel sections
std::cout << "end";
```

-
-
-
-

◦

OpenMPParallel For Loop

◦

```
// Splits element vector into element.size() / Thread Qty
// and allocate that range for each thread.
#pragma omp parallel for
for (size_t i = 0; i < element.size(); ++i)
    element[i] = ...

// Example Allocation (100 element per thread)
// Thread 1 : 0 ~ 99
// Thread 2 : 100 ~ 199
// Thread 2 : 200 ~ 299
// ...

// Continue process
// Only when all threads completed their allocated
// loop job
...
```

*for ◦

OpenMP/

std::vector OpenMP ◦

int ◦

◦

```
// The Master vector
// We want a vector of results gathered from slave threads
std::vector<int> Master;

// Hint the compiler to parallelize this { } of code
// with all available threads (usually the same as logical processor qty)
#pragma omp parallel
{
    // In this area, you can write any code you want for each
    // slave thread, in this case a vector to hold each of their results
    // We don't have to worry about how many threads were spawn or if we need
    // to repeat this declaration or not.
    std::vector<int> Slave;

    // Tell the compiler to use all threads allocated for this parallel region
    // to perform this loop in parts. Actual load appx = 1000000 / Thread Qty
    // The nowait keyword tells the compiler that the slave threads don't
    // have to wait for all other slaves to finish this for loop job
    #pragma omp for nowait
    for (size_t i = 0; i < 1000000; ++i
    {
        /* Do something */
        ....
        Slave.push_back(...);
    }

    // Slaves that finished their part of the job
    // will perform this thread by thread one at a time
    // critical section ensures that only 0 or 1 thread performs
    // the { } at any time
    #pragma omp critical
    {
        // Merge slave into master
        // use move iterators instead, avoid copy unless
        // you want to use it for something else after this section
        Master.insert(Master.end(),
                     std::make_move_iterator(Slave.begin()),
                     std::make_move_iterator(Slave.end()));
    }
}

// Have fun with Master vector
...
```

OpenMP <https://riptutorial.com/zh-TW/cplusplus/topic/8222/openmp>

38: std :: unordered_map

std :: unordered_map ◦ ◦ ◦ ◦

- O1 ◦ operator[] ◦

Examples

◦ ◦

```
unordered_map<string, int> first; //declaration of the map
first["One"] = 1; // [] operator used to insert the value
first["Two"] = 2;
first["Three"] = 3;
first["Four"] = 4;
first["Five"] = 5;

pair <string,int> bar = make_pair("Nine", 9); //make a pair of same type
first.insert(bar); //can also use insert to feed the values
```

```
unordered_map<data_type, data_type> variable_name; //declaration

variable_name[key_value] = mapped_value; //inserting values

variable_name.find(key_value); //returns iterator to the key value

variable_name.begin(); // iterator to the first element

variable_name.end(); // iterator to the last + 1 element
```

std :: unordered_map <https://riptutorial.com/zh-TW/cplusplus/topic/10540/std----unordered-map>

39:

Examples

system_clock°

C++ 11

```
#include <iostream>
#include <chrono>
#include <thread>

int main() {
    auto start = std::chrono::system_clock::now(); // This and "end"'s type is
    std::chrono::time_point
    { // The code to test
        std::this_thread::sleep_for(std::chrono::seconds(2));
    }
    auto end = std::chrono::system_clock::now();

    std::chrono::duration<double> elapsed = end - start;
    std::cout << "Elapsed time: " << elapsed.count() << "s";
}
```

sleep_forstd::chrono::seconds°

。 ///。

2000°

```
#include <iostream>
#include <string>
#include <chrono>
#include <ctime>

/**
 * Creates a std::tm structure from raw date.
 *
 * \param year (must be 1900 or greater)
 * \param month months since January - [1, 12]
 * \param day day of the month - [1, 31]
 * \param minutes minutes after the hour - [0, 59]
 * \param seconds seconds after the minute - [0, 61] (until C++11) / [0, 60] (since C++11)
 *
 * Based on http://en.cppreference.com/w/cpp/chrono/c/tm
 */
std::tm CreateTmStruct(int year, int month, int day, int hour, int minutes, int seconds) {
    struct tm tm_ret = {0};

    tm_ret.tm_sec = seconds;
    tm_ret.tm_min = minutes;
    tm_ret.tm_hour = hour;
    tm_ret.tm_mday = day;
    tm_ret.tm_mon = month - 1;
```

```

    tm_ret.tm_year = year - 1900;

    return tm_ret;
}

int get_days_in_year(int year) {

    using namespace std;
    using namespace std::chrono;

    // We want results to be in days
    typedef duration<int, ratio_multiply<hours::period, ratio<24> >::type> days;

    // Create start time span
    std::tm tm_start = CreateTmStruct(year, 1, 1, 0, 0, 0);
    auto tms = system_clock::from_time_t(std::mktime(&tm_start));

    // Create end time span
    std::tm tm_end = CreateTmStruct(year + 1, 1, 1, 0, 0, 0);
    auto tme = system_clock::from_time_t(std::mktime(&tm_end));

    // Calculate time duration between those two dates
    auto diff_in_days = std::chrono::duration_cast<days>(tme - tms);

    return diff_in_days.count();
}

int main()
{
    for ( int year = 2000; year <= 2016; ++year )
        std::cout << "There are " << get_days_in_year(year) << " days in " << year << "\n";
}

```

<https://riptutorial.com/zh-TW/cplusplus/topic/3936/-chrono->

40:

using

- `typename opt nested-name-specifier unqualified-id ;`
- `using :: unqualified-id ;`

`using using` ◦ `using-directive` `using namespace` ◦

`using typedef` ◦ ◦

Examples

`using cout std main std::cout cout` ◦

```
#include <iostream>
int main() {
    using std::cout;
    cout << "Hello, world!\n";
}
```

`using` ◦ `using std::cout` ◦

◦ `d1.foo` `Derived1::foo(const char*)` ◦ `Base::foo(int)` ◦ `d2.foo(42)` `using` `Base::foo(int)` `Derived2::foo` ◦
`foo` `Base::foo` ◦

```
struct Base {
    void foo(int);
};
struct Derived1 : Base {
    void foo(const char*);
};
struct Derived2 : Base {
    using Base::foo;
    void foo(const char*);
};
int main() {
    Derived1 d1;
    d1.foo(42); // error
    Derived2 d2;
    d2.foo(42); // OK
}
```

C++ 11

`using` ◦ ◦

```
struct Base {
    Base(int x, const char* s);
};
```

```
struct Derived1 : Base {
    Derived1(int x, const char* s) : Base(x, s) {}
};
struct Derived2 : Base {
    using Base::Base;
};
int main() {
    Derived1 d1(42, "Hello, world");
    Derived2 d2(42, "Hello, world");
}
```

Derived1Derived2Base◦ Derived1Derived2 **C ++ 11**◦

<https://riptutorial.com/zh-TW/cplusplus/topic/9301/>

41:

Examples

try/catch◦ try catch◦

```
#include <iostream>
#include <string>
#include <stdexcept>

int main() {
    std::string str("foo");

    try {
        str.at(10); // access element, may throw std::out_of_range
    } catch (const std::out_of_range& e) {
        // what() is inherited from std::exception and contains an explanatory message
        std::cout << e.what();
    }
}
```

catch◦ catch

```
std::string str("foo");

try {
    str.reserve(2); // reserve extra capacity, may throw std::length_error
    str.at(10); // access element, may throw std::out_of_range
} catch (const std::length_error& e) {
    std::cout << e.what();
} catch (const std::out_of_range& e) {
    std::cout << e.what();
}
```

catch◦ catchstd::length_errorstd::out_of_rangestd::exception

```
std::string str("foo");

try {
    str.reserve(2); // reserve extra capacity, may throw std::length_error
    str.at(10); // access element, may throw std::out_of_range
} catch (const std::exception& e) {
    std::cout << e.what();
}
```

catchcatch

```
try {
    /* Code throwing exceptions omitted. */
} catch (const std::exception& e) {
    /* Handle all exceptions of type std::exception. */
} catch (const std::runtime_error& e) {
    /* This block of code will never execute, because std::runtime_error inherits
```

```
    from std::exception, and all exceptions of type std::exception were already
    caught by the previous catch clause. */
}
```

catch-all

```
try {
    throw 10;
} catch (...) {
    std::cout << "caught an exception";
}
```

Rethrow

◦

```
try {
    ... // some code here
} catch (const SomeException& e) {
    std::cout << "caught an exception";
    throw;
}
```

throw;◦

C++ 11

std::exception_ptr C++rethrow_exception<exception>◦

```
#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>

void handle_eptr(std::exception_ptr eptr) // passing by value is ok
{
    try {
        if (eptr) {
            std::rethrow_exception(eptr);
        }
    } catch(const std::exception& e) {
        std::cout << "Caught exception \"" << e.what() << "\"\n";
    }
}

int main()
{
    std::exception_ptr eptr;
    try {
        std::string().at(1); // this generates an std::out_of_range
    } catch(...) {
        eptr = std::current_exception(); // capture
    }
    handle_eptr(eptr);
} // destructor for std::out_of_range called here, when the eptr is destructed
```

Try Blocks

```
struct A : public B
{
    A() try : B(), foo(1), bar(2)
    {
        // constructor body
    }
    catch (...)
    {
        // exceptions from the initializer list and constructor are caught here
        // if no exception is thrown here
        // then the caught exception is re-thrown.
    }

private:
    Foo foo;
    Bar bar;
};
```

```
void function_with_try_block()
try
{
    // try block body
}
catch (...)
{
    // catch block body
}
```

```
void function_with_try_block()
{
    try
    {
        // try block body
    }
    catch (...)
    {
        // catch block body
    }
}
```

catchcatch.

maintrymaintry. std::terminate.

```
struct A
{
    ~A() noexcept(false) try
    {
        // destructor body
    }
    catch (...)
    {
        // exceptions of destructor body are caught here
        // if no exception is thrown here
    }
};
```

```

        // then the caught exception is re-thrown.
    }
};

```

std::terminate ◦

const

const◦

```

try {
    // throw new std::runtime_error("Error!"); // Don't do this!
    // This creates an exception object
    // on the heap and would require you to catch the
    // pointer and manage the memory yourself. This can
    // cause memory leaks!

    throw std::runtime_error("Error!");
} catch (const std::runtime_error& e) {
    std::cout << e.what() << std::endl;
}

```

catchcatch◦ ◦ throw e; ;throw e;

```

#include <iostream>

struct BaseException {
    virtual const char* what() const { return "BaseException"; }
};

struct DerivedException : BaseException {
    // "virtual" keyword is optional here
    virtual const char* what() const { return "DerivedException"; }
};

int main(int argc, char** argv) {
    try {
        try {
            throw DerivedException();
        } catch (const BaseException& e) {
            std::cout << "First catch block: " << e.what() << std::endl;
            // Output ==> First catch block: DerivedException

            throw e; // This changes the exception to BaseException
                    // instead of the original DerivedException!
        }
    } catch (const BaseException& e) {
        std::cout << "Second catch block: " << e.what() << std::endl;
        // Output ==> Second catch block: BaseException
    }
    return 0;
}

```

const◦ ◦

catch◦ logic_error runtime_errorbad_alloc I/O◦

C++ 11

Web。◦ callstack。◦

◦

std::nested_exceptionstd::throw_with_nested

```
#include <stdexcept>
#include <exception>
#include <string>
#include <fstream>
#include <iostream>

struct MyException
{
    MyException(const std::string& message) : message(message) {}
    std::string message;
};

void print_current_exception(int level)
{
    try {
        throw;
    } catch (const std::exception& e) {
        std::cerr << std::string(level, ' ') << "exception: " << e.what() << '\n';
    } catch (const MyException& e) {
        std::cerr << std::string(level, ' ') << "MyException: " << e.message << '\n';
    } catch (...) {
        std::cerr << "Unkown exception\n";
    }
}

void print_current_exception_with_nested(int level = 0)
{
    try {
        throw;
    } catch (...) {
        print_current_exception(level);
    }
    try {
        throw;
    } catch (const std::nested_exception& nested) {
        try {
            nested.rethrow_nested();
        } catch (...) {
            print_current_exception_with_nested(level + 1); // recursion
        }
    } catch (...) {
        //Empty // End recursion
    }
}

// sample function that catches an exception and wraps it in a nested exception
void open_file(const std::string& s)
{
    try {
        std::ifstream file(s);
        file.exceptions(std::ios_base::failbit);
    }
}
```

```

    } catch(...) {
        std::throw_with_nested(MyException{"Couldn't open " + s});
    }
}

// sample function that catches an exception and wraps it in a nested exception
void run()
{
    try {
        open_file("nonexistent.file");
    } catch(...) {
        std::throw_with_nested( std::runtime_error("run() failed") );
    }
}

// runs the sample function above and prints the caught exception
int main()
{
    try {
        run();
    } catch(...) {
        print_current_exception_with_nested();
    }
}

```

```

exception: run() failed
MyException: Couldn't open nonexistent.file
exception: basic_ios::clear

```

`std::exception` **exception** `std::exception` ◦

std :: uncaught_exceptions

C++ 17

C++ 17 `int std::uncaught_exceptions() bool std::uncaught_exception() ◦ ◦`

```

#include <exception>
#include <string>
#include <iostream>

// Apply change on destruction:
// Rollback in case of exception (failure)
// Else Commit (success)
class Transaction
{
public:
    Transaction(const std::string& s) : message(s) {}
    Transaction(const Transaction&) = delete;
    Transaction& operator =(const Transaction&) = delete;
    void Commit() { std::cout << message << ": Commit\n"; }
    void RollBack() noexcept(true) { std::cout << message << ": Rollback\n"; }

    // ...

    ~Transaction() {
        if (uncaughtExceptionCount == std::uncaught_exceptions()) {

```

```

        Commit(); // May throw.
    } else { // current stack unwinding
        RollBack();
    }
}

private:
    std::string message;
    int uncaughtExceptionCount = std::uncaught_exceptions();
};

class Foo
{
public:
    ~Foo() {
        try {
            Transaction transaction("In ~Foo"); // Commit,
                                                // even if there is an uncaught exception
            //...
        } catch (const std::exception& e) {
            std::cerr << "exception/~Foo:" << e.what() << std::endl;
        }
    }
};

int main()
{
    try {
        Transaction transaction("In main"); // RollBack
        Foo foo; // ~Foo commit its transaction.
        //...
        throw std::runtime_error("Error");
    } catch (const std::exception& e) {
        std::cerr << "exception/main:" << e.what() << std::endl;
    }
}

```

```

In ~Foo: Commit
In main: Rollback
exception/main:Error

```

o

std::exception o std::exception

```

#include <exception>

class Except: virtual public std::exception {
protected:

    int error_number;           ///< Error number
    int error_offset;          ///< Error offset
    std::string error_message;  ///< Error message

public:

    /** Constructor (C++ STL string, int, int).

```

```

    * @param msg The error message
    * @param err_num Error number
    * @param err_off Error offset
    */
explicit
Except(const std::string& msg, int err_num, int err_off):
    error_number(err_num),
    error_offset(err_off),
    error_message(msg)
    {}

/** Destructor.
 * Virtual to allow for subclassing.
 */
virtual ~Except() throw () {}

/** Returns a pointer to the (constant) error description.
 * @return A pointer to a const char*. The underlying memory
 * is in possession of the Except object. Callers must
 * not attempt to free the memory.
 */
virtual const char* what() const throw () {
    return error_message.c_str();
}

/** Returns error number.
 * @return #error_number
 */
virtual int getErrorNumber() const throw() {
    return error_number;
}

/**Returns error offset.
 * @return #error_offset
 */
virtual int getErrorOffset() const throw() {
    return error_offset;
}
};

```

```

try {
    throw(Except("Couldn't do what you were expecting", -12, -34));
} catch (const Except& e) {
    std::cout<<e.what()
        <<"\nError number: "<<e.getErrorNumber()
        <<"\nError offset: "<<e.getErrorOffset();
}

```

o

std::runtime_error

```

#include <stdexcept>

class Except: virtual public std::runtime_error {
protected:

```

```

int error_number;          ///< Error number
int error_offset;         ///< Error offset

public:

/** Constructor (C++ STL string, int, int).
 * @param msg The error message
 * @param err_num Error number
 * @param err_off Error offset
 */
explicit
Except(const std::string& msg, int err_num, int err_off):
    std::runtime_error(msg)
    {
        error_number = err_num;
        error_offset = err_off;
    }

/** Destructor.
 * Virtual to allow for subclassing.
 */
virtual ~Except() throw () {}

/** Returns error number.
 * @return #error_number
 */
virtual int getErrorNumber() const throw() {
    return error_number;
}

/**Returns error offset.
 * @return #error_offset
 */
virtual int getErrorOffset() const throw() {
    return error_offset;
}

};

```

what() std::runtime_error what()° °

<https://riptutorial.com/zh-TW/cplusplus/topic/1354/>

42:

C++

C++ 0x

Examples

C++ 11

```
#include <mutex>
#include <condition_variable>

class Semaphore {
public:
    Semaphore (int count_ = 0)
    : count(count_)
    {
    }

    inline void notify( int tid ) {
        std::unique_lock<std::mutex> lock(mtx);
        count++;
        cout << "thread " << tid << " notify" << endl;
        //notify the waiting thread
        cv.notify_one();
    }

    inline void wait( int tid ) {
        std::unique_lock<std::mutex> lock(mtx);
        while(count == 0) {
            cout << "thread " << tid << " wait" << endl;
            //wait on the mutex until notify is called
            cv.wait(lock);
            cout << "thread " << tid << " run" << endl;
        }
        count--;
    }

private:
    std::mutex mtx;
    std::condition_variable cv;
    int count;
};
```

◦ ◦ notify_one() ◦

s1count1 ◦ notify ◦

```
int main()
{
    Semaphore sem(1);

    thread s1([&]() {
        while(true) {
            this_thread::sleep_for(std::chrono::seconds(5));
```

```
        sem.wait( 1 );
    }
});
thread s2([&]() {
    while(true){
        sem.wait( 2 );
    }
});
thread s3([&]() {
    while(true) {
        this_thread::sleep_for(std::chrono::milliseconds(600));
        sem.wait( 3 );
    }
});
thread s4([&]() {
    while(true) {
        this_thread::sleep_for(std::chrono::seconds(5));
        sem.notify( 4 );
    }
});

s1.join();
s2.join();
s3.join();
s4.join();

...
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/9785/>

43:

Examples

◦ ◦

“”◦

```
struct Inner {int i;};

const int NUM_INNER = 5;
class Value
{
private:
    Inner *array_; //Normally has reference semantics.

public:
    Value() : array_(new Inner[NUM_INNER]){}

    ~Value() {delete[] array_;}

    Value(const Value &val) : array_(new Inner[NUM_INNER])
    {
        for(int i = 0; i < NUM_INNER; ++i)
            array_[i] = val.array_[i];
    }

    Value &operator=(const Value &val)
    {
        for(int i = 0; i < NUM_INNER; ++i)
            array_[i] = val.array_[i];
        return *this;
    }
};
```

C++ 11

Value “”◦

```
struct Inner {int i;};

constexpr auto NUM_INNER = 5;
class Value
{
private:
    Inner *array_; //Normally has reference semantics.

public:
    Value() : array_(new Inner[NUM_INNER]){}

    //OK to delete even if nullptr
    ~Value() {delete[] array_;}

    Value(const Value &val) : array_(new Inner[NUM_INNER])
    {
```



```

    for(int i = 0; i < NUM_INNER; ++i)
        array_[i] = val.array_[i];
}

Value &operator=(const Value &val)
{
    for(int i = 0; i < NUM_INNER; ++i)
        array_[i] = val.array_[i];
    return *this;
}

//Movement means no memory allocation.
//Cannot throw exceptions.
Value(Value &&val) noexcept : array_(val.array_)
{
    //We've stolen the old value.
    val.array_ = nullptr;
}

//Cannot throw exceptions.
Value &operator=(Value &&val) noexcept
{
    //Clever trick. Since `val` is going to be destroyed soon anyway,
    //we swap his data with ours. His destructor will destroy our data.
    std::swap(array_, val.array_);
}
};

```

o

```

struct Inner {int i;};

constexpr auto NUM_INNER = 5;
class Value
{
private:
    Inner *array_; //Normally has reference semantics.

public:
    Value() : array_(new Inner[NUM_INNER]){}

    //OK to delete even if nullptr
    ~Value() {delete[] array_;}

    Value(const Value &val) = delete;
    Value &operator=(const Value &val) = delete;

    //Movement means no memory allocation.
    //Cannot throw exceptions.
    Value(Value &&val) noexcept : array_(val.array_)
    {
        //We've stolen the old value.
        val.array_ = nullptr;
    }

    //Cannot throw exceptions.
    Value &operator=(Value &&val) noexcept
    {
        //Clever trick. Since `val` is going to be destroyed soon anyway,
        //we swap his data with ours. His destructor will destroy our data.

```

```

    std::swap(array_, val.array_);
}
};

```

unique_ptrZero

```

struct Inner {int i;};

constexpr auto NUM_INNER = 5;
class Value
{
private:
    unique_ptr<Inner []>array_; //Move-only type.

public:
    Value() : array_(new Inner[NUM_INNER]){}

    //No need to explicitly delete. Or even declare.
    ~Value() = default; {delete[] array_;}

    //No need to explicitly delete. Or even declare.
    Value(const Value &val) = default;
    Value &operator=(const Value &val) = default;

    //Will perform an element-wise move.
    Value(Value &&val) noexcept = default;

    //Will perform an element-wise move.
    Value &operator=(Value &&val) noexcept = default;
};

```

◦ ◦

C++

```

int i = 5;
int j = i; //Copied
j += 20;
std::cout << i; //Prints 5; i is unaffected by changes to j.

```

```

std::vector<int> v1(5, 12); //array of 5 values, 12 in each.
std::vector<int> v2 = v1; //Copies the vector.
v2[3] = 6; v2[4] = 9;
std::cout << v1[3] << " " << v1[4]; //Writes "12 12", since v1 is unchanged.

```

◦

C++

```

int *pi = new int(4);
int *pi2 = pi;
pi = new int(16);
assert(pi2 != pi); //Will always pass.

int *pj = pi;

```

```
*pj += 5;  
std::cout << *pi; //Writes 9, since `pi` and `pj` reference the same object.
```

C++。

<https://riptutorial.com/zh-TW/cplusplus/topic/1955/>

44:

Examples

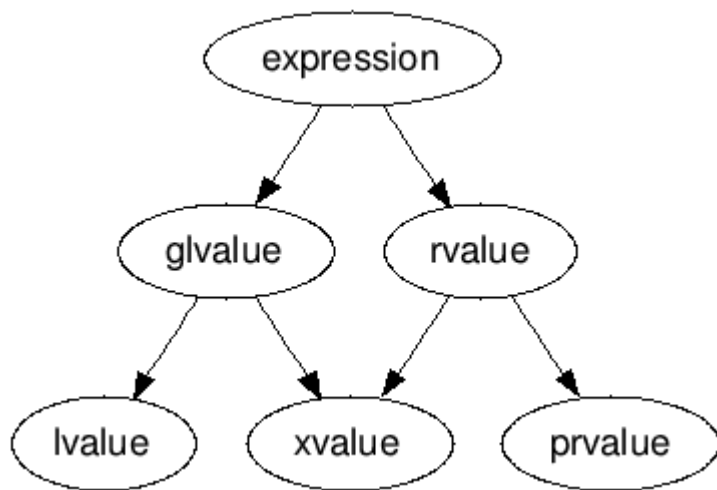
C++. C++.

o o o o

o r.

C++3xvalueprvalue. C++.

C++glvaluevalue. o



prvalue

prvaluevalue.

- std::string("123").
-
- -1 true 0.5f'a'
- lambda

& o

X

xvalueeXpiring. xvalue“eXpiring”.

```
struct X { int n; };  
extern X x;  
  
4; // prvalue: does not have an identity  
x; // lvalue
```

```

x.n;                // lvalue
std::move(x);      // xvalue
std::forward<X&>(x); // lvalue
X{4};              // prvalue: does not have an identity
X{4}.n;            // xvalue: does have an identity and denotes resources
                    // that can be reused

```

◦ ◦

```

struct X { ... };

X x;                // x is an lvalue
X* px = &x;         // px is an lvalue
*px = X{};          // *px is also an lvalue, X{} is a prvalue

X* foo_ptr();       // foo_ptr() is a prvalue
X& foo_ref();        // foo_ref() is an lvalue

```

4 'x'prvalues◦

glvalue

glvalue""◦ xvaluesprvalues◦

glvalue

```

struct X { int n; };
X foo();

X x;
x; // has a name, so it's a glvalue
std::move(x); // has a name (we're moving from "x"), so it's a glvalue
                // can be moved from, so it's an xvalue not an lvalue

foo(); // has no name, so is a prvalue, not a glvalue
X{};   // temporary has no name, so is a prvalue, not a glvalue
X{}.n; // HAS a name, so is a glvalue. can be moved from, so it's an xvalue

```

rvalue◦

rvalueT &&Texpr◦ rvalue;◦ ◦

rvaluexvalueprvalue◦

std::move◦ xvalueprvaluestd::move identityxvalue◦

```

std::string str("init"); //1
std::string test1(str);  //2
std::string test2(std::move(str)); //3

str = std::string("new value"); //4
std::string &&str_ref = std::move(str); //5
std::string test3(str_ref); //6

```

`std::string&&""` `str` **rvalue** `const std::string&` **overload**

3 `std::move` `T&& T` `std::move(str)` `std::string&&` `xvalue` `std::string` **move** **3** `str`

4 `std::string` `std::string&&` `std::string("new value")` **rvalue** **prvalue** `str`

5 `str_ref` **rvalue** `str`

`str_ref` `std::string` **rvalue** `str_ref` **rvalue** `str_ref` `std::string` **move** `str_ref` **6** `test3` `str` `test3`

`std::move`

<https://riptutorial.com/zh-TW/cplusplus/topic/763/>

45:

- `as-if`

Examples

/

- ◦

```
// source:  
  
int process(int value)  
{  
    return 2 * value;  
}  
  
int foo(int a)  
{  
    return process(a);  
}  
  
// program, after inlining:  
  
int foo(int a)  
{  
    return 2 * a; // the body of process() is copied into foo()  
}
```

-

classclassstruct 1◦

class

```
#include <cassert>  
  
struct Base {}; // empty class  
  
struct Derived1 : Base {  
    int i;  
};  
  
int main() {  
    // the size of any object of empty class type is at least 1  
    assert(sizeof(Base) == 1);  
  
    // empty base optimization applies  
    assert(sizeof(Derived1) == sizeof(int));  
}
```

`std::vector` `std::function` `std::shared_ptr` `vector` `begin` `endcapacity`

[cppreference](#)

<https://riptutorial.com/zh-TW/cplusplus/topic/9767/>

46:

C++

◦

constexpr

◦ ◦

Examples

◦

```
#include <iostream>

template<unsigned int n>
struct factorial
{
    enum
    {
        value = n * factorial<n - 1>::value
    };
};

template<>
struct factorial<0>
{
    enum { value = 1 };
};

int main()
{
    std::cout << factorial<7>::value << std::endl;    // prints "5040"
}
```

factorial::type::type::value

factorial::value

◦ factorial<0> ""

◦

```
int my_array[factorial<5>::value];
```

◦ ◦

◦ g++256 g++ -ftemplate-depth-X

C++ 11

C++ 11 `std::integral_constant`

```
#include <iostream>
#include <type_traits>

template<long long n>
struct factorial :
    std::integral_constant<long long, n * factorial<n - 1>::value> {};

template<>
struct factorial<0> :
    std::integral_constant<long long, 1> {};

int main()
{
    std::cout << factorial<7>::value << std::endl;    // prints "5040"
}
```

`constexpr`

```
#include <iostream>

constexpr long long factorial(long long n)
{
    return (n == 0) ? 1 : n * factorial(n - 1);
}

int main()
{
    char test[factorial(3)];
    std::cout << factorial(7) << '\n';
}
```

`factorial()` C++ 11 `constexpr`

C++ 14

C++ 14 `constexpr`

```
constexpr long long factorial(long long n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

```
constexpr long long factorial(int n)
{
    long long result = 1;
    for (int i = 1; i <= n; ++i) {
        result *= i;
    }
    return result;
}
```

C++ 17

C++ 17fold

```
#include <iostream>
#include <utility>

template <class T, T N, class I = std::make_integer_sequence<T, N>>
struct factorial;

template <class T, T N, T... Is>
struct factorial<T,N,std::index_sequence<T, Is...>> {
    static constexpr T value = (static_cast<T>(1) * ... * (Is + 1));
};

int main() {
    std::cout << factorial<int, 5>::value << std::endl;
}
```

◦ C++ 17 ◦

C++ 11

```
void print_all(std::ostream& os) {
    // base case
}

template <class T, class... Ts>
void print_all(std::ostream& os, T const& first, Ts const&... rest) {
    os << first;

    print_all(os, rest...);
}
```

◦

C++ 11

```
template <class... Ts>
void print_all(std::ostream& os, Ts const&... args) {
    using expander = int[];
    (void)expander{0,
        (void(os << args), 0)...
    };
}
```

TC ◦

C++ 17

C++ 17◦ fold-expression

```
template <class... Ts>
void print_all(std::ostream& os, Ts const&... args) {
    ((os << args), ...);
}
```

if constexpr

```
template <class T, class... Ts>
void print_all(std::ostream& os, T const& first, Ts const&... rest) {
    os << first;

    if constexpr (sizeof...(rest) > 0) {
        // this line will only be instantiated if there are further
        // arguments. if rest... is empty, there will be no call to
        // print_all(os).
        print_all(os, rest...);
    }
}
```

std :: integer_sequence

C++ 14

```
template <class T, T... Ints>
class integer_sequence;

template <std::size_t... Ints>
using index_sequence = std::integer_sequence<std::size_t, Ints...>;
```

```
template <class T, T N>
using make_integer_sequence = std::integer_sequence<T, /* a sequence 0, 1, 2, ..., N-1 */ >;

template<std::size_t N>
using make_index_sequence = make_integer_sequence<std::size_t, N>;
```

C++ 14 C++ 11。

std::tuple C++ 17 std::apply

```
namespace detail {
    template <class F, class Tuple, std::size_t... Is>
    decltype(auto) apply_impl(F&& f, Tuple&& tpl, std::index_sequence<Is...> ) {
        return std::forward<F>(f) (std::get<Is>(std::forward<Tuple>(tpl))...);
    }
}

template <class F, class Tuple>
decltype(auto) apply(F&& f, Tuple&& tpl) {
    return detail::apply_impl(std::forward<F>(f),
        std::forward<Tuple>(tpl),
        std::make_index_sequence<std::tuple_size<std::decay_t<Tuple>>::value>{});
}

// this will print 3
int f(int, char, double);

auto some_args = std::make_tuple(42, 'x', 3.14);
int r = apply(f, some_args); // calls f(42, 'x', 3.14)
```

- `std::advance()`

```

namespace details {
    template <class RAIter, class Distance>
    void advance(RAIter& it, Distance n, std::random_access_iterator_tag) {
        it += n;
    }

    template <class BidirIter, class Distance>
    void advance(BidirIter& it, Distance n, std::bidirectional_iterator_tag) {
        if (n > 0) {
            while (n--) ++it;
        }
        else {
            while (n++) --it;
        }
    }

    template <class InputIter, class Distance>
    void advance(InputIter& it, Distance n, std::input_iterator_tag) {
        while (n--) {
            ++it;
        }
    }
}

template <class Iter, class Distance>
void advance(Iter& it, Distance n) {
    details::advance(it, n,
        typename std::iterator_traits<Iter>::iterator_category{} );
}

```

`details::advance``std::XY_iterator_tag` ◦ `details::advance` ◦

`advance``iterator_traits<T>::iterator_category``iterator_tag``Iter` ◦ `iterator_category<Iter>::type`
`details::advance` ◦ `struct` ◦

SFINAE`enable_if` ◦

C++ 17 *if constexpr*`advance` ◦

- `std::hash`

```

#include <functional> // for std::hash
#include <type_traits> // for std::false_type and std::true_type
#include <utility> // for std::declval

template<class, class = void>
struct has_hash
    : std::false_type
{};

template<class T>
struct has_hash<T, decltype(std::hash<T>() (std::declval<T>()), void())>
    : std::true_type
{};

```

C++ 17

C++ 17 `std::void_t`

```
#include <functional> // for std::hash
#include <type_traits> // for std::false_type, std::true_type, std::void_t
#include <utility> // for std::declval

template<class, class = std::void_t<>>
struct has_hash
    : std::false_type
{};

template<class T>
struct has_hash<T, std::void_t<decltype(std::hash<T>())(std::declval<T>())>>
    : std::true_type
{};
```

`std::void_t`

```
template< class... > using void_t = void;
```

`operator<`

```
template<class, class = void>
struct has_less_than
    : std::false_type
{};

template<class T>
struct has_less_than<T, decltype(std::declval<T>() < std::declval<T>()), void()>
    : std::true_type
{};
```

`Tstd::hashstd::unordered_map<T> std::map<T>`

```
template <class K, class V>
using hash_invariant_map = std::conditional_t<
    has_hash<K>::value,
    std::unordered_map<K, V>,
    std::map<K, V>>;
```

C++ 11

C++ 11。

```
template <typename T>
constexpr T calculatePower(T value, unsigned power) {
    return power == 0 ? 1 : value * calculatePower(value, power-1);
}
```

`constexprconstexpr。`

`C++ 11 constexprreturn。`

◦ C++◦

```
void useExample() {
    constexpr int compileTimeCalculated = calculatePower(3, 3); // computes at compile time,
                                                                // as both arguments are known at compilation time
                                                                // and used for a constant expression.

    int value;
    std::cin >> value;
    int runtimeCalculated = calculatePower(value, 3); // runtime calculated,
                                                    // because value is known only at runtime.
}
```

C++ 17

fold

```
#include <iostream>
#include <utility>

template <class T, T V, T N, class I = std::make_integer_sequence<T, N>>
struct power;

template <class T, T V, T N, T... Is>
struct power<T, V, N, std::integer_sequence<T, Is...>> {
    static constexpr T value = (static_cast<T>(1) * ... * (V * static_cast<bool>(Is + 1)));
};

int main() {
    std::cout << power<int, 4, 2>::value << std::endl;
}
```

T

`std::enable_if` **SFINAE** T◦

`true``false``std::true_type``std::false_type` ◦

T `is_pointer``std::is_pointer`

```
template <typename T>
struct is_pointer_: std::false_type {};

template <typename T>
struct is_pointer_<T*>: std::true_type {};

template <typename T>
struct is_pointer: is_pointer_<typename std::remove_cv<T>::type> { }
```

1. `is_pointer_ std::false_type` ◦ `std::false_type` “false” ◦
2. `T*is_pointer_T` ◦ `std::true_type` ◦
3. `Tconstvolatile` ◦

`is_pointer<T>`

- Use `::value is_pointer<int>::value - valuestd::true_typestd::false_typebool;`
- `is_pointer<int>{} - std::is_pointerstd::true_typestd::false_typeconstexprstd::true_typestd::false_typestd::false_typeconstexprbool ◦`

“”

```
template <typename T>
constexpr bool is_pointer_v = is_pointer<T>::value;
```

C++ 17

C++ 17_v

```
template< class T > constexpr bool is_pointer_v = is_pointer<T>::value;
template< class T > constexpr bool is_reference_v = is_reference<T>::value;
```

IF-THEN-ELSE

C++ 11

`<type_traits>std::conditional`

```
template<typename T>
struct ValueOrPointer
{
    typename std::conditional<(sizeof(T) > sizeof(void*)), T*, T>::type vop;
};
```

`T;T◦ sizeof(ValueOrPointer)<= sizeof(void*) ◦`

/

C++ 11

`min ◦ min ◦`

```
template <typename T1, typename T2>
auto min(const T1 &a, const T2 &b)
-> typename std::common_type<const T1&, const T2&>::type
{
    return a < b ? a : b;
}

template <typename T1, typename T2, typename ... Args>
auto min(const T1 &a, const T2 &b, const Args& ... args)
-> typename std::common_type<const T1&, const T2&, const Args& ...>::type
{
    return min(min(a, b), args...);
}

auto minimum = min(4, 5.8f, 3, 1.8, 3, 1.1, 9);
```


47:

- `:: opt new opt new-type-id new-initializer opt`
- `:: opt new opt type-id new-initializer opt`
- `:: opt cast-expression`
- `:: opt delete [] cast-expression`
- `std :: unique_ptr < type-id > var_namenew type-id opt ; // C ++ 11`
- `std :: shared_ptr < type-id > var_namenew type-id opt ; // C ++ 11`
- `std :: shared_ptr < type-id > var_name = std :: make_shared < type-id > opt ; // C ++ 11`
- `std :: unique_ptr < type-id > var_name = std :: make_unique < type-id > opt ; // C ++ 14`

`::newdelete`
`newdelete`。

`new`
`placement new`。

`new Foo auto C ++ 11 decltype (auto) C ++ 14`。

• `iso`。

`delete [] new-expression new-expression []`。

```
using IA = int[4];
int* pIA = new IA;
delete[] pIA; // right
// delete pIA; // wrong
```

Examples

• •

```
int main() {
    int a = 0; //Stored on the stack
    return a;
}
```

“”。

```
float bar() {
    //f will be placed on the stack after anything else
    float f = 2;
    return f;
}

double foo() {
    //d will be placed just after anything within main()
    double d = bar();
    return d;
}
```

```
int main() {
    //The stack has no user variables stored in it until foo() is called
    return (int)foo();
}
```

◦

```
int* pA = nullptr;

void foo() {
    int b = *pA;
    pA = &b;
}

int main() {
    int a = 5;
    pA = &a;
    foo();
    //Undefined behavior, the value pointed to by pA is no longer in scope
    a = *pA;
}
```

“”。

C++ ◦

Free Store。 Free Store。

newdelete。

```
float *foo = nullptr;
{
    *foo = new float; // Allocates memory for a float
    float bar;        // Stack allocated
} // End lifetime of bar, while foo still alive

delete foo;          // Deletes the memory for the float at pF, invalidating the pointer
foo = nullptr;      // Setting the pointer to nullptr after delete is often considered good
practice
```

newdelete。 ◦ ◦

```
// Allocates memory for an array of 256 ints
int *foo = new int[256];
// Deletes an array of 256 ints at foo
delete[] foo;
```

newdelete *mallocfree*。 *newdelete* *mallocfree*。

```
struct ComplexType {
    int a = 0;
```

```

    ComplexType() { std::cout << "Ctor" << std::endl; }
    ~ComplexType() { std::cout << "Dtor" << std::endl; }
};

// Allocates memory for a ComplexType, and calls its constructor
ComplexType *foo = new ComplexType();
//Calls the destructor for ComplexType() and deletes memory for a ComplexType at pC
delete foo;

```

C++ 11

C++ 11.

C++ 14

C++ 14 `std::make_unique` **STL** `std::make_shared` *naked new/delete*.

Free Store `new`.

Placement New `'new'`

```

int a4byteInteger;

char *a4byteChar = new (&a4byteInteger) char[4];

```

`a4byteChar` `a4byteInteger` `'stack'`.

◦ `a4byteInteger` `'delete a4byteChar'`.

◦

```

int *a8byteDynamicInteger = new int[2];

char *a8byteChar = new (a8byteDynamicInteger) char[8];

```

`a8byteChar` `a8byteDynamicInteger` ◦ `delete a8byteDynamicInteger`

C++

```

struct ComplexType {
    int a;

    ComplexType() : a(0) {}
    ~ComplexType() {}
};

int main() {
    char* dynArray = new char[256];

    //Calls ComplexType's constructor to initialize memory as a ComplexType
    new((void*)dynArray) ComplexType();

    //Clean up memory once we're done

```

```
reinterpret_cast<ComplexType*>(dynArray)->~ComplexType();
delete[] dynArray;

//Stack memory can also be used with placement new
alignas(ComplexType) char localArray[256]; //alignas() available since C++11

new((void*)localArray) ComplexType();

//Only need to call the destructor for stack memory
reinterpret_cast<ComplexType*>(localArray)->~ComplexType();

return 0;
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2873/>

48:

inline° ° °

- `inline function_declaration`
- `inline function_definition`
- `class {function_definition};`

° °

inline° inline - ° ° inline°

C ++inline° ODR ;° inline°

C ++/'inline'

° ° ° °

C ++/'inline'

inline°

/

° ° °

-
- `'inline'`
 -

Examples

```
inline int add(int x, int y);
```

```
inline int add(int x, int y)
{
    return x + y;
}
```

```
// header (.hpp)
struct A
{
    void i_am_inlined()
    {
```

```
    }  
};  
  
struct B  
{  
    void i_am_NOT_inlined();  
};  
  
// source (.cpp)  
void B::i_am_NOT_inlined()  
{  
}
```

```
inline int add(int x, int y)  
{  
    return x + y;  
}  
  
int main()  
{  
    int a = 1, b = 2;  
    int c = add(a, b);  
}
```

add

```
int main()  
{  
    int a = 1, b = 2;  
    int c = a + b;  
}
```

◦ add ◦ - ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/7150/>

49:

◦ ◦

Examples

inline ◦ ◦ C ++ 17.cpp Foo::num_instances C ++ 17 Foo::num_instances Foo::num_instances int ◦

```
// warning: not thread-safe...
class Foo {
public:
    Foo() { ++num_instances; }
    ~Foo() { --num_instances; }
    inline static int num_instances = 0;
};
```

constexpr ◦

```
class MyString {
public:
    MyString() { /* ... */ }
    // ...
    static constexpr int max_size = INT_MAX / 2;
};
// in C++14, this definition was required in a single translation unit:
// constexpr int MyString::max_size;
```

<https://riptutorial.com/zh-TW/cplusplus/topic/9265/>

50:

Examples

◦ ◦

```
enum myEnum
{
    enumName1,
    enumName2,
};
```

◦ myEnum ◦ ◦

◦ ::◦ enumName1 ◦

C++ 11

◦ enumName1myEnum::enumName1 ◦

01◦ enumName10enumName21◦

◦ + 1◦

```
enum myEnum
{
    enumName1 = 1, // value will be 1
    enumName2 = 2, // value will be 2
    enumName3,    // value will be 3, previous value + 1
    enumName4 = 7, // value will be 7
    enumName5,    // value will be 8
    enumName6 = 5, // value will be 5, legal to go backwards
    enumName7 = 3, // value will be 3, legal to reuse numbers
    enumName8 = enumName4 + 2, // value will be 9, legal to take prior enums and adjust them
};
```

switch

switch◦ switch◦

```
enum State {
    start,
    middle,
    end
};

...

switch(myState) {
    case start:
        ...
}
```

```

    case middle:
        ...
} // warning: enumeration value 'end' not handled in switch [-Wswitch]

```

o

- enum

```

enum E {
    Begin,
    E1 = Begin,
    E2,
    // ..
    En,
    End
};

for (E e = E::Begin; e != E::End; ++e) {
    // Do job with e
}

```

C++ 11

enum class operator ++

```

E& operator ++ (E& e)
{
    if (e == E::End) {
        throw std::out_of_range("for E& operator ++ (E&)");
    }
    e = E(static_cast<std::underlying_type<E>::type>(e) + 1);
    return e;
}

```

- std::vector

```

enum E {
    E1 = 4,
    E2 = 8,
    // ..
    En
};

std::vector<E> build_all_E()
{
    const E all[] = {E1, E2, /*..*/ En};

    return std::vector<E>(all, all + sizeof(all) / sizeof(E));
}

std::vector<E> all_E = build_all_E();

```

```

for (std::vector<E>::const_iterator it = all_E.begin(); it != all_E.end(); ++it) {
    E e = *it;
    // Do job with e;
}

```

C++ 11

- `std::initializer_list`

```
enum E {
    E1 = 4,
    E2 = 8,
    // ..
    En
};

constexpr std::initializer_list<E> all_E = {E1, E2, /*..*/ En};
```

```
for (auto e : all_E) {
    // Do job with e
}
```

Scoped enums

C++ 11 ◦ `enumname::membername` ◦ `enum class` ◦ `enum class` ◦

```
enum class rainbow {
    RED,
    ORANGE,
    YELLOW,
    GREEN,
    BLUE,
    INDIGO,
    VIOLET
};
```

```
rainbow r = rainbow::INDIGO;
```

enum class **es**int ◦ `int x = rainbow::RED` ◦

Scoped ◦ `int` ◦ Tic-Tac-Toe

```
enum class piece : char {
    EMPTY = '\0',
    X = 'X',
    O = 'O',
};
```

enum ◦

C++ 11

```
...
enum class Status; // Forward declaration
Status doWork(); // Use the forward declaration
...
enum class Status { Invalid, Success, Fail };
Status doWork() // Full declaration required for implementation
```

```
{  
    return Status::Success;  
}
```

```
...  
enum Status: int; // Forward declaration, explicit type required  
Status doWork(); // Use the forward declaration  
...  
enum Status: int{ Invalid=0, Success, Fail }; // Must match forward declare type  
static_assert( Success == 1 );
```

Blind fruit merchant

<https://riptutorial.com/zh-TW/cplusplus/topic/2796/>

51:

Examples

gccgprof

GNU gprof profiler [gprof](#).

- 1.
- 2.
3. gprof

-pg

```
$ gcc -pg *.cpp -o app
```

```
$ gcc -O2 -pg *.cpp -o app
```

o

app

```
$ ./app
```

gmon.out

```
$ gprof app gmon.out
```

o

```
$ gprof app gmon.out | less
```

o

o

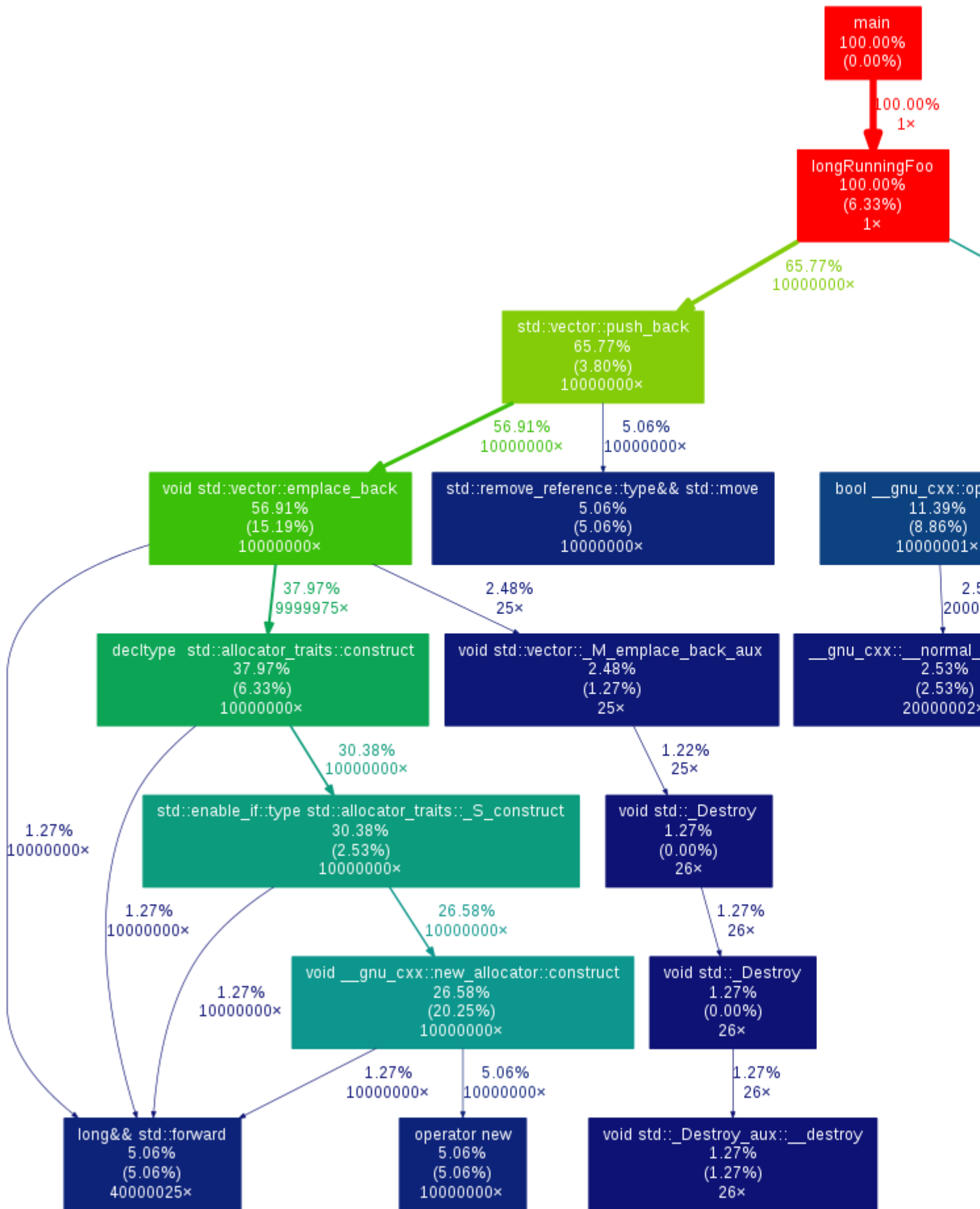
gperf2dot

o o

[gperf2dot](#)Linux perfcallgrindoprofile. gprof

```
# compile with profiling flags
g++ *.cpp -pg
```

```
# run to generate profiling data
./main
# translate profiling data to text, create image
gprof ./main | gprof2dot -s | dot -Tpng -o output.png
```



gccGoogle Perf ToolsCPU

1. Google Perf
- 2.
3. libprofiler
4. pprof

```
# compile code
g++ -O3 -std=c++11 main.cpp -o main

# run with profiler
LD_PRELOAD=/usr/local/lib/libprofiler.so CPUPROFILE=main.prof CPUPROFILE_FREQUENCY=100000
./main
```

- CPUPROFILE
- CPUPROFILE_FREQUENCY;

pprof

```
$ pprof --text ./main main.prof
PROFILE: interrupts/evictions/bytes = 67/15/2016
pprof --text --lines ./main main.prof
Using local file ./main.
Using local file main.prof.
Total: 67 samples
 22 32.8% 32.8%      67 100.0% longRunningFoo ??:0
 20 29.9% 62.7%      20 29.9% __memmove_ssse3_back /build/eglibc-3GlaMS/eglibc-
2.19/string/./sysdeps/x86_64/multiarch/memcpy-ssse3-back.S:1627
  4 6.0% 68.7%       4  6.0% __memmove_ssse3_back /build/eglibc-3GlaMS/eglibc-
2.19/string/./sysdeps/x86_64/multiarch/memcpy-ssse3-back.S:1619
  3 4.5% 73.1%       3  4.5% __random_r /build/eglibc-3GlaMS/eglibc-
2.19/stdlib/random_r.c:388
  3 4.5% 77.6%       3  4.5% __random_r /build/eglibc-3GlaMS/eglibc-
2.19/stdlib/random_r.c:401
  2 3.0% 80.6%       2  3.0% __munmap /build/eglibc-3GlaMS/eglibc-
2.19/misc/./sysdeps/unix/syscall-template.S:81
  2 3.0% 83.6%      12 17.9% __random /build/eglibc-3GlaMS/eglibc-
2.19/stdlib/random.c:298
  2 3.0% 86.6%       2  3.0% __random_r /build/eglibc-3GlaMS/eglibc-
2.19/stdlib/random_r.c:385
  2 3.0% 89.6%       2  3.0% rand /build/eglibc-3GlaMS/eglibc-2.19/stdlib/rand.c:26
  1 1.5% 91.0%       1  1.5% __memmove_ssse3_back /build/eglibc-3GlaMS/eglibc-
2.19/string/./sysdeps/x86_64/multiarch/memcpy-ssse3-back.S:1617
  1 1.5% 92.5%       1  1.5% __memmove_ssse3_back /build/eglibc-3GlaMS/eglibc-
2.19/string/./sysdeps/x86_64/multiarch/memcpy-ssse3-back.S:1623
  1 1.5% 94.0%       1  1.5% __random /build/eglibc-3GlaMS/eglibc-
2.19/stdlib/random.c:293
  1 1.5% 95.5%       1  1.5% __random /build/eglibc-3GlaMS/eglibc-
2.19/stdlib/random.c:296
  1 1.5% 97.0%       1  1.5% __random_r /build/eglibc-3GlaMS/eglibc-
2.19/stdlib/random_r.c:371
  1 1.5% 98.5%       1  1.5% __random_r /build/eglibc-3GlaMS/eglibc-
2.19/stdlib/random_r.c:381
  1 1.5% 100.0%      1  1.5% rand /build/eglibc-3GlaMS/eglibc-2.19/stdlib/rand.c:28
  0 0.0% 100.0%     67 100.0% __libc_start_main /build/eglibc-3GlaMS/eglibc-
2.19/csu/libc-start.c:287
  0 0.0% 100.0%     67 100.0% _start ??:0
  0 0.0% 100.0%     67 100.0% main ??:0
```



```
0 0.0% 100.0%      14 20.9% rand /build/eglibc-3GlaMS/eglibc-2.19/stdlib/rand.c:27
0 0.0% 100.0%      27 40.3% std::vector::_M_emplace_back_aux ??:0
```

...pdf

```
pprof --pdf ./main main.prof > out.pdf
```

<https://riptutorial.com/zh-TW/cplusplus/topic/5347/>

52:

- ◦
-

Examples

-
-

◦ ◦ ODR ◦ ◦

```
template<typename T>
void f(T*) { }

template<typename T>
void f(T) { }
```

“T”“T *”。

```
template<typename T>
void f(T (*x)[sizeof(T) + sizeof(T)]) { }

template<typename T>
void f(T (*x)[2 * sizeof(T)]) { }
```

<https://riptutorial.com/zh-TW/cplusplus/topic/4164/>

53:

◦

```
void foo(double, double);
void foo(long, long);

//Call foo with 2 ints
foo(1, 2); //Function call is ambiguous - int can be converted into a double/long at the same
time
```

Examples

◦

std::string

```
void print(const std::string &str)
{
    std::cout << "This is a string: " << str << std::endl;
}
```

int◦

```
void print_int(int num)
{
    std::cout << "This is an int: " << num << std::endl;
}
```

```
void print(int num)
{
    std::cout << "This is an int: " << num << std::endl;
}
```

print ◦ std::string int ◦

```
print("Hello world!"); //prints "This is a string: Hello world!"
print(1337);           //prints "This is an int: 1337"
```

```
print("Hello world!");
print_int(1337);
```

◦ ◦

```
void print(int num)
{
    std::cout << "This is an int: " << num << std::endl;
}
void print(double num)
```

```
{
    std::cout << "This is a double: " << num << std::endl;
}
```

print

```
print(5);
```

```
print(static_cast<double>(5));
print(static_cast<int>(5));
print(5.0);
```

```
// WRONG CODE
void print(int num1, int num2 = 0)    //num2 defaults to 0 if not included
{
    std::cout << "These are ints: << num1 << " and " << num2 << std::endl;
}
void print(int num)
{
    std::cout << "This is an int: " << num << std::endl;
}
```

print(17)◦

◦

```
// WRONG CODE
std::string getValue()
{
    return "hello";
}

int getValue()
{
    return 0;
}

int x = getValue();
```

getValue int ◦

cv-qualifier

CV;constvolatileconst volatile◦ this**cv-qualifiers**◦ constvolatileconst volatile◦

CV◦ constconstconstconstconst◦ **CVCV**◦

printconst

```
#include <iostream>

class Integer
{
```

```

public:
    Integer(int i_): i{i_}{}

    void print()
    {
        std::cout << "int: " << i << std::endl;
    }

    void print() const
    {
        std::cout << "const int: " << i << std::endl;
    }

protected:
    int i;
};

int main()
{
    Integer i{5};
    const Integer &ic = i;

    i.print(); // prints "int: 5"
    ic.print(); // prints "const int: 5"
}

```

const const const const / ° const **cv-qualifiers** °

```

class ConstCorrect
{
public:
    void good_func() const
    {
        std::cout << "I care not whether the instance is const." << std::endl;
    }

    void bad_func()
    {
        std::cout << "I can only be called on non-const, non-volatile instances." <<
std::endl;
    }
};

void i_change_no_state(const ConstCorrect& cc)
{
    std::cout << "I can take either a const or a non-const ConstCorrect." << std::endl;
    cc.good_func(); // Good. Can be called from const or non-const instance.
    cc.bad_func(); // Error. Can only be called from non-const instance.
}

void const_incorrect_func(ConstCorrect& cc)
{
    cc.good_func(); // Good. Can be called from const or non-const instance.
    cc.bad_func(); // Good. Can only be called from non-const instance.
}

```

const **mutator** const °

const◦ std::mutex mutable

```
class Integer
{
public:
    Integer(int i_): i{i_}{}

    int get() const
    {
        std::lock_guard<std::mutex> lock{mut};
        return i;
    }

    void set(int i_)
    {
        std::lock_guard<std::mutex> lock{mut};
        i = i_;
    }

protected:
    int i;
    mutable std::mutex mut;
};
```

<https://riptutorial.com/zh-TW/cplusplus/topic/510/>

54:

- `std::<T>`
- `std::atomic_flag`

`std::atomicTriviallyCopyable` ◦ `std::atomic_flag` ◦

Examples

◦

```
#include <thread>
#include <iostream>

//function will add all values including and between 'a' and 'b' to 'result'
void add(int a, int b, int * result) {
    for (int i = a; i <= b; i++) {
        *result += i;
    }
}

int main() {
    //a primitive data type has no thread safety
    int shared = 0;

    //create a thread that may run parallel to the 'main' thread
    //the thread will run the function 'add' defined above with paramters a = 1, b = 100,
    result = &shared
    //analogous to 'add(1,100, &shared);'
    std::thread addingThread(add, 1, 100, &shared);

    //attempt to print the value of 'shared' to console
    //main will keep repeating this until the addingThread becomes joinable
    while (!addingThread.joinable()) {
        //this may cause undefined behavior or print a corrupted value
        //if the addingThread tries to write to 'shared' while the main thread is reading it
        std::cout << shared << std::endl;
    }

    //rejoin the thread at the end of execution for cleaning purposes
    addingThread.join();

    return 0;
}
```

◦

```
#include <atomic>
#include <thread>
#include <iostream>
```

```

//function will add all values including and between 'a' and 'b' to 'result'
void add(int a, int b, std::atomic<int> * result) {
    for (int i = a; i <= b; i++) {
        //atomically add 'i' to result
        result->fetch_add(i);
    }
}

int main() {
    //atomic template used to store non-atomic objects
    std::atomic<int> shared = 0;

    //create a thread that may run parallel to the 'main' thread
    //the thread will run the function 'add' defined above with paramters a = 1, b = 100,
    result = &shared
    //analogous to 'add(1,100, &shared);'
    std::thread addingThread(add, 1, 10000, &shared);

    //print the value of 'shared' to console
    //main will keep repeating this until the addingThread becomes joinable
    while (!addingThread.joinable()) {
        //safe way to read the value of shared atomically for thread safe read
        std::cout << shared.load() << std::endl;
    }

    //rejoin the thread at the end of execution for cleaning purposes
    addingThread.join();

    return 0;
}

```

atomicstore()load()int°

<https://riptutorial.com/zh-TW/cplusplus/topic/3804/>

55:

Examples

using T

- T
- T
- T
- TClassTemplate<TemplateArguments>
- T U U void (*fptr) (A); f(fptr); void(A)
- T void(A)A
- T BA::*p; void (A::*pf) (B); f(p); f(pf); A B void(B)

o o o

f

```
namespace A {
    struct Z { };
    namespace I { void g(Z); }
    using namespace I;

    struct X { struct Y { }; friend void f(Y) { } };
    void f(X p) { }
    void f(std::shared_ptr<X> p) { }
}

// example calls
f(A::X());
f(A::X::Y());
f(std::make_shared<A::X>());

g(A::Z()); // invalid: "using namespace I;" is ignored!
```

<https://riptutorial.com/zh-TW/cplusplus/topic/5163/>

56:

Examples

```
template<class ... Types> struct Tuple {};
```

◦ ◦

parameter_pack ...parameter_pack

```
template<class T> // Base of recursion
void variadic_printer(T last_argument) {
    std::cout << last_argument;
}

template<class T, class ...Args>
void variadic_printer(T first_argument, Args... other_arguments) {
    std::cout << first_argument << "\n";
    variadic_printer(other_arguments...); // Parameter pack expansion
}
```

```
variadic_printer(1, 2, 3, "hello");variadic_printer(1, 2, 3, "hello");
```

```
1
2
3
hello
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7668/>

57:

Examples

const &

```
int i = 10;
int &refi = i;
```

refi

```
refi = 20; // i = 20;
```

```
int i = 10, j = 20;
int &refi = i, &refj = j;

// Common pitfall :
// int& refi = i, k = j;
// refi will be of type int&.
// though, k will be of type int, not int&!
```

o

```
int &i; // error: declaration of reference variable 'i' requires an initializer
```

nullptr

```
int *const ptri = nullptr;
int &refi = nullptr; // error: non-const lvalue reference to type 'int' cannot bind to a
temporary of type 'nullptr_t'
```

C++

C++ReferenceAlias o

o o

```
int main() {
    int i = 10;
    int &j = i;

    cout<<i<<endl;
    cout<<&b<<endl;
    return 0;
}
```

cout

```
void func (int &fParam ) {
    cout<<"Address inside function => "<<fParam<<endl;
}

int main() {
    int i = 10;
    cout<<"Address inside Main => "<<&i<<endl;

    func(i);

    return 0;
}
```

cout◦

C++ References **Alias**◦

```
int &i;
```

◦

<https://riptutorial.com/zh-TW/cplusplus/topic/1548/>

58:

C ++. C ++ 17 STLinvokestd :: functionoperatorlambdas. STL.

Stephan T. Lavavej <> .

Examples

C. C .

```
typedef returnType(*name)(arguments); // All
using name = returnType(*) (arguments); // <= C++11
using name = std::add_pointer<returnType(arguments)>::type; // <= C++11
using name = std::add_pointer_t<returnType(arguments)>; // <= C++14
```

```
using LessThanFunctionPtr = std::add_pointer_t<bool(int, int)>;
void sortVectorInt(std::vector<int>&v, LessThanFunctionPtr lessThan) {
    if (v.size() < 2)
        return;
    if (v.size() == 2) {
        if (!lessThan(v.front(), v.back())) // Invoke the function pointer
            std::swap(v.front(), v.back());
        return;
    }
    std::sort(v, lessThan);
}

bool lessThanInt(int lhs, int rhs) { return lhs < rhs; }
sortVectorInt(vectorOfInt, lessThanInt); // Passes the pointer to a free function

struct GreaterThanInt {
    static bool cmp(int lhs, int rhs) { return lhs > rhs; }
};
sortVectorInt(vectorOfInt, &GreaterThanInt::cmp); // Passes the pointer to a static member
function
```

- (*lessThan)(v.front(), v.back()) // All
- std::invoke(lessThan, v.front(), v.back()) // <= C++17

operatorFunctors

operator() . C ++ 11Lambdas .

```
struct Person {
    std::string name;
    unsigned int age;
};

// Functor which find a person by name
struct FindPersonByName {
    FindPersonByName(const std::string &name) : _name(name) {}
```

```

    // Overloaded method which will get called
    bool operator()(const Person &person) const {
        return person.name == _name;
    }
private:
    std::string _name;
};

std::vector<Person> v; // Assume this contains data
std::vector<Person>::iterator iFind =
    std::find_if(v.begin(), v.end(), FindPersonByName("Foobar"));
// ...

```

typedef ◦ `std::find_if`

```

template<typename Iterator, typename Predicate>
Iterator find_if(Iterator begin, Iterator end, Predicate &predicate) {
    for (Iterator i = begin, i != end, ++i)
        if (predicate(*i))
            return i;
    return end;
}

```

C ++ 17 `invoke` ◦ `std::invoke(predicate, *i)` ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/6073/>

59:

Examples

mutable **const**◦

◦

std::unique_lock **const**◦

◦

```
class pi_calculator {
public:
    double get_pi() const {
        if (pi_calculated) {
            return pi;
        } else {
            double new_pi = 0;
            for (int i = 0; i < 1000000000; ++i) {
                // some calculation to refine new_pi
            }
            // note: if pi and pi_calculated were not mutable, we would get an error from a
            compiler
            // because in a const method we can not change a non-mutable field
            pi = new_pi;
            pi_calculated = true;
            return pi;
        }
    }
private:
    mutable bool pi_calculated = false;
    mutable double pi = 0;
};
```

lambdas

lambdaoperator()const ◦ **lambda**const◦ **lambda**mutable operator()const

```
int a = 0;

auto bad_counter = [a] {
    return a++; // error: operator() is const
               // cannot modify members
};

auto good_counter = [a]() mutable {
    return a++; // OK
}

good_counter(); // 0
good_counter(); // 1
good_counter(); // 2
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2705/>

60:

-
- *opt { declaration-seq }*
- *opt { declaration-seq } / *C ++ 11 * /*
- *inline opt namespace attribute-specifier-seq identifier opt { declaration-seq } / *C ++ 17 * /*
- *enclosing-namespace-specifier :: identifier { declaration-seq } / *C ++ 17 * /*
- *namespace identifier = qualified-namespace-specifier ;*
- *using namespace nested-name-specifier opt namespace-name ;*
- *attribute-specifier-seq using namespace nested-name-specifier opt namespace-name ; / *C ++ 11 * /*

namespace

1. ◦ ◦
2. ◦
3. using *using* ◦ *using-directive* ◦

using namespace std; ◦ namespace std

```
//Really bad!
using namespace std;

//Calculates p^e and outputs it to std::cout
void pow(double p, double e) { /*...*/ }

//Calls pow
pow(5.0, 2.0); //Error! There is already a pow function in namespace std with the same
signature,
                //so the call is ambiguous
```

Examples

C ++C ++ ◦ ◦ <namespace>::<entity> ◦

```
namespace Example
{
    const int test = 5;

    const int test2 = test + 12; //Works within `Example` namespace
}

const int test3 = test + 3; //Fails; `test` not found outside of namespace.

const int test3 = Example::test + 3; //Works; fully qualified name used.
```

◦ ◦ ◦ ◦ C ++ ◦

```

namespace Electronics
{
    int TotalStock;
    class Headphones
    {
        // Description of a Headphone (color, brand, model number, etc.)
    };
    class Television
    {
        // Description of a Television (color, brand, model number, etc.)
    };
}

namespace Shoes
{
    int TotalStock;
    class Sandal
    {
        // Description of a Sandal (color, brand, model number, etc.)
    };
    class Slipper
    {
        // Description of a Slipper (color, brand, model number, etc.)
    };
}

```

::°

```

void bar() {
    // defined in global namespace
}
namespace foo {
    void bar() {
        // defined in namespace foo
    }
    void barbar() {
        bar(); // calls foo::bar()
        ::bar(); // calls bar() defined in global namespace
    }
}

```

```

//Creates namespace foo
namespace Foo
{
    //Declares function bar in namespace foo
    void bar() {}
}

```

bar ::

```

Foo::bar();

```

```

namespace A
{
    namespace B
    {
        namespace C
    }
}

```

```

    {
        void bar() {}
    }
}

```

C++ 17

```

namespace A::B::C
{
    void bar() {}
}

```

namespace°

```

namespace Foo
{
    void bar() {}
}

//some other stuff

namespace Foo
{
    void bar2() {}
}

```

“”

FooFoo:: using namespace Foo;Foo°

```

namespace Foo
{
    void bar() {}
    void baz() {}
}

//Have to use Foo::bar()
Foo::bar();

//Import Foo
using namespace Foo;
bar(); //OK
baz(); //OK

```

```

using Foo::bar;
bar(); //OK, was specifically imported
baz(); // Not OK, was not imported

```

using namespace° ° “using”°

```

/***** foo.h *****/
namespace Foo
{

```

```

class C;
}

/***** bar.h *****/
namespace Bar
{
class C;
}

/***** baz.h *****/
#include "foo.h"
using namespace Foo;

/***** main.cpp *****/
#include "bar.h"
#include "baz.h"

using namespace Bar;
C c; // error: Ambiguity between Bar::C and Foo::C

```

using-directive.

◦ “Argument Dependent Lookup”ADL

```

namespace Test
{
int call(int i);

class SomeClass {...};

int call_too(const SomeClass &data);
}

call(5); //Fails. Not a qualified function name.

Test::SomeClass data;

call_too(data); //Succeeds

```

callTest ◦ call_tooSomeClassTestADL ◦

ADL

ADL ◦

```

void foo();
namespace N {
struct X {};
void foo(X ) { std::cout << '1'; }
void qux(X ) { std::cout << '2'; }
}

struct C {
void foo() {}
void bar() {
foo(N::X{}); // error: ADL is disabled and C::foo() takes no arguments
}
}

```

```

    }
};

void bar() {
    extern void foo(); // redeclares ::foo
    foo(N::X{});      // error: ADL is disabled and ::foo() doesn't take any arguments
}

int qux;

void baz() {
    qux(N::X{}); // error: variable declaration disables ADL for "qux"
}

```

C++ 11

```
inline namespace inline namespace
```

```

namespace Outer
{
    inline namespace Inner
    {
        void foo();
    }
}

```

```

namespace Outer
{
    namespace Inner
    {
        void foo();
    }

    using Inner::foo;
}

```

```
Outer::Inner::Outer::
```

```

Outer::foo();
Outer::Inner::foo();

```

```
using namespace Inner;
```

```

#include <outer.h> // See below

class MyCustomType;
namespace Outer
{
    template <>
    void foo<MyCustomType>() { std::cout << "Specialization"; }
}

```

- Outer::foo

```
// outer.h
// include guard omitted for simplification

namespace Outer
{
    inline namespace Inner
    {
        template <typename T>
        void foo() { std::cout << "Generic"; }
    }
}
```

- using namespaceOuter::foo

```
// outer.h
// include guard omitted for simplification

namespace Outer
{
    namespace Inner
    {
        template <typename T>
        void foo() { std::cout << "Generic"; }
    }
    using namespace Inner;
    // Specialization of `Outer::foo` is not possible
    // it should be `Outer::Inner::foo`.
}
```

inline

```
namespace MyNamespace
{
    // Inline the last version
    inline namespace Version2
    {
        void foo(); // New version
        void bar();
    }

    namespace Version1 // The old one
    {
        void foo();
    }
}
```

```
MyNamespace::Version1::foo(); // old version
MyNamespace::Version2::foo(); // new version
MyNamespace::foo();           // default version : MyNamespace::Version1::foo();
```

/

o

```
namespace {
```

```
int foo = 42;
}
```

foo°

- **const**◦

```
// foo.h
namespace {
    std::string globalString;
}

// 1.cpp
#include "foo.h" //< Generates unnamed_namespace(1.cpp)::globalString ...

globalString = "Initialize";

// 2.cpp
#include "foo.h" //< Generates unnamed_namespace(2.cpp)::globalString ...

std::cout << globalString; //< Will always print the empty string
```

C++ 17

```
namespace a {
    namespace b {
        template<class T>
        struct qualifies : std::false_type {};
    }
}

namespace other {
    struct bob {};
}

namespace a::b {
    template<>
    struct qualifies<::other::bob> : std::true_type {};
}
```

namespace a::b **C++ 17** namespace a::b ◦

◦

```
namespace boost
{
    namespace multiprecision
    {
        class Number ...
    }
}

namespace Name1 = boost::multiprecision;

// Both Type declarations are equivalent
```

```
boost::multiprecision::Number X // Writing the full namespace path, longer
Name1::Number Y // using the name alias, shorter
```

```
namespace boost
{
    namespace multiprecision
    {
        class Number ...
    }
}

using namespace boost;

// Both Namespaces are equivalent
namespace Name1 = boost::multiprecision;
namespace Name2 = multiprecision;
```

```
namespace boost
{
    namespace multiprecision
    {
        class Number ...
    }
}

namespace numeric
{
    namespace multiprecision
    {
        class Number ...
    }
}

using namespace numeric;
using namespace boost;

// Not recommended as
// its not explicitly clear whether Name1 refers to
// numeric::multiprecision or boost::multiprecision
namespace Name1 = multiprecision;

// For clarity, its recommended to use absolute paths
// instead
namespace Name2 = numeric::multiprecision;
namespace Name3 = boost::multiprecision;
```

```
namespace =namespace ◦ ◦ AReallyLongName::AnotherReallyLongNamequxNN::◦
```

```
namespace AReallyLongName {
    namespace AnotherReallyLongName {
        int foo();
        int bar();
        void baz(int x, int y);
    }
}

void qux() {
    namespace N = AReallyLongName::AnotherReallyLongName;
}
```



```
N::baz(N::foo(), N::bar());  
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/495/>

61:

Singleton ◦ ◦

◦ Singleton◦

◦

◦ ◦

◦ Singleton◦

[Robert Nystrom](#)

Examples

Q & A <http://stackoverflow.com/a/1008289/3807729>

[c++Singleton](#)

◦

```
class S
{
public:
    static S& getInstance()
    {
        static S    instance; // Guaranteed to be destroyed.
                          // Instantiated on first use.
        return instance;
    }
private:
    S() {}; // Constructor? (the {} brackets) are needed here.

    // C++ 03
    // =====
    // Dont forget to declare these two. You want to make sure they
    // are unacceptable otherwise you may accidentally get copies of
    // your singleton appearing.
    S(S const&); // Don't Implement
    void operator=(S const&); // Don't implement

    // C++ 11
    // =====
    // We can use the better technique of deleting the methods
    // we don't want.
public:
    S(S const&) = delete;
    void operator=(S const&) = delete;

    // Note: Scott Meyers mentions in his Effective Modern
    // C++ book, that deleted functions should generally
```

```
        //      be public as it results in better error messages
        //      due to the compilers behavior to check accessibility
        //      before deleted status
};
```

:(
Singleton

C ++

C ++

SingletonGetInstance

C ++

C ++

```
class API
{
public:
    static API& instance();

    virtual ~API() {}

    virtual const char* func1() = 0;
    virtual void func2() = 0;

protected:
    API() {}
    API(const API&) = delete;
    API& operator=(const API&) = delete;
};

class WindowsAPI : public API
{
public:
    virtual const char* func1() override { /* Windows code */ }
    virtual void func2() override { /* Windows code */ }
};

class LinuxAPI : public API
{
public:
    virtual const char* func1() override { /* Linux code */ }
    virtual void func2() override { /* Linux code */ }
};

API& API::instance() {
#ifdef PLATFORM == WIN32
    static WindowsAPI instance;
#elif PLATFORM = LINUX
    static LinuxAPI instance;
```

```
#endif
    return instance;
}
```

API。 API。

Singeton

C ++ 11

C ++ 11。 。

```
class Foo
{
public:
    static Foo& instance()
    {
        static Foo inst;
        return inst;
    }
private:
    Foo() {}
    Foo(const Foo&) = delete;
    Foo& operator =(const Foo&) = delete;
};
```

- 。

。

std::shared_ptr

```
class Singleton
{
public:
    Singleton(Singleton const&) = delete;
    Singleton& operator=(Singleton const&) = delete;

    static std::shared_ptr<Singleton> instance()
    {
        static std::shared_ptr<Singleton> s{new Singleton};
        return s;
    }

private:
    Singleton() {}
};
```

。

<https://riptutorial.com/zh-TW/cplusplus/topic/2713/>

62:

Examples

INT

“”-32767+32767。

```
int x = 2;
int y = 3;
int z = x + y;
```

unsigned short longlong long **qv**。

truefalse 。

```
bool is_even(int x) {
    return x%2 == 0;
}
const bool b = is_even(47); // false
```

“” 。

char-127+1270255。

```
const char zero = '0';
const char one = zero + 1;
const char newline = '\n';
std::cout << one << newline; // prints 1 followed by a newline
```

char16_t

C++ 11

uint_least16_t **UTF-16**。

```
const char16_t message[] = u"你好\n"; // Chinese for "hello, world\n"
std::cout << sizeof(message)/sizeof(char16_t) << "\n"; // prints 7
```

char32_t

C++ 11

uint_least32_t **UTF-32**。

```
const char32_t full_house[] = U"𠄎𠄎𠄎"; // non-BMP characters
std::cout << sizeof(full_house)/sizeof(char32_t) << "\n"; // prints 6
```

◦ C ++◦

```
float area(float radius) {
    const float pi = 3.14159f;
    return pi*radius*radius;
}
```

◦ float ◦ longlong double◦

```
double area(double radius) {
    const double pi = 3.141592653589793;
    return pi*radius*radius;
}
```

int-2147483647+2147483647 - 2 ^ 31-1+2 ^ 31-1◦ long int ◦

```
const long approx_seconds_per_year = 60L*60L*24L*365L;
```

long double◦

```
long double area(long double radius) {
    const long double pi = 3.1415926535897932385L;
    return pi*radius*radius;
}
```

C ++ 11

longlong long longlong -9223372036854775807+9223372036854775807 - ^ 2 - 63 - 1+2 ^ 63 - 1◦

```
// support files up to 2 TiB
const long long max_file_size = 2LL << 40;
```

char-32767+32767◦ short int ◦

```
// (during the last year)
short hours_worked(short days_worked) {
    return 8*days_worked;
}
```

;void voidvoid ◦ ◦

void;int main()int main(void)◦ CC ++◦

void* "void"◦ void*CAPIqsort pthread_create ◦

void;

```
static_cast<void>(std::printf("Hello, %s!\n", name)); // discard return value
```

◦

wchar_t

◦ wchar_t UTF-16◦

ASCII 255char ◦

```
const wchar_t message_ahmaric[] = L"ሰላም ልዑል\n"; //Ahmaric for "hello, world\n"  
const wchar_t message_chinese[] = L"你好\n"; // Chinese for "hello, world\n"  
const wchar_t message_hebrew[] = L"שלום עולם\n"; //Hebrew for "hello, world\n"  
const wchar_t message_russian[] = L"Привет мир\n"; //Russian for "hello, world\n"  
const wchar_t message_tamil[] = L"ஹலோ உலகம்\n"; //Tamil for "hello, world\n"
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7839/>

63:

Examples

◦

```
class Shape {
public:
    virtual ~Shape() = default;
    virtual double get_surface() const = 0;
    virtual void describe_object() const { std::cout << "this is a shape" << std::endl; }

    double get_doubled_surface() const { return 2 * get_surface(); }
};
```

- virtual ◦ get_surface() describe_object() ◦ ◦
- get_surface() ◦ = 0 ◦ ◦
- ◦
- ◦ ◦ get_double_surface() ◦
- ◦ ◦ ◦

◦

```
class Square : public Shape {
    Point top_left;
    double side_length;
public:
    Square (const Point& top_left, double side)
        : top_left(top_left), side_length(side_length) {}

    double get_surface() override { return side_length * side_length; }
    void describe_object() override {
        std::cout << "this is a square starting at " << top_left.x << ", " << top_left.y
            << " with a length of " << side_length << std::endl;
    }
};
```

- ◦ ◦ virtual ◦ override ◦
- ◦
- ◦ ◦

```
int main() {

    Square square(Point(10.0, 0.0), 6); // we know it's a square, the compiler also
    square.describe_object();
    std::cout << "Surface: " << square.get_surface() << std::endl;
}
```



```

Circle circle(Point(0.0, 0.0), 5);

Shape *ps = nullptr; // we don't know yet the real type of the object
ps = &circle;        // it's a circle, but it could as well be a square
ps->describe_object();
std::cout << "Surface: " << ps->get_surface() << std::endl;
}

```

```

Shape *ps; // see example on defining a polymorphic class
ps = get_a_new_random_shape(); // if you don't have such a function yet, you
// could just write ps = new Square(0.0,0.0, 5);

```

ShapeSquareCircle ◦

```

std::cout << "Surface: " << ps->get_surface() << std::endl;

```

◦

◦ ◦

◦ ◦

```

class Circle: public Shape { // for Shape, see example on defining a polymorphic class
    Point center;
    double radius;
public:
    Circle (const Point& center, double radius)
        : center(center), radius(radius) {}

    double get_surface() const override { return r * r * M_PI; }

    // this is only for circles. Makes no sense for other shapes
    double get_diameter() const { return 2 * r; }
};

```

get_diameter() ◦ Shape

```

Shape* ps = get_any_shape();
ps->get_diameter(); // OUCH !!! Compilation error

```

psstatic_cast

```

std::cout << "Diameter: " << static_cast<Circle*>(ps)->get_diameter() << std::endl;

```

◦ psCircle ◦

dynamic_cast ◦

```

int main() {
    Circle circle(Point(0.0, 0.0), 10);
    Shape &shape = circle;
}

```

```

std::cout << "The shape has a surface of " << shape.get_surface() << std::endl;

//shape.get_diameter();    // OUCH !!! Compilation error

Circle *pc = dynamic_cast<Circle*>(&shape); // will be nullptr if ps wasn't a circle
if (pc)
    std::cout << "The shape is a circle of diameter " << pc->get_diameter() << std::endl;
else
    std::cout << "The shape isn't a circle !" << std::endl;
}

```

dynamic_cast ° °

/virtualprotected ° vtable ° /°

```

struct VirtualDestructor {
    virtual ~VirtualDestructor() = default;
};

struct VirtualDerived : VirtualDestructor {};

struct ProtectedDestructor {
    protected:
    ~ProtectedDestructor() = default;
};

struct ProtectedDerived : ProtectedDestructor {
    ~ProtectedDerived() = default;
};

// ...

VirtualDestructor* vd = new VirtualDerived;
delete vd; // Looks up VirtualDestructor::~~VirtualDestructor() in vtable, sees it's
           // VirtualDerived::~~VirtualDerived(), calls that.

ProtectedDestructor* pd = new ProtectedDerived;
delete pd; // Error: ProtectedDestructor::~~ProtectedDestructor() is protected.
delete static_cast<ProtectedDerived*>(pd); // Good.

```

°

<https://riptutorial.com/zh-TW/cplusplus/topic/1717/>

64: CRTP

◦ CRTPC ++ ◦

Examples

CRTP

CRTP◦ ◦ static_castthis◦

CRTP◦ ◦

C ++ 14

begin()end()◦ ◦ CRTPbegin()end()

```
#include <iterator>
template <typename Sub>
class Container {
private:
    // self() yields a reference to the derived type
    Sub& self() { return *static_cast<Sub*>(this); }
    Sub const& self() const { return *static_cast<Sub const*>(this); }

public:
    decltype(auto) front() {
        return *self().begin();
    }

    decltype(auto) back() {
        return *std::prev(self().end());
    }

    decltype(auto) size() const {
        return std::distance(self().begin(), self().end());
    }

    decltype(auto) operator[](std::size_t i) {
        return *std::next(self().begin(), i);
    }
};
```

begin()end()front() back() size()operator[]◦

```
#include <memory>
// A dynamically allocated array
template <typename T>
class DynArray : public Container<DynArray<T>> {
public:
    using Base = Container<DynArray<T>>;

    DynArray(std::size_t size)
        : size_{size},
```

```

    data_{std::make_unique<T[]>(size_)}
{ }

T* begin() { return data_.get(); }
const T* begin() const { return data_.get(); }
T* end() { return data_.get() + size_; }
const T* end() const { return data_.get() + size_; }

private:
    std::size_t size_;
    std::unique_ptr<T[]> data_;
};

```

DynArrayCRTP

```

DynArray<int> arr(10);
arr.front() = 2;
arr[2] = 5;
assert(arr.size() == 10);

```

```

DynArray<int> arr(10);
DynArray<int>::Base & base = arr;
base.begin(); // no virtual calls

```

begin() Container<DynArray<int>>Container<DynArray<int>>◦

DynArray

```

class A {};
class B: public A{};

A* a = new B;

```

CRTP

CRTP

```

struct IShape
{
    virtual ~IShape() = default;

    virtual void accept(IShapeVisitor&) const = 0;
};

struct Circle : IShape
{
    // ...
    // Each shape has to implement this method the same way
    void accept(IShapeVisitor& visitor) const override { visitor.visit(*this); }
    // ...
};

struct Square : IShape
{
    // ...
};

```

```

// Each shape has to implement this method the same way
void accept(IShapeVisitor& visitor) const override { visitor.visit(*this); }
// ...
};

```

IShapeIShape° °

```

template <class Derived>
struct IShapeAcceptor : IShape {
    void accept(IShapeVisitor& visitor) const override {
        // visit with our exact type
        visitor.visit(*static_cast<Derived const*>(this));
    }
};

```

```

struct Circle : IShapeAcceptor<Circle>
{
    Circle(const Point& center, double radius) : center(center), radius(radius) {}
    Point center;
    double radius;
};

struct Square : IShapeAcceptor<Square>
{
    Square(const Point& topLeft, double sideLength) : topLeft(topLeft), sideLength(sideLength)
{}
    Point topLeft;
    double sideLength;
};

```

°

CRTIP <https://riptutorial.com/zh-TW/cplusplus/topic/9269/-crtp->

65:

◦ 42x◦

C++ 11 ◦

Examples

bool◦

```
bool ok = true;
if (!f()) {
    ok = false;
    goto end;
}
```

bool◦

```
bool ok = true;
if (!f()) {
    ok = false;
    goto end;
}
```

nullptr

C++ 11

◦ ◦

```
Widget* p = new Widget();
delete p;
p = nullptr; // set the pointer to null after deletion
```

nullptr◦ nullptrstd::nullptr_t◦

```
void f(int* p);

template <class T>
void g(T* p);

void h(std::nullptr_t p);

int main() {
    f(nullptr); // ok
    g(nullptr); // error
    h(nullptr); // ok
}
```

this

◦ `this`

```
struct S {
    int x;
    S& operator=(const S& other) {
        x = other.x;
        // return a reference to the object being assigned to
        return *this;
    }
};
```

`this` `CV-X::` `const` `this` `const X*` `this` `const` ◦ `this` `volatile` ◦

C++ 11

`this` ◦

```
struct S;
struct T {
    T(const S* s);
    // ...
};
struct S {
    // ...
    T t{this};
};
```

`this` ◦

•

`1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9`

```
int d = 42;
```

•

`00,1,2,3,4,5,6,7`

```
int o = 052
```

•

`0x0X0,1,2,3,4,5,6,7,8,9aAbBcC dDeEfF`

```
int x = 0x2a; int X = 0X2A;
```

• `binary-literal` C++ 14

`0b0B0,1`

```
int b = 0b101010; // C++14
```

• `unsigned-suffix` `u`

```
unsigned int u_1 = 42u;
```

- long-suffixllong-long-suffixllllc ++ 11

```
unsigned long long l1 = 18446744073709550592ull; // C++11
unsigned long long l2 = 18'446'744'073'709'550'592llu; // C++14
unsigned long long l3 = 1844'6744'0737'0955'0592uLL; // C++14
unsigned long long l4 = 184467'440737'0'95505'92LLU; // C++14
```

0xDeAdBaBeU0XdeadBABEulong-long-suffixlllllll

◦ -1◦

C99CC ++long intunsigned long int◦

#if#elifstd :: intmax_tstd :: uintmax_t◦

<https://riptutorial.com/zh-TW/cplusplus/topic/7836/>

66:

◦ ◦

auto **C++ 11** register **C++ 17** static [thread_local](#) **C++ 11** externmutable ◦

decl-specifier-seq thread_local static extern ◦

◦ ◦ ◦

Examples

◦ const const ◦

```
class C {
    int x;
    mutable int times_accessed;
public:
    C(): x(0), times_accessed(0) {
    }
    int get_x() const {
        ++times_accessed; // ok: const member function can modify mutable data member
        return x;
    }
    void set_x(int x) {
        ++times_accessed;
        this->x = x;
    }
};
```

C++ 11

C++ 11 mutable ◦ [lambda](#) [lambda](#) const ◦ [lambda](#) copy ◦ [mutable lambdas](#) ◦

```
std::vector<int> my_iota(int start, int count) {
    std::vector<int> result(count);
    std::generate(result.begin(), result.end(),
        [start]() mutable { return start++; });
    return result;
}
```

lambda mutable ◦

C++ 17

◦ “CPU” **C++ 11** ◦

```
register int i = 0;
while (i < 100) {
    f(i);
    int g = i*i;
```

```
i += h(i, g);
}
```

register ◦ **CC ++** register ◦ register ◦

C ++ 17

register ◦ register ◦

static ◦

1. ◦

```
// internal function; can't be linked to
static double semiperimeter(double a, double b, double c) {
    return (a + b + c)/2.0;
}
// exported to client
double area(double a, double b, double c) {
    const double s = semiperimeter(a, b, c);
    return sqrt(s*(s-a)*(s-b)*(s-c));
}
```

2. thread_local ◦ ◦ ◦

```
void f() {
    static int count = 0;
    std::cout << "f has been called " << ++count << " times so far\n";
}
```

3. ◦

```
struct S {
    static S* create() {
        return new S;
    }
};
int main() {
    S* s = S::create();
}
```

23 static ◦

C ++ 03

◦ ◦

```
void f() {
    auto int x; // equivalent to: int x;
    auto y;    // illegal in C++; legal in C89
}
auto int z;   // illegal: namespace-scope variable cannot be automatic
```

EXTERN

extern

1. ◦ ◦

```
// global scope
int x;           // definition; x will be default-initialized
extern int y;    // declaration; y is defined elsewhere, most likely another TU
extern int z = 42; // definition; "extern" has no effect here (compiler may warn)
```

2. constconstexpr◦

```
// global scope
const int w = 42;           // internal linkage in C++; external linkage in C
static const int x = 42;    // internal linkage in both C++ and C
extern const int y = 42;    // external linkage in both C++ and C
namespace {
    extern const int z = 42; // however, this has internal linkage since
                             // it's in an unnamed namespace
}
```

3. ◦ ◦

```
// global scope
namespace {
    int x = 1;
    struct C {
        int x = 2;
        void f() {
            extern int x;           // redeclares namespace-scope x
            std::cout << x << '\n'; // therefore, this prints 1, not 2
        }
    };
}
void g() {
    extern int y; // y has external linkage; refers to global y defined elsewhere
}
```

extern ◦ ◦

```
void f();           // typically a forward declaration; f defined later in this TU
extern void g();    // typically not a forward declaration; g defined in another TU
```

fexterngextern ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/9225/>

67:

ref. ◦

```
template<class T>
void f(T&& x) // x is a forwarding reference, because T is deduced from a call to f()
{
    g(std::forward<T>(x)); // g() will receive an lvalue or an rvalue, depending on x
}
```

T

```
template<class T>
struct a
{
    a(T&& x); // x is a rvalue reference, not a forwarding reference
};
```

C++ 17

C++ 17. “a”

```
a example1(1);
// same as a<int> example1(1);

int x = 1;
a example2(x);
// same as a<int&> example2(x);
```

Examples

◦ `make_unique` ◦ `Tunique_ptr<T>` ◦

◦

```
template<class T, class... A>
unique_ptr<T> make_unique(A&&... args)
{
    return unique_ptr<T>(new T(std::forward<A>(args)...));
}
```

...◦ ◦ `std::forwardT` ◦ **ref**◦

```
struct foo
{
    foo() {}
    foo(const foo&) {} // copy constructor
    foo(foo&&) {} // copy constructor
    foo(int, int, int) {}
};
```

```
foo f;
auto p1 = make_unique<foo>(f);           // calls foo::foo(const foo&)
auto p2 = make_unique<foo>(std::move(f)); // calls foo::foo(foo&&)
auto p3 = make_unique<foo>(1, 2, 3);
```

<https://riptutorial.com/zh-TW/cplusplus/topic/1750/>

68:

Examples

TCP

Beej: TCP“Hello World”。 。 。

。 -

```
#include <cstring>    // sizeof()
#include <iostream>
#include <string>

// headers for socket(), getaddrinfo() and friends
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>

#include <unistd.h>    // close()

int main(int argc, char *argv[])
{
    // Let's check if port number is supplied or not..
    if (argc != 2) {
        std::cerr << "Run program as 'program <port>'\n";
        return -1;
    }

    auto &portNum = argv[1];
    const unsigned int backlog = 8; // number of connections allowed on the incoming queue

    addrinfo hints, *res, *p;    // we need 2 pointers, res to hold and p to iterate over
    memset(&hints, 0, sizeof(hints));

    // for more explanation, man socket
    hints.ai_family    = AF_UNSPEC;    // don't specify which IP version to use yet
    hints.ai_socktype = SOCK_STREAM;  // SOCK_STREAM refers to TCP, SOCK_DGRAM will be?
    hints.ai_flags     = AI_PASSIVE;

    // man getaddrinfo
    int gAddRes = getaddrinfo(NULL, portNum, &hints, &res);
    if (gAddRes != 0) {
        std::cerr << gai_strerror(gAddRes) << "\n";
        return -2;
    }

    std::cout << "Detecting addresses" << std::endl;

    unsigned int numOfAddr = 0;
    char ipStr[INET6_ADDRSTRLEN]; // ipv6 length makes sure both ipv4/6 addresses can be
    stored in this variable
```

```

// Now since getaddrinfo() has given us a list of addresses
// we're going to iterate over them and ask user to choose one
// address for program to bind to
for (p = res; p != NULL; p = p->ai_next) {
    void *addr;
    std::string ipVer;

    // if address is ipv4 address
    if (p->ai_family == AF_INET) {
        ipVer          = "IPv4";
        sockaddr_in *ipv4 = reinterpret_cast<sockaddr_in *>(p->ai_addr);
        addr           = &(ipv4->sin_addr);
        ++numOfAddr;
    }

    // if address is ipv6 address
    else {
        ipVer          = "IPv6";
        sockaddr_in6 *ipv6 = reinterpret_cast<sockaddr_in6 *>(p->ai_addr);
        addr           = &(ipv6->sin6_addr);
        ++numOfAddr;
    }

    // convert IPv4 and IPv6 addresses from binary to text form
    inet_ntop(p->ai_family, addr, ipStr, sizeof(ipStr));
    std::cout << "(" << numOfAddr << ") " << ipVer << " : " << ipStr
        << std::endl;
}

// if no addresses found :(
if (!numOfAddr) {
    std::cerr << "Found no host address to use\n";
    return -3;
}

// ask user to choose an address
std::cout << "Enter the number of host address to bind with: ";
unsigned int choice = 0;
bool madeChoice     = false;
do {
    std::cin >> choice;
    if (choice > (numOfAddr + 1) || choice < 1) {
        madeChoice = false;
        std::cout << "Wrong choice, try again!" << std::endl;
    } else
        madeChoice = true;
} while (!madeChoice);

p = res;

// let's create a new socket, socketFD is returned as descriptor
// man socket for more information
// these calls usually return -1 as result of some error
int sockFD = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
if (sockFD == -1) {
    std::cerr << "Error while creating socket\n";
    freeaddrinfo(res);
}

```

```

    return -4;
}

// Let's bind address to our socket we've just created
int bindR = bind(sockFD, p->ai_addr, p->ai_addrlen);
if (bindR == -1) {
    std::cerr << "Error while binding socket\n";

    // if some error occurs, make sure to close socket and free resources
    close(sockFD);
    freeaddrinfo(res);
    return -5;
}

// finally start listening for connections on our socket
int listenR = listen(sockFD, backlog);
if (listenR == -1) {
    std::cerr << "Error while Listening on socket\n";

    // if some error occurs, make sure to close socket and free resources
    close(sockFD);
    freeaddrinfo(res);
    return -6;
}

// structure large enough to hold client's address
sockaddr_storage client_addr;
socklen_t client_addr_size = sizeof(client_addr);

const std::string response = "Hello World";

// a fresh infinite loop to communicate with incoming connections
// this will take client connections one at a time
// in further examples, we're going to use fork() call for each client connection
while (1) {

    // accept call will give us a new socket descriptor
    int newFD
        = accept(sockFD, (sockaddr *) &client_addr, &client_addr_size);
    if (newFD == -1) {
        std::cerr << "Error while Accepting on socket\n";
        continue;
    }

    // send call sends the data you specify as second param and it's length as 3rd param,
    // also returns how many bytes were actually sent
    auto bytes_sent = send(newFD, response.data(), response.length(), 0);
    close(newFD);
}

close(sockFD);
freeaddrinfo(res);

return 0;
}

```



```
-  
Detecting addresses  
(1) IPv4 : 0.0.0.0  
(2) IPv6 : ::  
Enter the number of host address to bind with: 1
```

TCP

Hello TCP Server。 Hello TCP。

```
-
```

```
#include <cstring>  
#include <iostream>  
#include <string>  
  
#include <arpa/inet.h>  
#include <netdb.h>  
#include <sys/socket.h>  
#include <sys/types.h>  
#include <unistd.h>  
  
int main(int argc, char *argv[])  
{  
    // Now we're taking an ipaddress and a port number as arguments to our program  
    if (argc != 3) {  
        std::cerr << "Run program as 'program <ipaddress> <port>'\n";  
        return -1;  
    }  
  
    auto &ipAddress = argv[1];  
    auto &portNum = argv[2];  
  
    addrinfo hints, *p;  
    memset(&hints, 0, sizeof(hints));  
    hints.ai_family = AF_UNSPEC;  
    hints.ai_socktype = SOCK_STREAM;  
    hints.ai_flags = AI_PASSIVE;  
  
    int gAddRes = getaddrinfo(ipAddress, portNum, &hints, &p);  
    if (gAddRes != 0) {  
        std::cerr << gai_strerror(gAddRes) << "\n";  
        return -2;  
    }  
  
    if (p == NULL) {  
        std::cerr << "No addresses found\n";  
        return -3;  
    }  
  
    // socket() call creates a new socket and returns it's descriptor  
    int sockFD = socket(p->ai_family, p->ai_socktype, p->ai_protocol);  
    if (sockFD == -1) {  
        std::cerr << "Error while creating socket\n";  
        return -4;  
    }  
}
```

```

}

// Note: there is no bind() call as there was in Hello TCP Server
// why? well you could call it though it's not necessary
// because client doesn't necessarily has to have a fixed port number
// so next call will bind it to a random available port number

// connect() call tries to establish a TCP connection to the specified server
int connectR = connect(sockFD, p->ai_addr, p->ai_addrlen);
if (connectR == -1) {
    close(sockFD);
    std::cerr << "Error while connecting socket\n";
    return -5;
}

std::string reply(15, ' ');

// recv() call tries to get the response from server
// BUT there's a catch here, the response might take multiple calls
// to recv() before it is completely received
// will be demonstrated in another example to keep this minimal
auto bytes_rcv = recv(sockFD, &reply.front(), reply.size(), 0);
if (bytes_rcv == -1) {
    std::cerr << "Error while receiving bytes\n";
    return -6;
}

std::cout << "\nClient recieved: " << reply << std::endl;
close(sockFD);
freeaddrinfo(p);

return 0;
}

```

<https://riptutorial.com/zh-TW/cplusplus/topic/7177/>

69:

Examples

Char

char ◦ ◦

- char
-
-
- char16_t char32_t
- bool
- wchar_t

sizeof(char) / sizeof(signed char) / sizeof(unsigned char) §3.9.1.1[basic.fundamental / 1]
 §5.3.3.1[expr.sizeof]sizeof(bool) 3.9.1[basic.fundamental]◦

char

C ++5.3.3.1sizeofunsigned char signed charchar1 charsignedunsigned ◦

C ++ 14

char256UTF-8◦

3.9.1.2signed char short int int long intlong long int ◦ ◦ 3.9.1.3 unsigned char unsigned short
 int unsigned int unsigned long intunsigned long long int ◦ ◦ 3.9.1.1 charsigned charunsigned
 char◦

C ++ 11

C ++ 11 long longunsigned long longC ++◦ C99Clong long unsigned long long C◦

```
1 == sizeof(char) == sizeof(signed char) == sizeof(unsigned char)
<= sizeof(short) == sizeof(unsigned short)
<= sizeof(int) == sizeof(unsigned int)
<= sizeof(long) == sizeof(unsigned long)
```

C ++ 11

```
<= sizeof(long long) == sizeof(unsigned long long)
```

◦ 3.9.1.3C5.2.4.2.1◦ ;◦ ;◦

signed char	-127127 - 2 ⁷ - 12 ⁷ - 1	8
-------------	--	---

unsigned char	$2^{502} - 1$	8
signed short	$-2^{15} - 2^{15} - 1$	16
unsigned short	$2^{16} - 1$	16
signed int	$-2^{15} - 2^{15} - 1$	16
unsigned int	$2^{16} - 1$	16
signed long	$-2^{31} - 2^{31} - 1$	32
unsigned long	$2^{32} - 1$	32

C ++ 11

signed long long	$-2^{63} - 2^{63} - 1$	64
unsigned long long	$2^{64} - 1$	64

◦ 64LP64LLP64LLP6464Windows32intslong ints LP6464Linux32int64long s◦ ◦

C ++ 11

<stdint>int8_t int16_t int32_t int64_t intptr_t uint8_t uint16_t uint32_t uint64_tuintptr_t◦

C ++ 11

char16_tchar32_t

char16_tchar32_t5.3.3.13.9.1.5

- char16_tUTF-16uint_least16_tuint_least16_t ;16◦
- char32_tUTF-32uint_least32_tuint_least32_t ;32◦

bool

bool1◦

wchar_t

wchar_t 3.9.1.5◦ ◦ 5.3.3.18,1632;Unicodewchar_t32Windowswchar_t16◦ C90ISO 98991990§4.1.5

wchar_t8,1632◦

- Unix `wchar_t` 32 UTF-32
- Windows `wchar_t` 16 UTF-16
- 8 `wchar_t` 8

C++ 11

Unicode `char` UTF-8 `char16_t` UTF-16 `char32_t` UTF-32 `wchar_t`

	<code>int</code>	<code>long</code>	
LP32	16	32	32
ILP32	32	32	32
LLP64	32	32	64
LP64	32	64	64

- 16 Windows LP32
- 32* nix Unix Linux Mac OSX Unix Windows ILP32
- 64 Windows LLP64
- 64* nix LP64

C++ `char` `CHAR_BIT` limits 8.8 POSIX `CHAR_BIT` 8 `CHAR_BIT` 8, 16, 32, 64

`reinterpret_cast` "....."

```
int x = 42;
int* p = &x;
long addr = reinterpret_cast<long>(p);
std::cout << addr << "\n"; // prints some numeric address,
                          // probably in the architecture's native address format
```

`uintptr_t` `intptr_t`

```
// `uintptr_t` was not in C++03. It's in C99, in <stdint.h>, as an optional type
#include <stdint.h>

uintptr_t uip;
```

C++ 11

```
// There is an optional `std::uintptr_t` in C++11
#include <cstdint>

std::uintptr_t uip;
```

uintptr_t C99.3.2.3C ++ 11C99

voidvoid

uintptr_tchar *void * uintptr_tuintptr_t ° uintptr_tvoid *

- XSIX / Open System Interfacesuintptr_tuintptr_t °
- C; Cuintptr_t° POSIX2.12.3

void ° void * ° void *

- C99§7.18.1

typedefu6.2.5;°

uintptr_t°

◦ <limits>std::numeric_limits<T>° C<climits>> = C ++ 11 <stdint>°

- std::numeric_limits<signed char>::min() SCHAR_MIN -127°
- std::numeric_limits<signed char>::max() SCHAR_MAX 127°
- std::numeric_limits<unsigned char>::max() UCHAR_MAX 255°
- std::numeric_limits<short>::min() SHRT_MIN -32767°
- std::numeric_limits<short>::max() SHRT_MAX 32767°
- std::numeric_limits<unsigned short>::max() USHRT_MAX 65535°
- std::numeric_limits<int>::min() INT_MIN -32767°
- std::numeric_limits<int>::max() INT_MAX 32767°
- std::numeric_limits<unsigned int>::max() UINT_MAX 65535°
- std::numeric_limits<long>::min() LONG_MIN -2147483647°
- std::numeric_limits<long>::max() LONG_MAX 2147483647°
- std::numeric_limits<unsigned long>::max() ULONG_MAX 4294967295°

C ++ 11

- std::numeric_limits<long long>::min() LLONG_MIN -9223372036854775807°
- std::numeric_limits<long long>::max() LLONG_MAX 9223372036854775807°
- std::numeric_limits<unsigned long long>::max() ULLONG_MAX 18446744073709551615°

T max()min()° C<float>°

- digits10°
 - std::numeric_limits<float>::digits10FLT_DIG 6°

- `std::numeric_limits<double>::digits10` DBL_DIG **10**◦
- `std::numeric_limits<long double>::digits10` LDBL_DIG **10**◦
- `min_exponent10` EE10◦
 - `std::numeric_limits<float>::min_exponent10` FLT_MIN_10_EXP **-37**◦
 - `std::numeric_limits<double>::min_exponent10` DBL_MIN_10_EXP **-37**◦
 - `std::numeric_limits<long double>::min_exponent10` LDBL_MIN_10_EXP **-37**◦
- `max_exponent10` EE10◦
 - `std::numeric_limits<float>::max_exponent10` FLT_MAX_10_EXP **37**◦
 - `std::numeric_limits<double>::max_exponent10` DBL_MAX_10_EXP **37**◦
 - `std::numeric_limits<long double>::max_exponent10` LDBL_MAX_10_EXP **37**◦
- `is_iec559` IEC 559 / IEEE 754◦

`long double double float ;long double double double float◦ ◦`

`T std::numeric_limits<T>::radix` T◦

`std::numeric_limits<T>::is_iec559` T IEC 559 / IEEE 754◦

◦

```
// Suppose that on this implementation, the range of signed char is -128 to +127 and
// the range of unsigned char is 0 to 255
int x = 12345;
signed char sc = x; // sc has an implementation-defined value
unsigned char uc = x; // uc is initialized to 57 (i.e., 12345 modulo 256)
```

◦

```
enum E {
    RED,
    GREEN,
    BLUE,
};
using T = std::underlying_type<E>::type; // implementation-defined
```

`int int unsigned int◦ T int unsigned int short long long◦`

`sizeof◦`

<https://riptutorial.com/zh-TW/cplusplus/topic/1363/>

70:

Examples

“class” class structenum structenum class ◦

- ;◦ pp + 1◦ 0◦ ◦

```
class Empty_1 {}; // sizeof(Empty_1) == 1
class Empty_2 {}; // sizeof(Empty_2) == 1
class Derived : Empty_1 {}; // sizeof(Derived) == 1
class DoubleDerived : Empty_1, Empty_2 {}; // sizeof(DoubleDerived) == 1
class Holder { Empty_1 e; }; // sizeof(Holder) == 1
class DoubleHolder { Empty_1 e1; Empty_2 e2; }; // sizeof(DoubleHolder) == 2
class DerivedHolder : Empty_1 { Empty_1 e; }; // sizeof(DerivedHolder) == 2
```

-

```
struct S {
    int x;
    char* y;
};
```

Ssizeof(int)xsizeof(char*)y◦ ◦

- t1, t2, ... tN/sizeof(t1) + sizeof(t2) + ... + sizeof(tN)◦ ◦

```
struct AnInt { int i; };
// sizeof(AnInt) == sizeof(int)
// Assuming a typical 32- or 64-bit system, sizeof(AnInt) == 4 (4).
struct TwoInts { int i, j; };
// sizeof(TwoInts) >= 2 * sizeof(int)
// Assuming a typical 32- or 64-bit system, sizeof(TwoInts) == 8 (4 + 4).
struct IntAndChar { int i; char c; };
// sizeof(IntAndChar) >= sizeof(int) + sizeof(char)
// Assuming a typical 32- or 64-bit system, sizeof(IntAndChar) == 8 (4 + 1 + padding).
struct AnIntDerived : AnInt { long long l; };
// sizeof(AnIntDerived) >= sizeof(AnInt) + sizeof(long long)
// Assuming a typical 32- or 64-bit system, sizeof(AnIntDerived) == 16 (4 + padding + 8).
```

- ◦ memNalignof(memN) nalignofalignof ◦ alignofalignof ◦ ◦ alignofalignof “”◦ alignofalignof alignof◦

```
// Assume sizeof(short) == 2, sizeof(int) == 4, and sizeof(long long) == 8.
// Assume 4-byte alignment is specified to the compiler.
struct Char { char c; };
```



```

    // sizeof(Char)                == 1 (sizeof(char))
struct Int { int i; };
    // sizeof(Int)                 == 4 (sizeof(int))
struct CharInt { char c; int i; };
    // sizeof(CharInt)             == 8 (1 (char) + 3 (padding) + 4 (int))
struct ShortIntCharInt { short s; int i; char c; int j; };
    // sizeof(ShortIntCharInt)    == 16 (2 (short) + 2 (padding) + 4 (int) + 1 (char) +
    //                               3 (padding) + 4 (int))
struct ShortIntCharCharInt { short s; int i; char c; char d; int j; };
    // sizeof(ShortIntCharCharInt) == 16 (2 (short) + 2 (padding) + 4 (int) + 1 (char) +
    //                               1 (char) + 2 (padding) + 4 (int))
struct ShortCharShortInt { short s; char c; short t; int i; };
    // sizeof(ShortCharShortInt)  == 12 (2 (short) + 1 (char) + 1 (padding) + 2 (short) +
    //                               2 (padding) + 4 (int))
struct IntLLInt { int i; long long l; int j; };
    // sizeof(IntLLInt)           == 16 (4 (int) + 8 (long long) + 4 (int))
    // If packing isn't explicitly specified, most compilers will pack this as
    // 8-byte alignment, such that:
    // sizeof(IntLLInt)           == 24 (4 (int) + 4 (padding) + 8 (long long) +
    //                               4 (int) + 4 (padding))

// Assume sizeof(bool) == 1, sizeof(ShortIntCharInt) == 16, and sizeof(IntLLInt) == 24.
// Assume default alignment: alignof(ShortIntCharInt) == 4, alignof(IntLLInt) == 8.
struct ShortChar3ArrShortInt {
    short s;
    char c3[3];
    short t;
    int i;
};
// ShortChar3ArrShortInt has 4-byte alignment: alignof(int) >= alignof(char) &&
//                                           alignof(int) >= alignof(short)
// sizeof(ShortChar3ArrShortInt) == 12 (2 (short) + 3 (char[3]) + 1 (padding) +
//                                     2 (short) + 4 (int))
// Note that t is placed at alignment of 2, not 4. alignof(short) == 2.

struct Large_1 {
    ShortIntCharInt sici;
    bool b;
    ShortIntCharInt tjdj;
};
// Large_1 has 4-byte alignment.
// alignof(ShortIntCharInt) == alignof(int) == 4
// alignof(b) == 1
// Therefore, alignof(Large_1) == 4.
// sizeof(Large_1) == 36 (16 (ShortIntCharInt) + 1 (bool) + 3 (padding) +
//                       16 (ShortIntCharInt))
struct Large_2 {
    IntLLInt illi;
    float f;
    IntLLInt jmmj;
};
// Large_2 has 8-byte alignment.
// alignof(IntLLInt) == alignof(long long) == 8
// alignof(float) == 4
// Therefore, alignof(Large_2) == 8.
// sizeof(Large_2) == 56 (24 (IntLLInt) + 4 (float) + 4 (padding) + 24 (IntLLInt))

```

C++ 11

- alignas^o Chars<5>**8.4**^o4^o.

```

// This type shall always be aligned to a multiple of 4. Padding shall be inserted as
// needed.
// Chars<1>..Chars<4> are 4 bytes, Chars<5>..Chars<8> are 8 bytes, etc.
template<size_t SZ>
struct alignas(4) Chars { char arr[SZ]; };

static_assert(sizeof(Chars<1>) == sizeof(Chars<4>), "Alignment is strict.\n");

```

- ◦ ◦
- ◦

unsigned char ◦ unsigned char **8** char **256** {0,1...255} ◦ **42** 00101010 ◦

signed char ◦ signed char **88** ◦

◦

◦ unsigned int **6402** $2^{32} - 1$ ◦ **32** ◦

◦ ◦ unsigned short 00010100101010115291 short 5291 ◦

◦

◦ float double **IEEE 754** 3264 float **2381** ◦ ◦ "" ◦

◦ TT ◦

◦

```
int a[5][3];
```

5 int **5** ◦ a[0]a[0][0] a[0][1] a[0][2] a[1]a[1]a[1][0] a[1][1]a[1][2] ◦ ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/9329/>

71:

C++

21.

- 16,124,816,321,482,160
-
- void char
- CVCV

C++ 03 C++ 11 alignof alignas

Examples

C++ 11

alignof std::size_t

```
#include <iostream>
int main() {
    std::cout << "The alignment requirement of int is: " << alignof(int) << '\n';
}
```

int4

...

C++ 11

alignas

- alignas(x) xx
- alignas(T) T Talignof(T)

alignas

buf int unsigned char

```
alignas(int) unsigned char buf[sizeof(int)];
new (buf) int(42);
```

alignas

```
alignas(1) int i; //11-formed, unless `int` on this platform is aligned to 1 byte.
alignas(char) int j; //11-formed, unless `int` has the same or smaller alignment than `char`.
```

alignas ◦ 2 ◦ std::max_align_t std::max_align_t ◦ ◦ ◦

C ++ 17 operator new ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/9249/>

72:

- `[[details]]`
- `[[detailsarguments]]`
- `__attributeGCC / Clang / IBM`
- `__declspecMSVC`

Examples

`[[[]]]`

C++ 11

C++ 11 `[[noreturn]]` `return` `void` `std::terminate` `std::exit` `longjmp`

`std::terminate` `[[noreturn]]`

```
[[noreturn]] void ownAssertFailureHandler(std::string message) {
    std::cerr << message << std::endl;
    if (THROW_EXCEPTION_ON_ASSERT)
        throw AssertException(std::move(message));
    std::terminate();
}
```

`return` `ownAssertFailureHandler`

```
std::vector<int> createSequence(int end) {
    if (end > 0) {
        std::vector<int> sequence;
        sequence.reserve(end+1);
        for (int i = 0; i <= end; ++i)
            sequence.push_back(i);
        return sequence;
    }
    ownAssertFailureHandler("Negative number passed to createSequence()"s);
    // return std::vector<int>{}; //< Not needed because of [[noreturn]]
}
```

```
[[noreturn]] void assertPositive(int number) {
    if (number >= 0)
        return;
    else
        ownAssertFailureHandler("Positive number expected"s); //< [[noreturn]]
}
```

`[[noreturn]]` `void`

```

template<class InconsistencyHandler>
double fortyTwoDivideBy(int i) {
    if (i == 0)
        i = InconsistencyHandler::correct(i);
    return 42. / i;
}

struct InconsistencyThrower {
    static [[noreturn]] int correct(int i) { ownAssertFailureHandler("Unknown
inconsistency"s); }
}

struct InconsistencyChangeToOne {
    static int correct(int i) { return 1; }
}

double fortyTwo = fortyTwoDivideBy<InconsistencyChangeToOne>(0);
double unreachable = fortyTwoDivideBy<InconsistencyThrower>(0);

```

- `std::`
- `std::`
- `std::quick_exit`
- `std::`
- `std::`
- `std::rethrow_exception`
- `std::throw_with_nested`
- `std::nested_exception::rethrow_nested`

[[[]]]

C++ 17

caseswitch◦ `'break'`◦ ◦

C++ 17◦ `break[[fallthrough]] [[fallthrough]]`◦

```

switch(input) {
    case 2011:
    case 2014:
    case 2017:
        std::cout << "Using modern C++" << std::endl;
        [[fallthrough]]; // > No warning
    case 1998:
    case 2003:
        standard = input;
}

```

[[fallthrough]]◦

[[deprecated]][[deprecated“reason”]]

C++ 14

C++ 14 `[[deprecated]]` `[[deprecated("reason")]]`

```
void function(std::unique_ptr<A> &&a);

// Provides specific message which helps other programmers fixing there code
[[deprecated("Use the variant with unique_ptr instead, this function will be removed in the
next release")]]
void function(std::auto_ptr<A> a);

// No message, will result in generic warning if called.
[[deprecated]]
void function(A *a);
```

-
- **typedef**
-
-
-
-
-

c++ 14 7.6.5

`[[nodiscard]]`

C++ 17

`[[nodiscard]]` ◦ ◦

-
-

◦

```
template<typename Function>
[[nodiscard]] Finally<std::decay_t<Function>> onExit(Function &&f);

void f(int &i) {
    assert(i == 0); // Just to make comments clear!
    ++i;           // i == 1
    auto exit1 = onExit([&i]{ --i; }); // Reduce by 1 on exiting f()
    ++i;           // i == 2
    onExit([&i]{ --i; }); // BUG: Reducing by 1 directly
                                // Compiler warning expected
    std::cout << i << std::endl; // Expected: 2, Real: 1
}
```

`[[nodiscard]]` ◦

`onExitFinally` / `onExit` [Finally](#) / [ScopeExit](#) ◦

`[[maybe_unused]]`

[[maybe_unused]] ° ° °

° °

```
[[maybe_unused]] auto mapInsertResult = configuration.emplace("LicenseInfo",
stringifiedLicenseInfo);
assert(mapInsertResult.second); // We only get called during startup, so we can't be in the
map
```

° ° [[maybe_unused]] °

```
namespace {
    [[maybe_unused]] std::string createWindowsConfigFilePath(const std::string &relativePath);
    // TODO: Reuse this on BSD, MAC ...
    [[maybe_unused]] std::string createLinuxConfigFilePath(const std::string &relativePath);
}

std::string createConfigFilePath(const std::string &relativePath) {
    #if OS == "WINDOWS"
        return createWindowsConfigFilePath(relativePath);
    #elif OS == "LINUX"
        return createLinuxConfigFilePath(relativePath);
    #else
        #error "OS is not yet supported"
    #endif
}
```

[[maybe_unused]] °

<https://riptutorial.com/zh-TW/cplusplus/topic/5251/>

73:

- C++ union. C++.

`std::variant` C++ 17 union.

- .

Examples

◦

```
union U {
    int a;
    short b;
    float c;
};
U u;

//Address of a and b will be equal
(void*)&u.a == (void*)&u.b;
(void*)&u.a == (void*)&u.c;

//Assigning to any union member changes the shared memory of all members
u.c = 4.f;
u.a = 5;
u.c != 4.f;
```

◦

```
struct AnyType {
    enum {
        IS_INT,
        IS_FLOAT
    } type;

    union Data {
        int as_int;
        float as_float;
    } value;

    AnyType(int i) : type(IS_INT) { value.as_int = i; }
    AnyType(float f) : type(IS_FLOAT) { value.as_float = f; }

    int get_int() const {
        if(type == IS_INT)
            return value.as_int;
        else
            return (int)value.as_float;
    }

    float get_float() const {
        if(type == IS_FLOAT)
            return value.as_float;
    }
};
```

```
        else
            return (float)value.as_int;
    }
};
```

```
union U {
    int a;
    short b;
    float c;
};
U u;

u.a = 10;
if (u.b == 10) {
    // this is undefined behavior since 'a' was the last member to be
    // written to. A lot of compilers will allow this and might issue a
    // warning, but the result will be "as expected"; this is a compiler
    // extension and cannot be guaranteed across compilers (i.e. this is
    // not compliant/portable code).
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2678/>

74:

◦

C++ 11.

- `regex_match` //
- `regex_search` //
- `regex_replace` //
- `regex_token_iterator` // ◦ ◦
- `regex_iterator` // ◦ ◦

<pre>bool regex_match(BidirectionalIterator first, BidirectionalIterator last, smatch& sm, const regex& re, regex_constraints::match_flag_type flags)</pre>	BidirectionalIterator <code>smatch</code> <code>match_results</code> <code>BidirectionalIterator</code> <code>smatch</code> omitted <code>refirstlast</code>
<pre>bool regex_match(const string& str, smatch& sm, const regex re&, regex_constraints::match_flag_type flags)</pre>	string <code>const char*</code> L-Value <code>string</code> <i>R</i> <i>string</i> smatch <code>match</code> <code>str</code> <code>match_results</code> <code>smatch</code> <code>re</code> <code>str</code>

Examples

regex_match regex_search

```
const auto input = "Some people, when confronted with a problem, think \"I know, I'll use
regular expressions.\"";
smatch sm;

cout << input << endl;

// If input ends in a quotation that contains a word that begins with "reg" and another word
begining with "ex" then capture the preceeding portion of input
if (regex_match(input, sm, regex("(.*)\".*\breg.*\bex.*\"\\s*$"))) {
    const auto capture = sm[1].str();

    cout << '\t' << capture << endl; // Outputs: "\tSome people, when confronted with a
problem, think\n"

    // Search our capture for "a problem" or "# problems"
    if (regex_search(capture, sm, regex("(a|d+)\s+problems?"))) {
        const auto count = sm[1] == "a"s ? 1 : stoi(sm[1]);

        cout << '\t' << count << (count > 1 ? " problems\n" : " problem\n"); // Outputs: "\t1
problem\n"
        cout << "Now they have " << count + 1 << " problems.\n"; // Ouputs: "Now they have 2
problems\n"
    }
}
```

regex_replace

One True Brace

```
const auto input = "if (KnR)\n\tfoo();\nif (spaces) {\n    foo();\n}\nif (allman)\n{\n\tfoo();\n}\nif (horstmann)\n{\tfoo();\n}\nif (pico)\n{\tfoo(); }\nif (whitesmiths)\n{\t{\n\tfoo();\n\t}\n}";

cout << input << regex_replace(input, regex("(.*?)\\s*\\{?\\s*(.+?;?)\\s*\\}\\s*"), "$1\n\t\t$2\n\n") << endl;
```

regex_token_iterator

[std::regex_token_iterator](#) ◦

```
const auto input = "please split,this, csv, ,line, \\, \n";
const regex re{ "((?:[^\t\\,]|\\t|\\.)+)(?:,|$)" };
const vector<string> m_vecFields{ sregex_token_iterator(cbegin(input), cend(input), re, 1),
sregex_token_iterator() };

cout << input << endl;

copy(cbegin(m_vecFields), cend(m_vecFields), ostream_iterator<string>(cout, "\n"));
```

[regexL](#) ◦ [R](#) ◦

regex_iterator

[regex_iterator](#) ◦ [regex_iterator](#) ◦ [regex_iterator](#) ◦ [match_result](#) ◦ ◦ **C++** ◦

```
enum TOKENS {
    NUMBER,
    ADDITION,
    SUBTRACTION,
    MULTIPLICATION,
    DIVISION,
    EQUALITY,
    OPEN_PARENTHESIS,
    CLOSE_PARENTHESIS
};
```

```
const auto input = "42/2 + -8\t=\n(2 + 2) * 2 * 2 -3"s regex_iterator
```

```
vector<TOKENS> tokens;
const regex re{ "\\s*(\\(\\?)\\s*(-?\\s*\\d+)\\s*(\\))\\s*(?:\\(\\+)|\\(-)|\\(\\*)|\\(/)|\\(=))" };

for_each(sregex_iterator(cbegin(input), cend(input), re), sregex_iterator(), [&](const auto& i) {
    if(i[1].length() > 0) {
        tokens.push_back(OPEN_PARENTHESIS);
    }

    tokens.push_back(i[2].str().front() == '-' ? NEGATIVE_NUMBER : NON_NEGATIVE_NUMBER);
});
```

```

if(i[3].length() > 0) {
    tokens.push_back(CLOSE_PARENTHESIS);
}

auto it = next(cbegin(i), 4);

for(int result = ADDITION; it != cend(i); ++result, ++it) {
    if (it->length() > 0U) {
        tokens.push_back(static_cast<TOKENS>(result));
        break;
    }
}
});

match_results<string::const_reverse_iterator> sm;

if(regex_search(crbegin(input), crend(input), sm, regex{ tokens.back() == SUBTRACTION ?
"^\s*\d+\s*-?\s*(-?)" : "^\s*\d+\s*(-?)" })) {
    tokens.push_back(sm[1].length() == 0 ? NON_NEGATIVE_NUMBER : NEGATIVE_NUMBER);
}

```

regexLR Visual Studio regex_iterator Bug

```

std::vector<std::string> split(const std::string &str, std::string regex)
{
    std::regex r{ regex };
    std::sregex_token_iterator start{ str.begin(), str.end(), r, -1 }, end;
    return std::vector<std::string>(start, end);
}

```

```
split("Some string\t with whitespace ", "\s+"); // "Some", "string", "with", "whitespace"
```

const string input = "123456789012";
 regex_match(input, regex("\\d*")) regex_match(input, regex("\\d+"))
 input "123"7

```
regex_match(input, regex("\\d{7,}"))
```

input "123456789012". nm input 711

```
regex_match(input, regex("\\d{7,11}"));
```

[7,11]"123456789"

```
regex_match(input, regex("\\d?\\d{7,10}"))
```

10\d{7,10} . \d{0,1} . .

```
regex_match(input, regex("(?:\\d{3,4})?\\d{7}"))
```

\d{7} 7 . 734 .

`\d{3,4}?\d{7} \d{3,4}?` 343. input "1234567".

◦ ◦ ◦ `regex_match(input, regex("\\d{3,4}+\\d{7}"))` "1234567890" input `\d{3,4}+3 \d{3,4}+4`.

```
regex_match(input, regex("(?:.*\\d{3,4}){3}"))
```

input

```
123 456 7890
123-456-7890
(123)456-7890
123456 - 7890
```

input

```
12345 - 67890
```

`. *34` '0'. *`\d{3,4}`;.*

C ++4

- ^
- \$
- `\b\W`
- `\B\w`

```
auto input = "+1--12*123/+1234"s;
smatch sm;

if(regex_search(input, sm, regex{ "(?:^|\\b\\W) ([+-]?\\d+)" })) {

    do {
        cout << sm[1] << endl;
        input = sm.suffix().str();
    } while(regex_search(input, sm, regex{ "(?:^\\W|\\b\\W) ([+-]?\\d+)" }));
}
```

◦

<https://riptutorial.com/zh-TW/cplusplus/topic/1681/>

75: /GCC

Examples

o

```
#include <iostream>

int main(int argc, char *argv[])
{
    {
        int i = 2;
    }

    std::cout << i << std::endl; // i is not in the scope of the main function

    return 0;
}
```

```
#include <iostream>

int main(int argc, char *argv[])
{
    {
        int i = 2;
        std::cout << i << std::endl;
    }

    return 0;
}
```

```
#include <iostream> std::cout
```

```
#include <iostream>

int main(int argc, char *argv[])
{
    doCompile();

    return 0;
}

void doCompile()
{
    std::cout << "No!" << std::endl;
}
```

```
#include <iostream>

void doCompile(); // forward declare the function
```

```
int main(int argc, char *argv[])
{
    doCompile();

    return 0;
}

void doCompile()
{
    std::cout << "No!" << std::endl;
}
```

```
#include <iostream>

void doCompile() // define the function before using it
{
    std::cout << "No!" << std::endl;
}

int main(int argc, char *argv[])
{
    doCompile();

    return 0;
}
```

◦ ◦

~***!

◦ ◦

QMAKE

```
LIBS += nameOfLib
```

cmake

```
TARGET_LINK_LIBRARIES(target nameOfLib)
```

g ++

```
g++ -o main main.cpp -Llibrary/dir -lnameOfLib
```

.cpp functionsModule.cpp

```
g++ -o binName main.o functionsModule.o
```

```
#include "someFile.hpp" ◦
```


QMAKE

```
INCLUDEPATH += dir/Of/File
```

cmake

```
include_directories (dir/Of/File)
```

g ++

```
g++ -o main main.cpp -I dir/Of/File
```

[/GCC https://riptutorial.com/zh-TW/cplusplus/topic/4256/--gcc-](https://riptutorial.com/zh-TW/cplusplus/topic/4256/--gcc-)

76:

'const'. const. ""。。

。 const。

const。。

Examples

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

class A {
public:
    map<string, string> * mapOfStrings;
public:
    A() {
        mapOfStrings = new map<string, string>();
    }

    void insertEntry(string const & key, string const & value) const {
        (*mapOfStrings)[key] = value;           // This works? Yes it does.
        delete mapOfStrings;                   // This also works
        mapOfStrings = new map<string, string>(); // This * does * not work
    }

    void refresh() {
        delete mapOfStrings;
        mapOfStrings = new map<string, string>(); // Works as refresh is non const function
    }

    void getEntry(string const & key) const {
        cout << mapOfStrings->at(key);
    }
};

int main(int argc, char* argv[]) {

    A var;
    var.insertEntry("abc", "abcValue");
    var.getEntry("abc");
    getchar();
    return 0;
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7120/>

77:

- C++ STL ◦ STL structs/classes/arrays ◦

Examples

;const ◦

```
void calculate(int a, int b, int& c, int& d, int& e, int& f) {
    c = a + b;
    d = a - b;
    e = a * b;
    f = a / b;
}
```

```
void calculate(int a, int b, int* c, int* d, int* e, int* f) {
    *c = a + b;
    *d = a - b;
    *e = a * b;
    *f = a / b;
}
```

'OUT' #define ◦ ;

```
#define OUT

void calculate(int a, int b, OUT int& c) {
    c = a + b;
}
```

std::tuple

C++ 11

`std::tuple`

```
std::tuple<int, int, int, int> foo(int a, int b) { // or auto (C++14)
    return std::make_tuple(a + b, a - b, a * b, a / b);
}
```

C++ 17

C++ 17

```
std::tuple<int, int, int, int> foo(int a, int b) {
    return {a + b, a - b, a * b, a / b};
}
```

tuple `std::get`

```
auto mrvs = foo(5, 12);
auto add = std::get<0>(mrvs);
auto sub = std::get<1>(mrvs);
auto mul = std::get<2>(mrvs);
auto div = std::get<3>(mrvs);
```

`std::tie`tuple

```
int add, sub, mul, div;
std::tie(add, sub, mul, div) = foo(5, 12);
```

`std::ignore`

```
std::tie(add, sub, std::ignore, div) = foo(5, 12);
```

C++ 17

`std::tie`

```
auto [add, sub, mul, div] = foo(5,12);
```

`std::tie``std::make_tuple` °

```
std::tuple<int&, int&> minmax( int& a, int& b ) {
    if (b<a)
        return std::tie(b,a);
    else
        return std::tie(a,b);
}
```

```
void increase_least(int& a, int& b) {
    std::get<0>(minmax(a,b))++;
}
```

`std::forward_as_tuple``std::tie` ; °

std :: array

C++ 11

`std::array` °

```
std::array<int, 4> bar(int a, int b) {
    return { a + b, a - b, a * b, a / b };
}
```

int bar[4] **C** ° c++ **std** ° atsize °

std :: pair

`std::pair`

```
#include <utility>
std::pair<int, int> foo(int a, int b) {
    return std::make_pair(a+b, a-b);
}
```

C++ 11 `std::make_pair`

C++ 11

```
#include <utility>
std::pair<int, int> foo(int a, int b) {
    return {a+b, a-b};
}
```

`first second std::pair`

```
std::pair<int, int> mrvs = foo(5, 12);
std::cout << mrvs.first + mrvs.second << std::endl;
```

10

struct

`struct`

C++ 11

```
struct foo_return_type {
    int add;
    int sub;
    int mul;
    int div;
};

foo_return_type foo(int a, int b) {
    return {a + b, a - b, a * b, a / b};
}

auto calc = foo(5, 12);
```

C++ 11

```
struct foo_return_type {
    int add;
    int sub;
    int mul;
    int div;
    foo_return_type(int add, int sub, int mul, int div)
        : add(add), sub(sub), mul(mul), div(div) {}
};

foo_return_type foo(int a, int b) {
    return foo_return_type(a + b, a - b, a * b, a / b);
}
```

```
}  
  
foo_return_type calc = foo(5, 12);
```

```
struct calcfoo()
```

```
std::cout << calc.add << ' ' << calc.sub << ' ' << calc.mul << ' ' << calc.div << '\n';
```

```
17 -7 60 0
```

```
struct ° °
```

C++ 17

```
struct ° in-parametersout
```

```
int a=5, b=12;  
auto[add, sub, mul, div] = foo(a, b);  
std::cout << add << ' ' << sub << ' ' << mul << ' ' << div << '\n';
```

```
° struct ° °
```

C++ 17

```
C++ 17 std::tie()
```

```
std::map<std::string, int> m;  
  
// insert an element into the map and check if insertion succeeded  
auto [iterator, success] = m.insert({"Hello", 42});  
  
if (success) {  
    // your code goes here  
}  
  
// iterate over all elements without having to use the cryptic 'first' and 'second' names  
for (auto const& [key, value] : m) {  
    std::cout << "The value for " << key << " is " << value << '\n';  
}
```

```
std::pair std::tuple
```

```
struct A { int x; };  
struct B : A { int y; };  
B foo();  
  
// with structured bindings  
const auto [x, y] = foo();  
  
// equivalent code without structured bindings  
const auto result = foo();  
auto& x = result.x;  
auto& y = result.y;
```

“”◦ tuple_size tuple_elementget

```
namespace my_ns {
    struct my_type {
        int x;
        double d;
        std::string s;
    };
    struct my_type_view {
        my_type* ptr;
    };
}

namespace std {
    template<>
    struct tuple_size<my_ns::my_type_view> : std::integral_constant<std::size_t, 3>
    {};

    template<> struct tuple_element<my_ns::my_type_view, 0>{ using type = int; };
    template<> struct tuple_element<my_ns::my_type_view, 1>{ using type = double; };
    template<> struct tuple_element<my_ns::my_type_view, 2>{ using type = std::string; };
}

namespace my_ns {
    template<std::size_t I>
    decltype(auto) get(my_type_view const& v) {
        if constexpr (I == 0)
            return v.ptr->x;
        else if constexpr (I == 1)
            return v.ptr->d;
        else if constexpr (I == 2)
            return v.ptr->s;
        static_assert(I < 3, "Only 3 elements");
    }
}
```

```
my_ns::my_type t{1, 3.14, "hello world"};

my_ns::my_type_view foo() {
    return {&t};
}

int main() {
    auto[x, d, s] = foo();
    std::cout << x << ',' << d << ',' << s << '\n';
}
```

C++ 11

```
template <class F>
void foo(int a, int b, F consumer) {
    consumer(a + b, a - b, a * b, a / b);
}

// use is simple... ignoring some results is possible as well
foo(5, 12, [](int sum, int , int , int ){
    std::cout << "sum is " << sum << '\n';
});
```

“”。

C++ 17

```
template<class Tuple>
struct continuation {
    Tuple t;
    template<class F>
    decltype(auto) operator->*(F&& f)&&{
        return std::apply( std::forward<F>(f), std::move(t) );
    }
};
std::tuple<int,int,int,int> foo(int a, int b);

continuation(foo(5,12))->*[](int sum, auto&&...) {
    std::cout << "sum is " << sum << '\n';
};
```

C++ 14 C++ 11。

std::vector

std::vector。int std::vector

```
#include <vector>
#include <iostream>

// the following function returns all integers between and including 'a' and 'b' in a vector
// (the function can return up to std::vector::max_size elements with the vector, given that
// the system's main memory can hold that many items)
std::vector<int> fillVectorFrom(int a, int b) {
    std::vector<int> temp;
    for (int i = a; i <= b; i++) {
        temp.push_back(i);
    }
    return temp;
}

int main() {
    // assigns the filled vector created inside the function to the new vector 'v'
    std::vector<int> v = fillVectorFrom(1, 10);

    // prints "1 2 3 4 5 6 7 8 9 10 "
    for (int i = 0; i < v.size(); i++) {
        std::cout << v[i] << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

。。

```
template<typename Incrementable, typename OutputIterator>
void generate_sequence(Incrementable from, Incrementable to, OutputIterator output) {
    for (Incrementable k = from; k != to; ++k)
        *output++ = k;
```



```
}
```

```
std::vector<int> digits;  
generate_sequence(0, 10, std::back_inserter(digits));  
// digits now contains {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/487/>

78:

- C ++3forwhiledo ... while◦
 - ;
 - ;
 - for *for-init-statement* ; *condition* ; *expression statement* ;
 - for *range-declaration* *for-range-initializer* ;
 -
 -

algorithm◦

◦

◦

Examples

C ++ 11

for

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

v val◦

```
for (for-range-declaration : for-range-initializer ) statement
```

```
{
    auto&& __range = for-range-initializer;
    auto __begin = begin-expr, __end = end-expr;
    for (; __begin != __end; ++__begin) {
        for-range-declaration = *__begin;
        statement
    }
}
```

C ++ 17

```
{
    auto&& __range = for-range-initializer;
    auto __begin = begin-expr;
```

```

auto __end = end-expr; // end is allowed to be a different type than begin in C++17
for (; __begin != __end; ++__begin) {
    for-range-declaration = *__begin;
    statement
}
}

```

C++ 20 Ranges TS

```

{
    auto&& __range = v;
    auto __begin = v.begin(), __end = v.end();
    for (; __begin != __end; ++__begin) {
        auto val = *__begin;
        std::cout << val << " ";
    }
}

```

auto val ◦ const auto &val ◦ auto ; ◦

◦

```

vector<float> v = {0.4f, 12.5f, 16.234f};

for(float &val: v)
{
    std::cout << val << " ";
}

```

const const

```

const vector<float> v = {0.4f, 12.5f, 16.234f};

for(const float &val: v)
{
    std::cout << val << " ";
}

```

const ◦ ◦

```

vector<bool> v(10);

for(auto&& val: v)
{
    val = true;
}

```

“” for

- ```
float arr[] = {0.4f, 12.5f, 16.234f};

for(auto val: arr)
{
 std::cout << val << " ";
}
```

```
}
```

```
float *arr = new float[3]{0.4f, 12.5f, 16.234f};

for(auto val: arr) //Compile error.
{
 std::cout << val << " ";
}
```

- begin()end() ◦

```
struct Rng
{
 float arr[3];

 // pointers are iterators
 const float* begin() const {return &arr[0];}
 const float* end() const {return &arr[3];}
 float* begin() {return &arr[0];}
 float* end() {return &arr[3];}
};

int main()
{
 Rng rng = {{0.4f, 12.5f, 16.234f}};

 for(auto val: rng)
 {
 std::cout << val << " ";
 }
}
```

- begin(type)end(type)end(type) type ◦

```
namespace Mine
{
 struct Rng {float arr[3];};

 // pointers are iterators
 const float* begin(const Rng &rng) {return &rng.arr[0];}
 const float* end(const Rng &rng) {return &rng.arr[3];}
 float* begin(Rng &rng) {return &rng.arr[0];}
 float* end(Rng &rng) {return &rng.arr[3];}
}

int main()
{
 Mine::Rng rng = {{0.4f, 12.5f, 16.234f}};

 for(auto val: rng)
 {
 std::cout << val << " ";
 }
}
```

forloop bodyconditiontrue◦ initialization statement◦ iteration execution◦

for

```
for (/*initialization statement*/; /*condition*/; /*iteration execution*/)
{
 // body of the loop
}
```

- initialization statement for° int i = 0, a = 2, b = 3 ° ° °
- condition false °
- iteration execution iteration executionforbreak goto return° iteration executiona++, b+=10, c=b+a °

forwhile

```
/*initialization*/
while (/*condition*/)
{
 // body of the loop; using 'continue' will skip to increment part below
 /*iteration execution*/
}
```

for°

```
for(int i = 0; i < 10; i++) {
 std::cout << i << std::endl;
}
```

```
for(int a = 0, b = 10, c = 20; (a+b+c < 100); c--, b++, a+=c) {
 std::cout << a << " " << b << " " << c << std::endl;
}
```

```
int i = 99; //i = 99
for(int i = 0; i < 10; i++) { //we declare a new variable i
 //some operations, the value of i ranges from 0 to 9 during loop execution
}
//after the loop is executed, we can access i with value of 99
```

```
int i = 99; //i = 99
for(i = 0; i < 10; i++) { //we are using already declared variable i
 //some operations, the value of i ranges from 0 to 9 during loop execution
}
//after the loop is executed, we can access i with value of 10
```

- - ° °
- forfor°
- initialization statementcondition°

010

```
for (int counter = 0; counter <= 10; ++counter)
{
```

```

 std::cout << counter << '\n';
}
// counter is not accessible here (had value 11 at the end)

```

- `int counter = 0` `counter` **0**. `for`
- `counter <= 10` `counter` **10**. `true` ◦ `false` ◦
- `++counter` `counter` **1** ◦

```

// infinite loop
for (;;)
 std::cout << "Never ending!\n";

```

while

```

// infinite loop
while (true)
 std::cout << "Never ending!\n";

```

break goto return ◦

<algorithm> **STL** vector

```

std::vector<std::string> names = {"Albert Einstein", "Stephen Hawking", "Michael Ellis"};
for(std::vector<std::string>::iterator it = names.begin(); it != names.end(); ++it) {
 std::cout << *it << std::endl;
}

```

while false ◦ ◦

## 09

```

int i = 0;
while (i < 10)
{
 std::cout << i << " ";
 ++i; // Increment counter
}
std::cout << std::endl; // End of line; "0 1 2 3 4 5 6 7 8 9" is printed to the console

```

## C++ 17

## C++ 17

```

while (int i = 0; i < 10)
//... The rest is the same

```

```

while (true)
{
 // Do something forever (however, you can exit the loop by calling 'break')
}

```

while do...while

## ◦ do-while ◦

forwhile◦

```
for (int i = 0; i < 5; ++i) {
 do_something(i);
}
// i is no longer in scope.

for (auto& a : some_container) {
 a.do_something();
}
// a is no longer in scope.

while(std::shared_ptr<Object> p = get_object()) {
 p->do_something();
}
// p is no longer in scope.
```

do...whiledo...while;

```
//This doesn't compile
do {
 s = do_something();
} while (short s > 0);

// Good
short s;
do {
 s = do_something();
} while (s > 0);
```

do...whilewhile ◦

## Do-while

*do-whilewhile* ◦ ◦

0 false

```
int i =0;
do
{
 std::cout << i;
 ++i; // Increment counter
}
while (i < 0);
std::cout << std::endl; // End of line; 0 is printed to the console
```

while(condition);while(condition);while(condition); *do-while*◦

*do-while*false

```
int i =0;
```

```

while (i < 0)
{
 std::cout << i;
 ++i; // Increment counter
}
std::cout << std::endl; // End of line; nothing is printed to the console

```

break goto return *while* false ◦

```

int i = 0;
do
{
 std::cout << i;
 ++i; // Increment counter
 if (i > 5)
 {
 break;
 }
}
while (true);
std::cout << std::endl; // End of line; 0 1 2 3 4 5 is printed to the console

```

## do-while

```

#define BAD_MACRO(x) f1(x); f2(x); f3(x);

// Only the call to f1 is protected by the condition here
if (cond) BAD_MACRO(var);

#define GOOD_MACRO(x) do { f1(x); f2(x); f3(x); } while(0)

// All calls are protected here
if (cond) GOOD_MACRO(var);

```

◦ ◦ break continue ◦

break ◦

```

for (int i = 0; i < 10; i++)
{
 if (i == 4)
 break; // this will immediately exit our loop
 std::cout << i << '\n';
}

```

```

1
2
3

```

continue ◦

```

for (int i = 0; i < 6; i++)
{
 if (i % 2 == 0) // evaluates to true if i is even

```



```

 continue; // this will immediately go back to the start of the loop
/* the next line will only be reached if the above "continue" statement
does not execute */
std::cout << i << " is an odd number\n";
}

```

```

1 is an odd number
3 is an odd number
5 is an odd number

```

breakcontinue ° ° breakfor

```

for (int i = 0; i < 4; i++)
{
 std::cout << i << '\n';
}

```

continue

```

for (int i = 0; i < 6; i++)
{
 if (i % 2 != 0) {
 std::cout << i << " is an odd number\n";
 }
}

```

-

for°

```

template<class Iterator, class Sentinel=Iterator>
struct range_t {
 Iterator b;
 Sentinel e;
 Iterator begin() const { return b; }
 Sentinel end() const { return e; }
 bool empty() const { return begin()==end(); }
 range_t without_front(std::size_t count=1) const {
 if (std::is_same< std::random_access_iterator_tag, typename
std::iterator_traits<Iterator>::iterator_category >{}) {
 count = (std::min)(std::size_t(std::distance(b,e)), count);
 }
 return {std::next(b, count), e};
 }
 range_t without_back(std::size_t count=1) const {
 if (std::is_same< std::random_access_iterator_tag, typename
std::iterator_traits<Iterator>::iterator_category >{}) {
 count = (std::min)(std::size_t(std::distance(b,e)), count);
 }
 return {b, std::prev(e, count)};
 }
};

template<class Iterator, class Sentinel>
range_t<Iterator, Sentinel> range(Iterator b, Sentinel e) {
 return {b,e};
}

```

```
}
template<class Iterable>
auto range(Iterable& r) {
 using std::begin; using std::end;
 return range(begin(r),end(r));
}

template<class C>
auto except_first(C& c) {
 auto r = range(c);
 if (r.empty()) return r;
 return r.without_front();
}
```

```
std::vector<int> v = {1,2,3,4};

for (auto i : except_first(v))
 std::cout << i << '\n';
```

```
2
3
4
```

forfor(:range\_expression) for°

<https://riptutorial.com/zh-TW/cplusplus/topic/589/>

# 79:

|    |    |    |    |    |    |    |    |     |     |   |
|----|----|----|----|----|----|----|----|-----|-----|---|
| +  | -  | *  | /  | \  |    | << | >> |     |     |   |
| += | -= | *= | /= | =  | \= | =  | =  | <<= | >>= | = |
| == | =  | <  | >  | <= | >= | && |    | *.  | ->* |   |

|    |
|----|
| && |
|    |

## Examples

◦

- (... op pack)

```
((Pack1 op Pack2) op ...) op PackN
```

- **Unary Right Fold** (pack op ...)

```
Pack1 op (... (Pack(N-1) op PackN))
```

```
template<typename... Ts>
int sum(Ts... args)
{
 return (... + args); //Unary left fold
 //return (args + ...); //Unary right fold

 // The two are equivalent if the operator is associative.
 // For +, ((1+2)+3) (left fold) == (1+(2+3)) (right fold)
 // For -, ((1-2)-3) (left fold) != (1-(2-3)) (right fold)
}

int result = sum(1, 2, 3); // 6
```

◦

- - (value op ... op pack) -

```
((Value op Pack1) op Pack2) op ... op PackN
```

- **Binary Right Fold** (pack op ... op value) -

```
Pack1 op (... op (Pack(N-1) op (PackN op Value)))
```

```
template<typename... Ts>
int removeFrom(int num, Ts... args)
{
 return (num - ... - args); //Binary left fold
 // Note that a binary right fold cannot be used
 // due to the lack of associativity of operator-
}

int result = removeFrom(1000, 5, 10, 15); // 'result' is 1000 - 5 - 10 - 15 = 970
```

## ◦ C++ 11

```
template <class... Ts>
void print_all(std::ostream& os, Ts const&... args) {
 using expander = int[];
 (void)expander{0,
 (void(os << args), 0)...
 };
}
```

## fold

```
template <class... Ts>
void print_all(std::ostream& os, Ts const&... args) {
 (void(os << args), ...);
}
```

◦

<https://riptutorial.com/zh-TW/cplusplus/topic/2676/>

# 80:

- Class.....
  - type \* ptr =Class :: member; //
  - type Class :: \* ptr =Class :: member; //
- - ;
  - Class \* p =instance;
- - ptr =Class :: i; //i
  - instance◦ \* ptr = 1; //i
  - p -> \* ptr = 1; //pi
- - ptr =Class :: F; //'F'
  - \* PTR5; //F
  - -> \* PTR6; //pF.

## Examples

static C / C ++

- class ;
- public protected private◦

static class

```
typedef int Fn(int); // Fn is a type-of function that accepts an int and returns an int

// Note that MyFn() is of type 'Fn'
int MyFn(int i) { return 2*i; }

class Class {
public:
 // Note that Static() is of type 'Fn'
 static int Static(int i) { return 3*i; }
}; // Class

int main() {
 Fn *fn; // fn is a pointer to a type-of Fn

 fn = &MyFn; // Point to one function
 fn(3); // Call it
 fn = &Class::Static; // Point to the other function
 fn(4); // Call it
} // main()
```

“”◦ ...

```
typedef int Fn(int); // Fn is a type-of function that accepts an int and returns an int
```

```

class Class {
public:
 // Note that A() is of type 'Fn'
 int A(int a) { return 2*a; }
 // Note that B() is of type 'Fn'
 int B(int b) { return 3*b; }
}; // Class

int main() {
 Class c; // Need a Class instance to play with
 Class *p = &c; // Need a Class pointer to play with

 Fn Class::*fn; // fn is a pointer to a type-of Fn within Class

 fn = &Class::A; // fn now points to A within any Class
 (c.*fn)(5); // Pass 5 to c's function A (via fn)
 fn = &Class::B; // fn now points to B within any Class
 (p->*fn)(6); // Pass 6 to c's (via p) function B (via fn)
} // main()

```

.\*->\*

class "" class...

```

class Class {
public:
 int x, y, z;
 char m, n, o;
}; // Class

int x; // Global variable

int main() {
 Class c; // Need a Class instance to play with
 Class *p = &c; // Need a Class pointer to play with

 int *p_i; // Pointer to an int

 p_i = &x; // Now pointing to x
 p_i = &c.x; // Now pointing to c's x

 int Class::*p_C_i; // Pointer to an int within Class

 p_C_i = &Class::x; // Point to x within any Class
 int i = c.*p_C_i; // Use p_C_i to fetch x from c's instance
 p_C_i = &Class::y; // Point to y within any Class
 i = c.*p_C_i; // Use p_C_i to fetch y from c's instance

 p_C_i = &Class::m; // ERROR! m is a char, not an int!

 char Class::*p_C_c = &Class::m; // That's better...
} // main()

```

- int Class::\*ptr; ◦
- .\* ◦
- ->\*-> ◦

static C/C++

- class ;
- public protectedprivate°

staticclass

```
class Class {
public:
 static int i;
}; // Class

int Class::i = 1; // Define the value of i (and where it's stored!)

int j = 2; // Just another global variable

int main() {
 int k = 3; // Local variable

 int *p;

 p = &k; // Point to k
 *p = 2; // Modify it
 p = &j; // Point to j
 *p = 3; // Modify it
 p = &Class::i; // Point to Class::i
 *p = 4; // Modify it
} // main()
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2130/>

# 81:

o o o

```
“” * “” & “” -> “” * “” -> “” & “”
```

- `<> * <>;`
- `<> * <> =<>;`
- `<> * <> = <>;`
- `int * foo; //`
- `int * bar = myIntVar;`
- `long * bar [2];`
- `long * bar [] = {myLongVar1myLongVar2}; //long * bar [2]`

o

```
int* a, b, c; //Only a is a pointer, the others are regular ints.

int* a, *b, *c; //These are three pointers!

int *foo[2]; //Both *foo[0] and *foo[1] are pointers.
```

## Examples

### C++ 11

C++ 11 `nullptr` ◦ `nullptr` `NULL` ◦

```
*int *pointer_to_int; ◦
```

```
int * ◦ int ◦
```

```
// Declare a struct type `big_struct` that contains
// three long long ints.
typedef struct {
 long long int foo1;
 long long int foo2;
 long long int foo3;
} big_struct;

// Create a variable `bar` of type `big_struct`
big_struct bar;
// Create a variable `p_bar` of type `pointer to big_struct`.
// Initialize it to `nullptr` (a null pointer).
big_struct *p_bar0 = nullptr;

// Print the size of `bar`
std::cout << "sizeof(bar) = " << sizeof(bar) << std::endl;
// Print the size of `p_bar`.
std::cout << "sizeof(p_bar0) = " << sizeof(p_bar0) << std::endl;
```



```
/* Produces:
 sizeof(bar) = 24
 sizeof(p_bar0) = 8
*/
```

---

```
;
nullptr
```

## operator &

```
// Copy `p_bar0` into `p_bar1`.
big_struct *p_bar1 = p_bar0;

// Take the address of `bar` into `p_bar2`
big_struct *p_bar2 = &bar;

// p_bar1 is now nullptr, p_bar2 is &bar.

p_bar0 = p_bar2;

// p_bar0 is now &bar.

p_bar2 = nullptr;

// p_bar0 == &bar
// p_bar1 == nullptr
// p_bar2 == nullptr
```

- ;
- nullptr
- 

---

```
& * const
```

```
(*p_bar0).fool = 5;

// `p_bar0` points to `bar`. This prints 5.
std::cout << "bar.fool = " << bar.fool << std::endl;

// Assign the value pointed to by `p_bar0` to `baz`.
big_struct baz;
baz = *p_bar0;

// Now `baz` contains a copy of the data pointed to by `p_bar0`.
// Indeed, it contains a copy of `bar`.

// Prints 5 as well
std::cout << "baz.fool = " << baz.fool << std::endl;
```

```
*.->
```

```
std::cout << "bar.foo1 = " << (*p_bar0).foo1 << std::endl; // Prints 5
std::cout << "bar.foo1 = " << p_bar0->foo1 << std::endl; // Prints 5
```

◦ ◦

```
big_struct *never_do_this() {
 // This is a local variable. Outside `never_do_this` it doesn't exist.
 big_struct retval;
 retval.foo1 = 11;
 // This returns the address of `retval`.
 return &retval;
 // `retval` is destroyed and any code using the value returned
 // by `never_do_this` has a pointer to a memory location that
 // contains garbage data (or is inaccessible).
}
```

g++clang++

```
(Clang) warning: address of stack memory associated with local variable 'retval' returned [-Wreturn-stack-address]
(Gcc) warning: address of local variable `retval' returned [-Wreturn-local-addr]
```

## null

```
void naive_code(big_struct *ptr_big_struct) {
 // ... some code which doesn't check if `ptr_big_struct` is valid.
 ptr_big_struct->foo1 = 12;
}

// Segmentation fault.
naive_code(nullptr);
```

## Address-of operator ◦ Contents-ofDereference\*◦

```
int var = 20;
int *ptr;
ptr = &var;

cout << var << endl;
//Outputs 20 (The value of var)

cout << ptr << endl;
//Outputs 0x234f119 (var's memory location)

cout << *ptr << endl;
//Outputs 20(The value of the variable stored in the pointer ptr
```

\*◦ **dereference**◦ ◦



◦ ◦ ◦

◦ void◦

```
char* str = new char[10]; // str = 0x010
++str; // str = 0x011 in this case sizeof(char) = 1 byte

int* arr = new int[10]; // arr = 0x00100
++arr; // arr = 0x00104 if sizeof(int) = 4 bytes

void* ptr = (void*)new char[10];
++ptr; // void is incomplete.
```

◦ ◦

◦

1◦



;1◦ /◦

```
char* str = new char[10]; // str = 0x010
str += 2; // str = 0x010 + 2 * sizeof(char) = 0x012

int* arr = new int[10]; // arr = 0x100
arr += 2; // arr = 0x100 + 2 * sizeof(int) = 0x108, assuming sizeof(int) ==
4.
```

◦ ;◦

$PQ - Qi - j$  ◦ `std::ptrdiff_t <cstdlib>` ◦

```
char* start = new char[10]; // str = 0x010
char* test = &start[5];
std::ptrdiff_t diff = test - start; //Equal to 5.
std::ptrdiff_t diff = start - test; //Equal to -5; ptrdiff_t is signed.
```

<https://riptutorial.com/zh-TW/cplusplus/topic/3056/>

# 82:

std::sortalgorithm°

## Examples

std::sort

### C++ 11

```
#include <vector>
#include <algorithm>
#include <functional>

std::vector<int> v = {5,1,2,4,3};

//sort in ascending order (1,2,3,4,5)
std::sort(v.begin(), v.end(), std::less<int>());

// Or just:
std::sort(v.begin(), v.end());

//sort in descending order (5,4,3,2,1)
std::sort(v.begin(), v.end(), std::greater<int>());

//Or just:
std::sort(v.rbegin(), v.rend());
```

### C++ 14

### C++ 14

```
std::sort(v.begin(), v.end(), std::less<>()); // ascending order
std::sort(v.begin(), v.end(), std::greater<>()); // descending order
```

std::sortoperator< **on**std::sort bool bool ° °

sort°

```
// Include sequence containers
#include <vector>
#include <deque>
#include <list>

// Insert sorting algorithm
#include <algorithm>

class Base {
public:

 // Constructor that set variable to the value of v
 Base(int v): variable(v) {
 }
}
```

```

// Use variable to provide total order operator less
//`this` always represents the left-hand side of the compare.
bool operator<(const Base &b) const {
 return this->variable < b.variable;
}

int variable;
};

int main() {
 std::vector <Base> vector;
 std::deque <Base> deque;
 std::list <Base> list;

 // Create 2 elements to sort
 Base a(10);
 Base b(5);

 // Insert them into backs of containers
 vector.push_back(a);
 vector.push_back(b);

 deque.push_back(a);
 deque.push_back(b);

 list.push_back(a);
 list.push_back(b);

 // Now sort data using operator<(const Base &b) function
 std::sort(vector.begin(), vector.end());
 std::sort(deque.begin(), deque.end());
 // List must be sorted differently due to its design
 list.sort();

 return 0;
}

```

```

// Include sequence containers
#include <vector>
#include <deque>
#include <list>

// Insert sorting algorithm
#include <algorithm>

class Base {
public:

 // Constructor that set variable to the value of v
 Base(int v): variable(v) {
 }

 int variable;
};

bool compare(const Base &a, const Base &b) {
 return a.variable < b.variable;
}

```

```

int main() {
 std::vector <Base> vector;
 std::deque <Base> deque;
 std::list <Base> list;

 // Create 2 elements to sort
 Base a(10);
 Base b(5);

 // Insert them into backs of containers
 vector.push_back(a);
 vector.push_back(b);

 deque.push_back(a);
 deque.push_back(b);

 list.push_back(a);
 list.push_back(b);

 // Now sort data using comparing function
 std::sort(vector.begin(), vector.end(), compare);
 std::sort(deque.begin(), deque.end(), compare);
 list.sort(compare);

 return 0;
}

```

## lambdaC ++ 11

### C ++ 11

```

// Include sequence containers
#include <vector>
#include <deque>
#include <list>
#include <array>
#include <forward_list>

// Include sorting algorithm
#include <algorithm>

class Base {
public:

 // Constructor that set variable to the value of v
 Base(int v): variable(v) {
 }

 int variable;
};

int main() {
 // Create 2 elements to sort
 Base a(10);
 Base b(5);

 // We're using C++11, so let's use initializer lists to insert items.
 std::vector <Base> vector = {a, b};
}

```

```

std::deque<Base> deque = {a, b};
std::list<Base> list = {a, b};
std::array<Base, 2> array = {a, b};
std::forward_list<Base> flist = {a, b};

// We can sort data using an inline lambda expression
std::sort(std::begin(vector), std::end(vector),
 [](const Base &a, const Base &b) { return a.variable < b.variable;});

// We can also pass a lambda object as the comparator
// and reuse the lambda multiple times
auto compare = [](const Base &a, const Base &b) {
 return a.variable < b.variable;};
std::sort(std::begin(deque), std::end(deque), compare);
std::sort(std::begin(array), std::end(array), compare);
list.sort(compare);
flist.sort(compare);

return 0;
}

```

std::sort algorithm ◦ std::sort; ◦ std::sort ◦

◦ ◦

std::sort std::sort

## C++ 11

```

#include <vector>
#include <algorithm>

std::vector<int> MyVector = {3, 1, 2}

//Default comparison of <
std::sort(MyVector.begin(), MyVector.end());

```

std::sort ◦ std::list std::forward\_list **C++ 11** std::sort ◦ sort ◦

## C++ 11

```

#include <list>
#include <algorithm>

std::list<int> MyList = {3, 1, 2}

//Default comparison of <
//Whole list only.
MyList.sort();

```

sort ◦ list forward\_list

```

void sort_sublist(std::list<int>& mylist, std::list<int>::const_iterator start,
 std::list<int>::const_iterator end) {
 //extract and sort half-open sub range denoted by start and end iterator
 std::list<int> tmp;

```

```

tmp.splice(tmp.begin(), list, start, end);
tmp.sort();
//re-insert range at the point we extracted it from
list.splice(end, tmp);
}

```

## std::map

◦ std::string ◦

```

#include <iostream>
#include <utility>
#include <map>

int main()
{
 std::map<double, std::string> sorted_map;
 // Sort the names of the planets according to their size
 sorted_map.insert(std::make_pair(0.3829, "Mercury"));
 sorted_map.insert(std::make_pair(0.9499, "Venus"));
 sorted_map.insert(std::make_pair(1, "Earth"));
 sorted_map.insert(std::make_pair(0.532, "Mars"));
 sorted_map.insert(std::make_pair(10.97, "Jupiter"));
 sorted_map.insert(std::make_pair(9.14, "Saturn"));
 sorted_map.insert(std::make_pair(3.981, "Uranus"));
 sorted_map.insert(std::make_pair(3.865, "Neptune"));

 for (auto const& entry: sorted_map)
 {
 std::cout << entry.second << " (" << entry.first << " of Earth's radius)" << '\n';
 }
}

```

```

Mercury (0.3829 of Earth's radius)
Mars (0.532 of Earth's radius)
Venus (0.9499 of Earth's radius)
Earth (1 of Earth's radius)
Neptune (3.865 of Earth's radius)
Uranus (3.981 of Earth's radius)
Saturn (9.14 of Earth's radius)
Jupiter (10.97 of Earth's radius)

```

multimap ◦

std::greater<>

```

#include <iostream>
#include <utility>
#include <map>

int main()
{
 std::multimap<int, std::string, std::greater<int>> sorted_map;
 // Sort the names of animals in descending order of the number of legs
 sorted_map.insert(std::make_pair(6, "bug"));
 sorted_map.insert(std::make_pair(4, "cat"));
}

```



```

sorted_map.insert(std::make_pair(100, "centipede"));
sorted_map.insert(std::make_pair(2, "chicken"));
sorted_map.insert(std::make_pair(0, "fish"));
sorted_map.insert(std::make_pair(4, "horse"));
sorted_map.insert(std::make_pair(8, "spider"));

for (auto const& entry: sorted_map)
{
 std::cout << entry.second << " (has " << entry.first << " legs)" << '\n';
}
}

```

```

centipede (has 100 legs)
spider (has 8 legs)
bug (has 6 legs)
cat (has 4 legs)
horse (has 4 legs)
chicken (has 2 legs)
fish (has 0 legs)

```

sortsort ◦ C◦

## C++ 11

```

int arr1[] = {36, 24, 42, 60, 59};

// sort numbers in ascending order
sort(std::begin(arr1), std::end(arr1));

// sort numbers in descending order
sort(std::begin(arr1), std::end(arr1), std::greater<int>());

```

## C++ 11""

## C++ 11

```

// Use a hard-coded number for array size
sort(arr1, arr1 + 5);

// Alternatively, use an expression
const size_t arr1_size = sizeof(arr1) / sizeof(*arr1);
sort(arr1, arr1 + arr1_size);

```

<https://riptutorial.com/zh-TW/cplusplus/topic/1675/>

# 83:

## Examples

◦

- 7237498123◦
- 237498123237499123◦
- 23749912320249472◦

C++14 Simple Quotation Mark '◦ ◦

### C ++ 14

```
long long decn = 1'000'000'00011;
long long hexn = 0xFFFF'FFF11;
long long octn = 00'23'0011;
long long binn = 0b1010'001111;
```

◦

- 1048576 1'048'576 0X100000 0x10'00000'004'000'000◦
- 1.602'176'565e-19 1.602176565e-19◦

◦

### C ++ 14

```
long long a1 = 12345678911;
long long a2 = 123'456'78911;
long long a3 = 12'34'56'78'911;
long long a4 = 12345'678911;
```

user-defined

### C ++ 14

```
std::chrono::seconds tiempo = 1'674'456s + 5'300h;
```

<https://riptutorial.com/zh-TW/cplusplus/topic/10595/>

# 84:

- ◦
- 

## Examples

- 

```
#include <stddef.h> // size_t, ptrdiff_t

//----- Machinery:

using Size = ptrdiff_t;

template< class Item, size_t n >
constexpr auto n_items(Item (&)[n]) noexcept
 -> Size
{ return n; }

//----- Usage:

#include <iostream>
using namespace std;
auto main()
 -> int
{
 int const a[] = {3, 1, 4, 1, 5, 9, 2, 6, 5, 4};
 Size const n = n_items(a);
 int b[n] = {}; // An array of the same size as a.

 (void) b;
 cout << "Size = " << n << "\n";
}

```

**C idiom** sizeof(a)/sizeof(a[0])◦

**C++ 11**

**C++ 11**

```
std::extent<decltype(MyArray)>::value;
```

```
char MyArray[] = { 'X','o','c','e' };
const auto n = std::extent<decltype(MyArray)>::value;
std::cout << n << "\n"; // Prints 4

```

**C++ 17C++**◦ size\_t Sizesize\_tg++◦

## C++ 17 `std::size` ◦

```
// Example of raw dynamic size array. It's generally better to use std::vector.
#include <algorithm> // std::sort
#include <iostream>
using namespace std;

auto int_from(istream& in) -> int { int x; in >> x; return x; }

auto main()
 -> int
{
 cout << "Sorting n integers provided by you.\n";
 cout << "n? ";
 int const n = int_from(cin);
 int* a = new int[n]; // ← Allocation of array of n items.

 for(int i = 1; i <= n; ++i)
 {
 cout << "The #" << i << " number, please: ";
 a[i-1] = int_from(cin);
 }

 sort(a, a + n);
 for(int i = 0; i < n; ++i) { cout << a[i] << ' '; }
 cout << '\n';

 delete[] a;
}
```

T a[n];n C99 VLA ◦ C++VLA ◦ new[] -expression

```
int* a = new int[n]; // ← Allocation of array of n items.
```

...delete[] -expression

```
delete[] a;
```

() new int[n] () ◦ ◦

delete[]new[] ◦ std::unique\_ptr ◦ std::vector std::vector ◦

## std::vector ◦

```
// Example of std::vector as an expanding dynamic size array.
#include <algorithm> // std::sort
#include <iostream>
#include <vector> // std::vector
using namespace std;

int int_from(std::istream& in) { int x = 0; in >> x; return x; }

int main()
{
 cout << "Sorting integers provided by you.\n";
}
```

```

cout << "You can indicate EOF via F6 in Windows or Ctrl+D in Unix-land.\n";
vector<int> a; // ← Zero size by default.

while(cin)
{
 cout << "One number, please, or indicate EOF: ";
 int const x = int_from(cin);
 if(!cin.fail()) { a.push_back(x); } // Expands as necessary.
}

sort(a.begin(), a.end());
int const n = a.size();
for(int i = 0; i < n; ++i) { cout << a[i] << ' '; }
cout << '\n';
}

```

std::vector<T> v; v.data() API push\_back

$n$  push\_back  $O(n)$

$1 + 2 + 4 + 8 + 16 = 31$   $n = 34$   $2 \times n$

vector

## 2D

```

// A fixed size raw array matrix (that is, a 2D raw array).
#include <iostream>
#include <iomanip>
using namespace std;

auto main() -> int
{
 int const n_rows = 3;
 int const n_cols = 7;
 int const m[n_rows][n_cols] = // A raw array matrix.
 {
 { 1, 2, 3, 4, 5, 6, 7 },
 { 8, 9, 10, 11, 12, 13, 14 },
 { 15, 16, 17, 18, 19, 20, 21 }
 };

 for(int y = 0; y < n_rows; ++y)
 {
 for(int x = 0; x < n_cols; ++x)
 {
 cout << setw(4) << m[y][x]; // Note: do NOT use m[y,x]!
 }
 cout << '\n';
 }
}

```

```

1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21

```

C++ [ i ] m[y]my y m[y][x] y x

“”。

C++ m[y][x]operator[] m(x,y)m.at(x,y)m.item(x,y)

## std::vector

C++ 14C++ Boost

## BoostC++

```
vector<vector<int>> m(3, vector<int>(7));
```

...vectorstd::vector nn3.m[y][x]

x y

```
// A dynamic size matrix using std::vector for storage.

//----- Machinery:
#include <algorithm> // std::copy
#include <assert.h> // assert
#include <initializer_list> // std::initializer_list
#include <vector> // std::vector
#include <stddef.h> // ptrdiff_t

namespace my {
 using Size = ptrdiff_t;
 using std::initializer_list;
 using std::vector;

 template< class Item >
 class Matrix
 {
 private:
 vector<Item> items_;
 Size n_cols_;

 auto index_for(Size const x, Size const y) const
 -> Size
 { return y*n_cols_ + x; }

 public:
 auto n_rows() const -> Size { return items_.size()/n_cols_; }
 auto n_cols() const -> Size { return n_cols_; }

 auto item(Size const x, Size const y)
 -> Item&
 { return items_[index_for(x, y)]; }

 auto item(Size const x, Size const y) const
 -> Item const&
 { return items_[index_for(x, y)]; }

 Matrix(): n_cols_(0) {}
 };
}
```

```

Matrix(Size const n_cols, Size const n_rows)
 : items_(n_cols*n_rows)
 , n_cols_(n_cols)
{}

Matrix(initializer_list< initializer_list<Item> > const& values)
 : items_(
 , n_cols_(values.size() == 0? 0 : values.begin()->size())
 {
 for(auto const& row : values)
 {
 assert(Size(row.size()) == n_cols_);
 items_.insert(items_.end(), row.begin(), row.end());
 }
 }
};
} // namespace my

//----- Usage:
using my::Matrix;

auto some_matrix()
 -> Matrix<int>
{
 return
 {
 { 1, 2, 3, 4, 5, 6, 7 },
 { 8, 9, 10, 11, 12, 13, 14 },
 { 15, 16, 17, 18, 19, 20, 21 }
 };
}

#include <iostream>
#include <iomanip>
using namespace std;
auto main() -> int
{
 Matrix<int> const m = some_matrix();
 assert(m.n_cols() == 7);
 assert(m.n_rows() == 3);
 for(int y = 0, y_end = m.n_rows(); y < y_end; ++y)
 {
 for(int x = 0, x_end = m.n_cols(); x < x_end; ++x)
 {
 cout << setw(4) << m.item(x, y); // ← Note: not `m[y][x]`!
 }
 cout << '\n';
 }
}

```

```

1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21

```

## C++。

operator()。

◦ []◦

**typ** int arrayOfInts[5]

```
int arrayOfInts[5];
```

```
int arrayOfInts[5] = {10, 20, 30, 40, 50};
```

◦ ◦ **5**

```
int arrayOfInts[] = {10, 20, 30, 40, 50};
```

◦ ◦ **5◦**

```
int arrayOfInts[5] = {10,20}; // means 10, 20, 0, 0, 0
```

◦

```
char arrayOfChars[5]; // declare the array and allocate the memory, don't initialize
```

```
char arrayOfChars[5] = { 'a', 'b', 'c', 'd', 'e' }; //declare and initialize
```

```
double arrayOfDoubles[5] = {1.14159, 2.14159, 3.14159, 4.14159, 5.14159};
```

```
string arrayOfStrings[5] = { "C++", "is", "super", "duper", "great!"};
```

**0◦**

```
int array[5] = { 10/*Element no.0*/, 20/*Element no.1*/, 30, 40, 50/*Element no.4*/};
std::cout << array[4]; //outputs 50
std::cout << array[0]; //outputs 10
```

<https://riptutorial.com/zh-TW/cplusplus/topic/3017/>



# 85: I / O.

## C++ I/O.

`std::istream` ◦

`std::ostream` ◦

`std::streambuf` ◦

`operator>>` ◦

`operator<<` ◦

**Streams** `std::locale` ◦

[<iostream> Library](#)

## Examples

3 `ifstream` `ofstream` `fstream` ◦

```
std::ifstream ifs("foo.txt"); // ifstream: Opens file "foo.txt" for reading only.
std::ofstream ofs("foo.txt"); // ofstream: Opens file "foo.txt" for writing only.
std::fstream iofs("foo.txt"); // fstream: Opens file "foo.txt" for reading and writing.
```

`open()`

```
std::ifstream ifs;
ifs.open("bar.txt"); // ifstream: Opens file "bar.txt" for reading only.

std::ofstream ofs;
ofs.open("bar.txt"); // ofstream: Opens file "bar.txt" for writing only.

std::fstream iofs;
iofs.open("bar.txt"); // fstream: Opens file "bar.txt" for reading and writing.
```

◦ ...

```
// Try to read the file 'foo.txt'.
std::ifstream ifs("fooo.txt"); // Note the typo; the file can't be opened.

// Check if the file has been opened successfully.
if (!ifs.is_open()) {
 // The file hasn't been opened; take appropriate actions here.
 throw CustomException(ifs, "File could not be opened");
}
```

## Windows

```
// Open the file 'c:\folder\foo.txt' on Windows.
std::ifstream ifs("c:\\folder\\foo.txt"); // using escaped backslashes
```

## C++ 11

```
// Open the file 'c:\folder\foo.txt' on Windows.
std::ifstream ifs(R"(c:\folder\foo.txt)"); // using raw literal
```

```
// Open the file 'c:\folder\foo.txt' on Windows.
std::ifstream ifs("c:/folder/foo.txt");
```

## C++ 11

### WindowsASCII

```
// Open the file 'пример\foo.txt' on Windows.
std::ifstream ifs(LR"(пример\foo.txt)"); // using wide characters with raw literal
```

◦

>> ◦ *foo.txt*

```
John Doe 25 4 6 1987
Jane Doe 15 5 24 1976
```

```
// Define variables.
std::ifstream is("foo.txt");
std::string firstname, lastname;
int age, bmonth, bday, byear;

// Extract firstname, lastname, age, bday month, bday day, and bday year in that order.
// Note: '>>' returns false if it reached EOF (end of file) or if the input data doesn't
// correspond to the type of the input variable (for example, the string "foo" can't be
// extracted into an 'int' variable).
while (is >> firstname >> lastname >> age >> bmonth >> bday >> byear)
 // Process the data that has been read.
```

>>

- \n ◦
- ◦

*foo.txt*

```
John
Doe 25
4 6 1987
```

```
Jane
```

```
Doe
15 5
24
1976
```

>>> ◦ while ◦ bool ◦ bool()true ◦ bool()false ◦ while ◦

```
// Opens 'foo.txt'.
std::ifstream is("foo.txt");
std::string whole_file;

// Sets position to the end of the file.
is.seekg(0, std::ios::end);

// Reserves memory for the file.
whole_file.reserve(is.tellg());

// Sets position to the start of the file.
is.seekg(0, std::ios::beg);

// Sets contents of 'whole_file' to all characters in the file.
whole_file.assign(std::istreambuf_iterator<char>(is),
 std::istreambuf_iterator<char>());
```

string ◦

`getline()`

```
std::ifstream is("foo.txt");

// The function getline returns false if there are no more lines.
for (std::string str; std::getline(is, str);) {
 // Process the line that has been read.
}
```

read()

```
std::ifstream is("foo.txt");
char str[4];

// Read 4 characters from the file.
is.read(str, 4);
```

failbit ◦ fail()

```
is.read(str, 4); // This operation might fail for any reason.

if (is.fail())
 // Failed to read!
```

◦ ofstream <<

```
std::ofstream os("foo.txt");
if(os.is_open()){
```

```
os << "Hello World!";
}
```

<< write()

```
std::ofstream os("foo.txt");
if(os.is_open()){
 char data[] = "Foo";

 // Writes 3 characters from data -> "Foo".
 os.write(data, 3);
}
```

badbit ◦ bad()

```
os << "Hello Badbit!"; // This operation might fail for any reason.
if (os.bad())
 // Failed to write!
```

◦ ◦

std::ios◦

open()

```
std::ofstream os("foo.txt", std::ios::out | std::ios::trunc);

std::ifstream is;
is.open("foo.txt", std::ios::in | std::ios::binary);
```

ios::out ios::in ios::out **iostream**◦

- ifstream - in
- ofstream out
- fstream - inout

|        |  |  |  |   |
|--------|--|--|--|---|
|        |  |  |  |   |
| app    |  |  |  | ◦ |
| binary |  |  |  | ◦ |
| in     |  |  |  | ◦ |
| out    |  |  |  | ◦ |
| trunc  |  |  |  | ◦ |
| ate    |  |  |  | ◦ |

binary/;'\n'◦

WindowsCRLF "\r\n" ◦

"\n" => "\r\n"

"\r\n" => "\n"

C++ ◦ ◦ {}

```
std::string const prepared_data = prepare_data();
{
 // Open a file for writing.
 std::ofstream output("foo.txt");

 // Write data.
 output << prepared_data;
} // The ofstream will go out of scope here.
// Its destructor will take care of closing the file properly.
```

fstreamclose()

```
// Open the file "foo.txt" for the first time.
std::ofstream output("foo.txt");

// Get some data to write from somewhere.
std::string const prepared_data = prepare_data();

// Write data to the file "foo.txt".
output << prepared_data;

// Close the file "foo.txt".
output.close();

// Preparing data might take a long time. Therefore, we don't open the output file stream
// before we actually can write some data to it.
std::string const more_prepared_data = prepare_complex_data();

// Open the file "foo.txt" for the second time once we are ready for writing.
output.open("foo.txt");

// Write the data to the file "foo.txt".
output << more_prepared_data;

// Close the file "foo.txt" once again.
output.close();
```

◦ ◦ **oder** ◦ flush() std::flush

```
std::ofstream os("foo.txt");
os << "Hello World!" << std::flush;

char data[3] = "Foo";
os.write(data, 3);
os.flush();
```

std::endl

```
// Both following lines do the same thing
```

```
os << "Hello World!\n" << std::flush;
os << "Hello world!" << std::endl;
```

◦ ◦ I/O◦

## ASCIIstd :: string

```
std::ifstream f("file.txt");

if (f)
{
 std::stringstream buffer;
 buffer << f.rdbuf();
 f.close();

 // The content of "file.txt" is available in the string `buffer.str()`
}
```

[rdbuf\(\)](#) [streambuf](#) [stringstream::operator<<](#) [member](#) [buffer](#) ◦

---

## Scott Meyers Effective STL

```
std::ifstream f("file.txt");

if (f)
{
 std::string str((std::istreambuf_iterator<char>(f)),
 std::istreambuf_iterator<char>());

 // Operations on `str`...
}
```

## STL◦

◦

```
std::ifstream f("file.txt");

if (f)
{
 f.seekg(0, std::ios::end);
 const auto size = f.tellg();

 std::string str(size, ' ');
 f.seekg(0);
 f.read(&str[0], size);
 f.close();

 // Operations on `str`...
}
```

◦

std::stringoperator>>°

```
std::ifstream file("file3.txt");

std::vector<std::string> v;

std::string s;
while(file >> s) // keep reading until we run out
{
 v.push_back(s);
}
```

operator>>“”° std::istream\_iterator“”°

```
std::ifstream file("file3.txt");

std::vector<std::string> v(std::istream_iterator<std::string>(file),
 std::istream_iterator<std::string>{});
```

std::istream\_iterator°

```
// Unfortunately there is no built in type that reads line using >>
// So here we build a simple helper class to do it. That will convert
// back to a string when used in string context.
struct Line
{
 // Store data here
 std::string data;
 // Convert object to string
 operator std::string const&() const {return data;}
 // Read a line from a stream.
 friend std::istream& operator>>(std::istream& stream, Line& line)
 {
 return std::getline(stream, line.data);
 }
};

std::ifstream file("file3.txt");

// Read the lines of a file into a container.
std::vector<std::string> v(std::istream_iterator<Line>(file),
 std::istream_iterator<Line>{});
```

‘struct’°

## C++ 11

```
struct info_type
{
 std::string name;
 int age;
 float height;

 // we define an overload of operator>> as a friend function which
```

```

// gives in privileged access to private data members
friend std::istream& operator>>(std::istream& is, info_type& info)
{
 // skip whitespace
 is >> std::ws;
 std::getline(is, info.name);
 is >> info.age;
 is >> info.height;
 return is;
}
};

void func4()
{
 auto file = std::ifstream("file4.txt");

 std::vector<info_type> v;

 for(info_type info; file >> info;) // keep reading until we run out
 {
 // we only get here if the read succeeded
 v.push_back(info);
 }

 for(auto const& info: v)
 {
 std::cout << " name: " << info.name << '\n';
 std::cout << " age: " << info.age << " years" << '\n';
 std::cout << "height: " << info.height << "lbs" << '\n';
 std::cout << '\n';
 }
}

```

## file4.txt

```

Wogger Wabbit
2
6.2
Bilbo Baggins
111
81.3
Mary Poppins
29
154.8

```

```

name: Wogger Wabbit
age: 2 years
height: 6.2lbs

name: Bilbo Baggins
age: 111 years
height: 81.3lbs

name: Mary Poppins
age: 29 years
height: 154.8lbs

```

```

std::ifstream src("source_filename", std::ios::binary);

```



```
std::ofstream dst("dest_filename", std::ios::binary);
dst << src.rdbuf();
```

## C++ 17

### C++ 17 `<filesystem>` `copy_file`

```
std::filesystem::copy_file("source_filename", "dest_filename");
```

`boost.filesystem` C++ 17 ISO C++.

`eof` true . .

```
while (!f.eof())
{
 // Everything is OK

 f >> buffer;

 // What if *only* now the eof / fail bit is set?

 /* Use `buffer` */
}
```

```
while (!f.eof())
{
 f >> buffer >> std::ws;

 if (f.fail())
 break;

 /* Use `buffer` */
}
```

```
while (f >> buffer)
{
 /* Use `buffer` */
}
```

.

- `std::ws`
- `std::basic_ios::fail` true

`std::locale` `std::basic_ios::imbue()`

- .
- .

.

UTF-8. UTF-8BOM;

```

#include <iostream>
#include <fstream>
#include <locale>

int main()
{
 std::cout << "User-preferred locale setting is "
 << std::locale("").name().c_str() << std::endl;

 // Write a floating-point value using the user's preferred locale.
 std::ofstream ofs1;
 ofs1.imbue(std::locale(""));
 ofs1.open("file1.txt");
 ofs1 << 78123.456 << std::endl;

 // Use a specific locale (names are system-dependent)
 std::ofstream ofs2;
 ofs2.imbue(std::locale("en_US.UTF-8"));
 ofs2.open("file2.txt");
 ofs2 << 78123.456 << std::endl;

 // Switch to the classic "C" locale
 std::ofstream ofs3;
 ofs3.imbue(std::locale::classic());
 ofs3.open("file3.txt");
 ofs3 << 78123.456 << std::endl;
}

```

“C”。 “C”

```

78,123.456
78,123.456
78123.456

```

```

78 123,456
78,123.456
78123.456

```

。

I / O. <https://riptutorial.com/zh-TW/cplusplus/topic/496/i---o->

# 86:

- `std::shared_ptr<ClassType> variableName = std::make_shared<ClassType>(arg1, arg2, ...);`
- `std::shared_ptr<ClassType> variableName (new ClassType(arg1, arg2, ...));`
- `std::unique_ptr<ClassType> variableName = std::make_unique<ClassType>(arg1, arg2, ...); // C++ 14`
- `std::unique_ptr<ClassType> variableName (new ClassType(arg1, arg2, ...));`

C++ newdelete ""

.

""

```
#include <memory>
```

## Examples

### std::shared\_ptr

`std::shared_ptr` `std::unique_ptr`

• `std::shared_ptr`

```
// Creation: 'firstShared' is a shared pointer for a new instance of 'Foo'
std::shared_ptr<Foo> firstShared = std::make_shared<Foo>(*args*//);
```

`shared_ptr`

```
std::shared_ptr<Foo> secondShared(firstShared); // 1st way: Copy constructing
std::shared_ptr<Foo> secondShared;
secondShared = firstShared; // 2nd way: Assigning
```

`secondSharedFoofirstShared`

• \* ->

```
secondShared->test(); // Calls Foo::test()
```

`shared_ptrFoo`

```
shared_ptrbad_alloc • shared_ptr<T> shared_ptr<T>(new T(args))T • make_shared<T>(args)
allocate_shared<T>(alloc, args)
```

### shared\_ptr[]

## C++ 11 C++ 17

make\_shared<>◦

newstd::default\_deleteshared\_ptr<>◦

### 10

```
shared_ptr<int> sh(new int[10], std::default_delete<int[]>());
```

std::default\_deletedelete[]◦

```
template<class Arr>
struct shared_array_maker {};
template<class T, std::size_t N>
struct shared_array_maker<T[N]> {
 std::shared_ptr<T> operator() const {
 auto r = std::make_shared<std::array<T,N>>();
 if (!r) return {};
 return {r.data(), r};
 }
};
template<class Arr>
auto make_shared_array()
-> decltype(shared_array_maker<Arr>{}())
{ return shared_array_maker<Arr>{}(); }
```

make\_shared\_array<int[10]>◦ 10shared\_ptr<int>◦

## C++ 17

C++ 17 shared\_ptr ◦ array-deleter[]

```
std::shared_ptr<int[]> sh(new int[10]);
sh[0] = 42;
```

```
struct Foo { int x; };
std::shared_ptr<Foo> p1 = std::make_shared<Foo>();
std::shared_ptr<int> p2(p1, &p1->x);
```

p2p1Foop2intx ◦ p1Foop2◦

shared\_ptrshared\_ptr ◦ Fooshared\_ptr

```
Foo *foo = new Foo;
std::shared_ptr<Foo> shared1(foo);
std::shared_ptr<Foo> shared2(foo); // don't do this

shared1.reset(); // this will delete foo, since shared1
 // was the only shared_ptr that owned it
```

```
shared2->test(); // UNDEFINED BEHAVIOR: shared2's foo has been
 // deleted already!!
```

## shared\_ptr

shared\_ptr◦ std::move

```
shared_ptr<int> up = make_shared<int>();
// Transferring the ownership
shared_ptr<int> up2 = move(up);
// At this point, the reference count of up = 0 and the
// ownership of the pointer is solely with up2 with reference count = 1
```

## std :: weak\_ptr

std::weak\_ptrstd::shared\_ptr◦ ◦

---

std::weak\_ptr

```
#include <memory>
#include <vector>

struct TreeNode {
 std::weak_ptr<TreeNode> parent;
 std::vector< std::shared_ptr<TreeNode> > children;
};

int main() {
 // Create a TreeNode to serve as the root/parent.
 std::shared_ptr<TreeNode> root(new TreeNode);

 // Give the parent 100 child nodes.
 for (size_t i = 0; i < 100; ++i) {
 std::shared_ptr<TreeNode> child(new TreeNode);
 root->children.push_back(child);
 child->parent = root;
 }

 // Reset the root shared pointer, destroying the root object, and
 // subsequently its child nodes.
 root.reset();
}
```

std::weak\_ptrparent◦ parent◦ main()◦ std::shared\_ptrchildren◦

shared\_ptrshared\_ptrweak\_ptr◦

```
#include <memory>
int main()
{
 {
 std::weak_ptr<int> wk;
 {
```

```

 // std::make_shared is optimized by allocating only once
 // while std::shared_ptr<int>(new int(42)) allocates twice.
 // Drawback of std::make_shared is that control block is tied to our integer
 std::shared_ptr<int> sh = std::make_shared<int>(42);
 wk = sh;
 // sh memory should be released at this point...
 }
 // ... but wk is still alive and needs access to control block
}
// now memory is released (sh and wk)
}

```

`std::weak_ptr` `std::weak_ptr::lock()` `std::shared_ptr`

```

#include <cassert>
#include <memory>
int main()
{
 {
 std::weak_ptr<int> wk;
 std::shared_ptr<int> sp;
 {
 std::shared_ptr<int> sh = std::make_shared<int>(42);
 wk = sh;
 // calling lock will create a shared_ptr to the object referenced by wk
 sp = wk.lock();
 // sh will be destroyed after this point, but sp is still alive
 }
 // sp still keeps the data alive.
 // At this point we could even call lock() again
 // to retrieve another shared_ptr to the same data from wk
 assert(*sp == 42);
 assert(!wk.expired());
 // resetting sp will delete the data,
 // as it is currently the last shared_ptr with ownership
 sp.reset();
 // attempting to lock wk now will return an empty shared_ptr,
 // as the data has already been deleted
 sp = wk.lock();
 assert(!sp);
 assert(wk.expired());
 }
}

```

## std :: unique\_ptr

### C++ 11

`std::unique_ptr` `std::shared_ptr` `std::weak_ptr`

```

// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);

```

**C++ 11** `std::unique_ptr` **C++ 14** `std::make_unique`

ptrint° °

std::unique\_ptrstd::make\_unique

```
// Creates a unique_ptr to an int with value 59
std::unique_ptr<int> ptr = std::make_unique<int>(59);

// Creates a unique_ptr to an array of 15 ints
std::unique_ptr<int[]> ptr = std::make_unique<int[]>(15);
```

std::unique\_ptr°

---

std::movenullptr°

```
// 1. std::unique_ptr
std::unique_ptr<int> ptr = std::make_unique<int>();

// Change value to 1
*ptr = 1;

// 2. std::unique_ptr (by moving 'ptr' to 'ptr2', 'ptr' doesn't own the object anymore)
std::unique_ptr<int> ptr2 = std::move(ptr);

int a = *ptr2; // 'a' is 1
int b = *ptr; // undefined behavior! 'ptr' is 'nullptr'
 // (because of the move command above)
```

unique\_ptr

```
void foo(std::unique_ptr<int> ptr)
{
 // Your code goes here
}

std::unique_ptr<int> ptr = std::make_unique<int>(59);
foo(std::move(ptr))
```

unique\_ptr° **C++ 11**unique\_ptr°

```
std::unique_ptr<int> foo()
{
 std::unique_ptr<int> ptr = std::make_unique<int>(59);
 return ptr;
}

std::unique_ptr<int> ptr = foo();
```

```
int* foo_cpp03();

int* p = foo_cpp03(); // do I own p? do I have to delete it at some point?
 // it's not readily apparent what the answer is.
```

**C++ 14**

## C++ 14 `make_unique` ◦ C++ 11

```
template<typename T, typename... Args>
typename std::enable_if<!std::is_array<T>::value, std::unique_ptr<T>>::type
make_unique(Args&&... args)
{ return std::unique_ptr<T>(new T(std::forward<Args>(args)...)); }

// Use make_unique for arrays
template<typename T>
typename std::enable_if<std::is_array<T>::value, std::unique_ptr<T>>::type
make_unique(size_t n)
{ return std::unique_ptr<T>(new typename std::remove_extent<T>::type[n]()); }
```

## C++ 11

`std::auto_ptr` `unique_ptr` `std::vector` ◦ ◦ 10 `int[]` `int`

```
std::unique_ptr<int[]> arr_ptr = std::make_unique<int[]>(10);
```

```
auto arr_ptr = std::make_unique<int[]>(10);
```

`arr_ptr`

```
arr_ptr[2] = 10; // Modify third element
```

◦ ◦ `unique_ptr` `vector` ◦ ◦

C++ 11 `std::auto_ptr` ◦ `unique_ptr` `auto_ptr` `ptr` ◦

## C

### CSDL2 ◦

```
std::unique_ptr<SDL_Surface> a; // won't work, UNSAFE!
```

◦ `SDL_Surface` `SDL_FreeSurface` () C ◦

```
std::unique_ptr<SDL_Surface, void(*) (SDL_Surface*)> a(pointer, SDL_FreeSurface);
```

`operator()`

```
struct SurfaceDeleter {
 void operator()(SDL_Surface* surf) {
 SDL_FreeSurface(surf);
 }
};

std::unique_ptr<SDL_Surface, SurfaceDeleter> a(pointer, SurfaceDeleter{}); // safe
std::unique_ptr<SDL_Surface, SurfaceDeleter> b(pointer); // equivalent to the above
// as the deleter is value-
initialized
```



[unique\\_ptr](#) ◦

```
unique_ptr◦ std::unique_ptr<SDL_Surface, SurfaceDeleter>std::unique_ptr<SDL_Surface,
void(*) (SDL_Surface*)> SDL_Surface*◦
```

[shared\\_ptr](#)[unique\\_ptr](#) ◦ [shared\\_ptr](#)**API**◦ [shared\\_ptr](#)[unique\\_ptr](#)◦

```
// deleter required at construction time and is part of the type
std::unique_ptr<SDL_Surface, void(*) (SDL_Surface*)> a(pointer, SDL_FreeSurface);

// deleter is only required at construction time, not part of the type
std::shared_ptr<SDL_Surface> b(pointer, SDL_FreeSurface);
```

## C ++ 17

template auto

```
template <auto DeleteFn>
struct FunctionDeleter {
 template <class T>
 void operator()(T* ptr) {
 DeleteFn(ptr);
 }
};

template <class T, auto DeleteFn>
using unique_ptr_deleter = std::unique_ptr<T, FunctionDeleter<DeleteFn>>;
```

```
unique_ptr_deleter<SDL_Surface, SDL_FreeSurface> c(pointer);
```

autovoid SDL\_FreeSurface fclose ◦

## auto\_ptr

## C ++ 11

std::auto\_ptr**C ++ 11****C ++ 17**◦ **C ++ 03**◦ std::move**unique\_ptr**std::auto\_ptr◦

std::unique\_ptrstd::auto\_ptr ◦ std::auto\_ptr◦

std::auto\_ptr**RAII**

```
{
 std::auto_ptr<int> p(new int(42));
 std::cout << *p;
} // p is deleted here, no memory leaked
```

```
std::auto_ptr<X> px = ...;
std::auto_ptr<X> py = px;
// px is now empty
```

## std::auto\_ptr

```
void f(std::auto_ptr<X>) {
 // assumes ownership of X
 // deletes it at end of scope
};

std::auto_ptr<X> px = ...;
f(px); // f acquires ownership of underlying X
 // px is now empty
px->foo(); // NPE!
// px.~auto_ptr() does NOT delete
```

“”  
◦ auto\_ptrconst◦

```
template <typename T>
class auto_ptr {
 T* ptr;
public:
 auto_ptr(auto_ptr& rhs)
 : ptr(rhs.release())
 { }

 auto_ptr& operator=(auto_ptr& rhs) {
 reset(rhs.release());
 return *this;
 }

 T* release() {
 T* tmp = ptr;
 ptr = nullptr;
 return tmp;
 }

 void reset(T* tmp = nullptr) {
 if (ptr != tmp) {
 delete ptr;
 ptr = tmp;
 }
 }

 /* other functions ... */
};
```

◦ T

```
T a = ...;
T b(a);
assert(b == a);
```

auto\_ptr◦ auto\_ptr◦

## shared\_ptr

enable\_shared\_from\_this◦

enable\_shared\_from\_this shared\_from\_this shared\_ptr this ◦

shared\_ptr

```
#include <memory>
class A: public enable_shared_from_this<A> {
};
A* ap1 =new A();
shared_ptr<A> ap2(ap1); // First prepare a shared pointer to the object and hold it!
// Then get a shared pointer to the object from the object itself
shared_ptr<A> ap3 = ap1->shared_from_this();
int c3 =ap3.use_count(); // =2: pointing to the same object
```

2enable\_shared\_from\_this ◦

```
#include <memory> // enable_shared_from_this

class Widget : public std::enable_shared_from_this< Widget >
{
public:
 void DoSomething()
 {
 std::shared_ptr< Widget > self = shared_from_this();
 someEvent -> Register(self);
 }
private:
 ...
};

int main()
{
 ...
 auto w = std::make_shared< Widget >();
 w -> DoSomething();
 ...
}
```

shared\_ptr shared\_from\_this() shared\_from\_this() ◦ C++ 17 std::bad\_alloc ◦

shared\_from\_this() shared\_ptr shared\_ptr ◦

## std :: shared\_ptr

std::shared\_ptr static\_cast const\_cast dynamic\_cast reinterpret\_cast ◦ std::static\_pointer\_cast  
std::const\_pointer\_cast std::dynamic\_pointer\_cast std::const\_pointer\_cast  
std::dynamic\_pointer\_cast std::reinterpret\_pointer\_cast

```
struct Base { virtual ~Base() noexcept {} };
struct Derived: Base {};
auto derivedPtr(std::make_shared<Derived>());
auto basePtr(std::static_pointer_cast<Base>(derivedPtr));
auto constBasePtr(std::const_pointer_cast<Base const>(basePtr));
auto constDerivedPtr(std::dynamic_pointer_cast<Derived const>(constBasePtr));
```

std::reinterpret\_pointer\_cast

```
template <typename To, typename From>
inline std::shared_ptr<To> reinterpret_pointer_cast(
 std::shared_ptr<From> const & ptr) noexcept
{ return std::shared_ptr<To>(ptr, reinterpret_cast<To *>(ptr.get())); }
```

## value\_ptr

value\_ptr ◦ ◦

```
// Like std::default_delete:
template<class T>
struct default_copier {
 // a copier must handle a null T const* in and return null:
 T* operator()(T const* tin) const {
 if (!tin) return nullptr;
 return new T(*tin);
 }
 void operator()(void* dest, T const* tin) const {
 if (!tin) return;
 return new(dest) T(*tin);
 }
};
// tag class to handle empty case:
struct empty_ptr_t {};
constexpr empty_ptr_t empty_ptr{};
// the value pointer type itself:
template<class T, class Copier=default_copier<T>, class Deleter=std::default_delete<T>,
 class Base=std::unique_ptr<T, Deleter>
>
struct value_ptr:Base, private Copier {
 using copier_type=Copier;
 // also typedefs from unique_ptr

 using Base::Base;

 value_ptr(T const& t):
 Base(std::make_unique<T>(t)),
 Copier()
 {}
 value_ptr(T && t):
 Base(std::make_unique<T>(std::move(t))),
 Copier()
 {}
 // almost-never-empty:
 value_ptr():
 Base(std::make_unique<T>()),
 Copier()
 {}
 value_ptr(empty_ptr_t) {}

 value_ptr(Base b, Copier c={}):
 Base(std::move(b)),
 Copier(std::move(c))
 {}

 Copier const& get_copier() const {
```

```

 return *this;
}

value_ptr clone() const {
 return {
 Base(
 get_copier()(this->get()),
 this->get_deleter()
),
 get_copier()
 };
}
value_ptr(value_ptr&&)=default;
value_ptr& operator=(value_ptr&&)=default;

value_ptr(value_ptr const& o):value_ptr(o.clone()) {}
value_ptr& operator=(value_ptr const&o) {
 if (o && *this) {
 // if we are both non-null, assign contents:
 **this = *o;
 } else {
 // otherwise, assign a clone (which could itself be null):
 *this = o.clone();
 }
 return *this;
}
value_ptr& operator=(T const& t) {
 if (*this) {
 **this = t;
 } else {
 *this = value_ptr(t);
 }
 return *this;
}
value_ptr& operator=(T && t) {
 if (*this) {
 **this = std::move(t);
 } else {
 *this = value_ptr(std::move(t));
 }
 return *this;
}
T& get() { return **this; }
T const& get() const { return **this; }
T* get_pointer() {
 if (!*this) return nullptr;
 return std::addressof(get());
}
T const* get_pointer() const {
 if (!*this) return nullptr;
 return std::addressof(get());
}
// operator-> from unique_ptr
};
template<class T, class...Args>
value_ptr<T> make_value_ptr(Args&&... args) {
 return {std::make_unique<T>(std::forward<Args>(args)...) };
}

```

empty\_ptr\_t value\_ptr ◦ unique\_ptr explicit operator bool() const ◦ .get().get\_pointer() ◦

pImplpImpl ◦

Copier ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/509/>

## 87:

- ◦
- ◦

## Examples

- 

```
// Forward declaration of functions.
void friend_function();
void non_friend_function();

class PrivateHolder {
public:
 PrivateHolder(int val) : private_value(val) {}
private:
 int private_value;
 // Declare one of the function as a friend.
 friend void friend_function();
};

void non_friend_function() {
 PrivateHolder ph(10);
 // Compilation error: private_value is private.
 std::cout << ph.private_value << std::endl;
}

void friend_function() {
 // OK: friends may access private values.
 PrivateHolder ph(10);
 std::cout << ph.private_value << std::endl;
}
```

- ◦
- PrivateHolderPrivateHolder

```
class PrivateHolderDerived : public PrivateHolder {
public:
 PrivateHolderDerived(int val) : PrivateHolder(val) {}
private:
 int derived_private_value = 0;
};
```

```
void friend_function() {
 PrivateHolderDerived pd(20);
 // OK.
 std::cout << pd.private_value << std::endl;
 // Compilation error: derived_private_value is private.
 std::cout << pd.derived_private_value << std::endl;
}
```

PrivateHolderDerivedPrivateHolder::private\_value ◦

```
class Accesser {
public:
 void private_accesser();
};

class PrivateHolder {
public:
 PrivateHolder(int val) : private_value(val) {}
 friend void Accesser::private_accesser();
private:
 int private_value;
};

void Accesser::private_accesser() {
 PrivateHolder ph(10);
 // OK: this method is declares as friend.
 std::cout << ph.private_value << std::endl;
}
```

◦

```
class Accesser {
public:
 void private_accesser1();
 void private_accesser2();
};

class PrivateHolder {
public:
 PrivateHolder(int val) : private_value(val) {}
 friend class Accesser;
private:
 int private_value;
};

void Accesser::private_accesser1() {
 PrivateHolder ph(10);
 // OK.
 std::cout << ph.private_value << std::endl;
}

void Accesser::private_accesser2() {
 PrivateHolder ph(10);
 // OK.
 std::cout << ph.private_value + 1 << std::endl;
}
```

◦ ◦

```
class Accesser {
public:
 void private_accesser1();
 void private_accesser2();
private:
 int private_value = 0;
};
```



```
class PrivateHolder {
public:
 PrivateHolder(int val) : private_value(val) {}
 // Accesser is a friend of PrivateHolder
 friend class Accesser;
 void reverse_acesse() {
 // but PrivateHolder cannot access Accesser's members.
 Accesser a;
 std::cout << a.private_value;
 }
private:
 int private_value;
};
```

<https://riptutorial.com/zh-TW/cplusplus/topic/3275/>

# 88:

## PromisesFutures

`std::promise`

`std::future`

## Examples

### `std::future` `std::promise`

```
{
 auto promise = std::promise<std::string>();

 auto producer = std::thread([&
 {
 promise.set_value("Hello World");
 }]);

 auto future = promise.get_future();

 auto consumer = std::thread([&
 {
 std::cout << future.get();
 }]);

 producer.join();
 consumer.join();
}
```

`std::async` `std::deferred` `std::async` `std::future` `std::future`

```
template<typename F>
auto async_deferred(F&& func) -> std::future<decltype(func())>
{
 using result_type = decltype(func());

 auto promise = std::promise<result_type>();
 auto future = promise.get_future();

 std::thread(std::bind([=] (std::promise<result_type>& promise)
 {
 try
 {
 promise.set_value(func());
 // Note: Will not work with std::promise<void>. Needs some meta-template
 programming which is out of scope for this example.
 }
 catch(...)
 {
 promise.set_exception(std::current_exception());
 }
 }, std::move(promise))).detach();
}
```

```
 return future;
}
```

## std::packaged\_taskstd::future

std::packaged\_task

```
template<typename F>
auto async_deferred(F&& func) -> std::future<decltype(func())>
{
 auto task = std::packaged_task<decltype(func())()>(std::forward<F>(func));
 auto future = task.get_future();

 std::thread(std::move(task)).detach();

 return std::move(future);
}
```

◦ ◦ std::thread◦

std::asyncstd::future◦

## std::future\_errorstd::future\_errc

std::promisestd::futurestd::future\_error◦

std::future\_errc

```
enum class future_errc {
 broken_promise = /* the task is no longer shared */,
 future_already_retrieved = /* the answer was already retrieved */,
 promise_already_satisfied = /* the answer was stored already */,
 no_state = /* access to a promise in non-shared state */
};
```

```
int test()
{
 std::promise<int> pr;
 return 0; // returns ok
}
```

```
int test()
{
 std::promise<int> pr;
 auto fut = pr.get_future(); //blocks indefinitely!
 return 0;
}
```

```
int test()
{
 std::promise<int> pr;
 auto fut1 = pr.get_future();
}
```

```

try{
 auto fut2 = pr.get_future(); // second attempt to get future
 return 0;
}
catch(const std::future_error& e)
{
 cout << e.what() << endl; // Error: "The future has already been retrieved
from the promise or packaged_task."
 return -1;
}
return fut2.get();
}

```

## std :: promise

```

int test()
{
 std::promise<int> pr;
 auto fut = pr.get_future();
 try{
 std::promise<int> pr2(std::move(pr));
 pr2.set_value(10);
 pr2.set_value(10); // second attempt to set promise throws exception
 }
 catch(const std::future_error& e)
 {
 cout << e.what() << endl; // Error: "The state of the promise has already been
set."
 return -1;
 }
 return fut.get();
}

```

## std :: futurestd :: async

std::asyncmerge\_sort<sup>o</sup> std::future

```

#include <iostream>
using namespace std;

void merge(int low,int mid,int high, vector<int>&num)
{
 vector<int> copy(num.size());
 int h,i,j,k;
 h=low;
 i=low;
 j=mid+1;

 while((h<=mid)&&(j<=high))
 {
 if(num[h]<=num[j])
 {
 copy[i]=num[h];
 h++;
 }
 else

```

```

 {
 copy[i]=num[j];
 j++;
 }
 i++;
 }
 if(h>mid)
 {
 for(k=j;k<=high;k++)
 {
 copy[i]=num[k];
 i++;
 }
 }
 else
 {
 for(k=h;k<=mid;k++)
 {
 copy[i]=num[k];
 i++;
 }
 }
 for(k=low;k<=high;k++)
 swap(num[k], copy[k]);
}

void merge_sort(int low,int high,vector<int>& num)
{
 int mid;
 if(low<high)
 {
 mid = low + (high-low)/2;
 auto future1 = std::async(std::launch::deferred, [&]()
 {
 merge_sort(low,mid,num);
 });
 auto future2 = std::async(std::launch::deferred, [&]()
 {
 merge_sort(mid+1,high,num);
 });

 future1.get();
 future2.get();
 merge(low,mid,high,num);
 }
}

```

std::launch\_deferred std::async ◦ ◦ std::async std::future::get () ◦

std::launch\_async ◦

std::launch::deferred | std::launch::async ◦

- std :: async ◦
- std :: future ◦
- std :: promise ◦
- std :: packaged\_task ◦



# 89:

## Examples

o

```
struct {
 int foo;
 double bar;
} foobar;

foobar.foo = 5;
foobar.bar = 4.0;

class {
 int baz;
public:
 int buzz;

 void setBaz(int v) {
 baz = v;
 }
} barbar;

barbar.setBaz(15);
barbar.buzz = 2;
```

## C++

```
struct Example {
 struct {
 int inner_b;
 };

 int outer_b;

 //The anonymous struct's members are accessed as if members of the parent struct
 Example() : inner_b(2), outer_b(4) {
 inner_b = outer_b + 2;
 }
};

Example ex;

//The same holds true for external code referencing the struct
ex.inner_b -= ex.outer_b;
```

typedefusing

## C++ 11

```
using vec2d = struct {
 float x;
 float y;
```

```
};
```

```
typedef struct {
 float x;
 float y;
} vec2d;
```

```
vec2d pt;
pt.x = 4.f;
pt.y = 3.f;
```

- “struct”。

```
struct Sample {
 union {
 int a;
 int b;
 };
 int c;
};
int main()
{
 Sample sa;
 sa.a =3;
 sa.b =4;
 sa.c =5;
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2704/>



# 90:

UBISO C ++§1.3.24N4296”。

UB。

UB。

C ++。

- ◦
- ◦
- ◦
- ◦

◦

◦

C ++“”。

- ◦
- gccclang“Undefined Behavior Sanitizer” -fsanitize=undefined ◦
- lint◦

C ++ 14ISO / IEC 1488220141.9

1. ◦ []
2. sizeof(int) ◦ ◦ ◦ []
3. ◦ ◦ ◦ ◦
4. const◦ [ ◦ - ]

## Examples

```
int *ptr = nullptr;
*ptr = 1; // Undefined behavior
```

\*ptr◦

◦

**voidreturn**

voidreturn

◦

```
int function() {
 // Missing return statement
}

int main() {
 function(); //Undefined Behavior
}
```

◦

```
main◦ mainreturnreturn 0;◦
```

## C ++ 11

```
char *str = "hello world";
str[0] = 'H';
```

"hello world"◦

strC ++ 03◦ 2003◦ 2003◦

## C ++ 11

C ++ 11◦

```
char *str = const_cast<char *>("hello world");
str[0] = 'H';
```

```
int array[] = {1, 2, 3, 4, 5};
array[5] = 0; // Undefined behavior
```

array + 5 ◦

```
const int *end = array + 5; // Pointer to one past the last index
for (int *p = array; p != end; ++p)
 // Do something with `p`
```

◦ ◦

```
int x = 5 / 0; // Undefined behavior
```

0◦

```
float x = 5.0f / 0.0f; // x is +infinity
```

IEEE-754NaN 0.0f infinity -infinity ◦

```
int x = INT_MAX + 1;

// x can be anything -> Undefined behavior
```

◦

## C++ 115/4

◦

```
unsigned int x = UINT_MAX + 1;

// x is 0
```

$2^{nn}$

## C++ 113.9.1/4

```
signed int x ;
if(x > x + 1)
{
 //do something
}
```

“if”

```
int a;
std::cout << a; // Undefined behavior!
```

a◦

“”◦ a◦ “”◦

◦ - ◦

## C++ 14

unsigned char

- ;
- ;
- unsigned char;
- unsigned char;
- unsigned char;

◦ ◦

static

```
static int a;
```

```
std::cout << a; // Defined behavior, 'a' is 0
```

## ODR.

foo.h

```
class Foo {
public:
 double x;
private:
 int y;
};

Foo get_foo();
```

foo.cpp

```
#include "foo.h"
Foo get_foo() { /* implementation */ }
```

main.cpp

```
// I want access to the private member, so I am going to replace Foo with my own type
class Foo {
public:
 double x;
 int y;
};
Foo get_foo(); // declare this function ourselves since we aren't including foo.h
int main() {
 Foo foo = get_foo();
 // do something with foo.y
}
```

**class** ::Foo° °

newdelete° newdelete°

newdelete[]° delete[]new°

mallocfree°

```
int* p1 = new int;
delete p1; // correct
// delete[] p1; // undefined
// free(p1); // undefined

int* p2 = new int[10];
delete[] p2; // correct
// delete p2; // undefined
// free(p2); // undefined

int* p3 = static_cast<int*>(malloc(sizeof(int)));
free(p3); // correct
```

```
// delete p3; // undefined
// delete[] p3; // undefined
```

**C++** mallocfree newdelete std::vectorstd::stringraw newdelete[]

**CV**

```
float x = 42;
int y = reinterpret_cast<int*>(x);
```

- - ◦
  - charunsigned charchar
  - ◦
- ◦

```
struct Base {
};
struct Derived : Base {
 void f() {}
};
struct Unrelated {};
Unrelated u;
Derived& r1 = reinterpret_cast<Derived*>(u); // ok
r1.f(); // UB
Base b;
Derived& r2 = reinterpret_cast<Derived*>(b); // ok
r2.f(); // UB
```

**C++IEEE 754**

```
float x = 1.0;
for (int i = 0; i < 10000; i++) {
 x *= 10.0; // will probably overflow eventually; undefined behavior
}
```

## 10.4

;10.3

**C++Scott Meyers destructors**

```
class transaction
{
public:
 transaction(){ log_it(); }
 virtual void log_it() const = 0;
};

class sell_transaction : public transaction
```

```
{
public:
 virtual void log_it() const { /* Do something */ }
};
```

sell\_transaction

```
sell_transaction s;
```

sell\_transaction sell\_transaction transaction ◦ transaction sell\_transaction transaction ◦

transaction::transaction() log\_it - sell\_transaction::log\_it ◦

- log\_it ◦
- log\_it transaction::log\_it ◦

◦

```
class base { };
class derived: public base { };

int main() {
 base* p = new derived();
 delete p; // This is undefined behavior!
}
```

[expr.delete]§5.3.5/3 virtual delete

◦

◦

◦ ◦

```
#include <iostream>
int& getX() {
 int x = 42;
 return x;
}
int main() {
 int& r = getX();
 std::cout << r << "\n";
}
```

getX ◦ ◦ r ◦ 42 ◦

std::posix

17.6.4.2.1 / 1 std

## C++std::std.

posix 17.6.4.2.2 / 1

## C++posix::posix.

```
#include <algorithm>

namespace std
{
 int foo(){}
}
```

algorithm ◦

◦ ◦ ◦

```
class foo
{
 // Stuff
};
```

```
namespace std
{
 template<>
 struct hash<foo>
 {
 public:
 size_t operator()(const foo &f) const;
 };
}
```

- 
- 
- 

◦

```
double x = 1e100;
int y = x; // int probably cannot hold numbers that large, so this is UB
```

static\_cast resp. resp. reference. ◦

◦ ◦

```
int f();
void (*p)() = reinterpret_cast<void(*)>(f);
p(); // undefined
```

## const

const

◦ const constconst ◦ constmutableconst ◦

const\_cast

```
const int x = 123;
const_cast<int&>(x) = 456;
std::cout << x << '\n';
```

const int123 ◦ const ◦

```
#include <iostream>

class Foo* instance;

class Foo {
public:
 int get_x() const { return m_x; }
 void set_x(int x) { m_x = x; }
private:
 Foo(int x, Foo*& this_ref): m_x(x) {
 this_ref = this;
 }
 int m_x;
 friend const Foo& getFoo();
};

const Foo& getFoo() {
 static const Foo foo(123, instance);
 return foo;
}

void do_evil(int x) {
 instance->set_x(x);
}

int main() {
 const Foo& foo = getFoo();
 do_evil(456);
 std::cout << foo.get_x() << '\n';
}
```

getFooconst Foo m\_x123 ◦ do\_evil foo.m\_x456.

do\_evil; Foo\* setter ◦ const\_cast const Foo ◦ Foo ◦ constconst const Foo\* thisFoo\* const Foo\* ◦

◦

◦ static\_cast ◦

```
struct Base { int x; };
struct Derived : Base { int y; };
int Derived::*pdy = &Derived::y;
int Base::*pby = static_cast<int Base::*>(pdy);

Base* b1 = new Derived;
b1->*pby = 42; // ok; sets y in Derived object to 42
```



```
Base* b2 = new Base;
b2->*pby = 42; // undefined; there is no y member in Base
```

static\_castTD::\*TB::\* B◦ ◦

- ◦

```
int a[10];
int* p1 = &a[5];
int* p2 = p1 + 4; // ok; p2 points to a[9]
int* p3 = p1 + 5; // ok; p2 points to one past the end of a
int* p4 = p1 + 6; // UB
int* p5 = p1 - 5; // ok; p2 points to a[0]
int* p6 = p1 - 6; // UB
int* p7 = p3 - 5; // ok; p7 points to a[5]
```

- ◦ 0◦

```
int a[10];
int b[10];
int *p1 = &a[8], *p2 = &a[3];
int d1 = p1 - p2; // yields 5
int *p3 = p1 + 2; // ok; p3 points to one past the end of a
int d2 = p3 - p2; // yields 7
int *p4 = &b[0];
int d3 = p4 - p1; // UB
```

- std::ptrdiff\_t◦

- cv-qualification◦ “[◦ ”

```
struct Base { int x; };
struct Derived : Base { int y; };
Derived a[10];
Base* p1 = &a[1]; // ok
Base* p2 = p1 + 1; // UB; p1 points to Derived
Base* p3 = p1 - 1; // likewise
Base* p4 = &a[2]; // ok
auto p5 = p4 - p1; // UB; p4 and p1 point to Derived
const Derived* p6 = &a[1];
const Derived* p7 = p6 + 1; // ok; cv-qualifiers don't matter
```

◦ ◦

```
const int a = 42;
const int b = a << -1; // UB
const int c = a << 0; // ok
const int d = a << 32; // UB if int is 32 bits or less
const int e = a >> 32; // also UB if int is 32 bits or less
const signed char f = 'x';
const int g = f << 10; // ok even if signed char is 10 bits or less;
// int must be at least 16 bits
```

## [[noreturn]]

### C++ 11

#### [dcl.attr.noreturn]

```
[[noreturn]] void f() {
 throw "error"; // OK
}
[[noreturn]] void g(int i) { // behavior is undefined if called with an argument <= 0
 if (i > 0)
 throw "positive";
}
```

◦

```
struct S {
 ~S() { std::cout << "destroying S\n"; }
};
int main() {
 S s;
 s.~S();
} // UB: s destroyed a second time here
```

std::unique\_ptr<T>T ◦

```
void f(std::unique_ptr<S> p);
int main() {
 S s;
 std::unique_ptr<S> p(&s);
 f(std::move(p)); // s destroyed upon return from f
} // UB: s destroyed
```

shared\_ptr◦

```
void f(std::shared_ptr<S> p1, std::shared_ptr<S> p2);
int main() {
 S* p = new S;
 // I want to pass the same object twice...
 std::shared_ptr<S> sp1(p);
 std::shared_ptr<S> sp2(p);
 f(sp1, sp2);
} // UB: both sp1 and sp2 will destroy s separately
// NB: this is correct:
// std::shared_ptr<S> sp(p);
// f(sp, sp);
```

#### [temp.inst] / 17

```
template<class T> class X {
 X<T>* p; // OK
 X<T*> a; // implicit generation of X<T> requires
 // the implicit instantiation of X<T*> which requires
```

```
};
 // the implicit instantiation of X<T**> which ...
```

<https://riptutorial.com/zh-TW/cplusplus/topic/1812/>

# 91:

◦ ◦

## Examples

### TU

◦

- Foo.cpp

```
#include <iostream>

int dummyFoo = ((std::cout << "foo"), 0);
```

- bar.cpp

```
#include <iostream>

int dummyBar = ((std::cout << "bar"), 0);
```

- main.cpp

```
int main() {}
```

```
foobar
```

```
barfoo
```

### *Fiasco* ◦

◦

```
enum class E {
 X = 1,
 Y = 1000,
};
// assume 1000 does not fit into a char
char c1 = static_cast<char>(E::X); // c1 is 1
char c2 = static_cast<char>(E::Y); // c2 has an unspecified value
```

◦

```
enum Color {
 RED = 1,
 GREEN = 2,
```

```

 BLUE = 3,
};
Color c = static_cast<Color>(4);

```

```

enum Scale {
 ONE = 1,
 TWO = 2,
 FOUR = 4,
};
Scale s = static_cast<Scale>(3);

```

s3ONE TWOFOUR ◦

\*

void\*T\*T◦

```

// Suppose that alignof(int) is 4
int x = 42;
void* p1 = &x;
// Do some pointer arithmetic...
void* p2 = static_cast<char*>(p1) + 2;
int* p3 = static_cast<int*>(p2);

```

p3p2int;◦

## reinterpret\_cast

reinterpret\_cast◦

```

int f();
auto fp = reinterpret_cast<int(*)>(int>(&f)); // fp has unspecified value

```

## C ++ 03

reinterpret\_cast◦

```

int x = 42;
char* p = reinterpret_cast<char*>>(&x); // p has unspecified value

```

static\_cast<char\*>(static\_cast<void\*>>(&x))px◦ C ++ 11◦ ◦

< > <=>=

- 1◦

```

int x;
int y;
const bool b1 = &x < &y; // unspecified
int a[10];
const bool b2 = &a[0] < &a[1]; // true

```

```
const bool b3 = &a[0] < &x; // unspecified
const bool b4 = (a + 9) < (a + 10); // true
 // note: a+10 points past the end of the array
```

- 

```
class A {
public:
 int x;
 int y;
 bool f1() { return &x < &y; } // true; x comes before y
 bool f2() { return &x < &z; } // unspecified
private:
 int z;
};
```

- 

- sizeof◦
- ◦
- ◦
- offsetof◦
- ◦

- 

```
void f() {
 int x;
 int& r = x;
 // do something with r
}
```

rxfrx r ◦

- x = 1, y = 2x = 2, y = 1◦

```
int f(int x, int y) {
 printf("x = %d, y = %d\n", x, y);
}
int get_val() {
 static int x = 0;
 return ++x;
}
int main() {
 f(get_val(), get_val());
}
```

## C++ 17

## C++ 17◦

-

```

struct from_int {
 from_int(int x) { std::cout << "from_int (" << x << ")\n"; }
};
int make_int(int x){ std::cout << "make_int (" << x << ")\n"; return x; }

void foo(from_int a, from_int b) {
}
void bar(from_int a, from_int b) {
}

auto which_func(bool b){
 std::cout << b?"foo":"bar" << "\n";
 return b?foo:bar;
}

int main(int argc, char const*const* argv) {
 which_func(true)(make_int(1), make_int(2));
}

```

```

bar
make_int(1)
from_int(1)
make_int(2)
from_int(2)

```

```

bar
make_int(2)
from_int(2)
make_int(1)
from_int(1)

```

barmakefrom

```

bar
make_int(2)
make_int(1)
from_int(2)
from_int(1)

```

◦ `make_int` **C++ 17** `bar` `from_int` `make_int` `from_int` ◦

## C++ 11

◦ `v2{1, 2, 3, 4}` `v1` ◦

```

int main() {
 std::vector<int> v1{1, 2, 3, 4};
 std::vector<int> v2 = std::move(v1);
}

```

◦ `std::unique_ptr<T>` `null` ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/4939/>

# 92:

CC ++。 C ++。 。

C ++。 /GNU / LinuxGNU MakeVisual C ++ / Visual StudioNMAKE。

IDEIDE。 IDE/CMake for EclipseMicrosoft Visual Studio 2012。

## Examples

### CMake

CMakeIDE。 CMake“Hello World”C ++ 。

CMake“CMakeLists.txt”。 CMakeLists.txt

```
cmake_minimum_required(VERSION 2.4)

project(HelloWorld)

add_executable(HelloWorld main.cpp)
```

Coliru 。

CMakemain.cpp“HelloWorld”。

/ IDE

```
> cmake .
```

```
> cmake --build .
```

。 “out-of-source”

```
> mkdir build
> cd build
> cmake ..
> cmake --build .
```

### CMakeshell

```
> cmake -E make_directory build
> cmake -E chdir build cmake ..
> cmake --build build
```

CMakeIDE 。

[Visual Studio](#)`nmake`makefile



```
> cmake -G "NMake Makefiles" ..
> nmake
```

## GNU make

---

GNU Make<sub>make shell</sub>。 GNU Make<sub>Make</sub>。 Unix<sub>POSIX</sub>Linux<sub>Mac OS X</sub>BSD。

GNU Make<sub>GNU</sub>GNU / Linux。 GNU Make<sub>Windows</sub>Mac OS X。 。 GNU Make<sub>CC ++</sub>。

---

make<sub>Makefile</sub>。 Makefile

## Makefile

```
Set some variables to use in our command
First, we set the compiler to be g++
CXX=g++

Then, we say that we want to compile with g++'s recommended warnings and some extra ones.
CXXFLAGS=-Wall -Wextra -pedantic

This will be the output file
EXE=app

SRCS=main.cpp

When you call `make` at the command line, this "target" is called.
The $(EXE) at the right says that the `all` target depends on the `$(EXE)` target.
$(EXE) expands to be the content of the EXE variable
Note: Because this is the first target, it becomes the default target if `make` is called
without target
all: $(EXE)

This is equivalent to saying
app: $(SRCS)
$(SRCS) can be separated, which means that this target would depend on each file.
Note that this target has a "method body": the part indented by a tab (not four spaces).
When we build this target, make will execute the command, which is:
g++ -Wall -Wextra -pedantic -o app main.cpp
I.E. Compile main.cpp with warnings, and output to the file ./app
$(EXE): $(SRCS)
 @$(CXX) $(CXXFLAGS) -o $@ $(SRCS)

This target should reverse the `all` target. If you call
make with an argument, like `make clean`, the corresponding target
gets called.
clean:
 @rm -f $(EXE)
```

```
◦ Makefile:10: *** missing separator. Stop. Makefile:10: *** missing separator.
Stop.
```

```
$ cd ~/Path/to/project
$ make
```

```
$ ls
app main.cpp Makefile

$./app
Hello World!

$ make clean
$ ls
main.cpp Makefile
```

---

## make。 a.cpp b.cpp b.cpp。

```
.
+-- src
| +-- a.cpp
| +-- a.hpp
| +-- b.cpp
| +-- b.hpp
+-- Makefile
```

## Makefile

### Makefile

```
CXX=g++
CXXFLAGS=-Wall -Wextra -pedantic
EXE=app

SRCS_GLOB=src/*.cpp
SRCS=$(wildcard $(SRCS_GLOB))
OBJS=$(SRCS:.cpp=.o)

all: $(EXE)

$(EXE): $(OBJS)
 @$ (CXX) -o $@ $ (OBJS)

depend: .depend

.depend: $(SRCS)
 @-rm -f ./depend
 @$ (CXX) $(CXXFLAGS) -MM $^>>./depend

clean:
 -rm -f $(EXE)
 -rm $ (OBJS)
 -rm *~
 -rm .depend

include .depend
```

### 。 Makefile。

---

make [stackoverflowdmckeestackoverflow](#) ◦

## SCons

[Scons -A Python -language"Hello World"C ++](#) ◦

SConstruct [SCons](#) ◦ `hello.cpp` ◦

```
Program('hello.cpp')
```

scons ◦

```
$ scons
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
g++ -o hello.o -c hello.cpp
g++ -o hello hello.o
scons: done building targets.
```

◦

EnvironmentGlob ◦ SConstruct

```
env=Environment(CPPPATH='/usr/include/boost/',
 CPPDEFINES=[],
 LIBS=[],
 SCONS_CXX_STANDARD="c++11"
)

env.Program('hello', Glob('src/*.cpp'))
```

srccpphello ◦ `CPPPATH/usr/include/boost` **C ++ 11** ◦

---

Ninja<sup>™</sup> ◦ [NinjaCMakeMeson](#) ◦

[NinjaC ++PythonChromiumSCons](#) ◦

## NMAKEMicrosoft

---

NMAKEMicrosoft [Microsoft Visual Studio/Visual C ++](#) ◦

[NMAKEMakeUnixMakeWindowsUnix](#) ◦

## AutotoolsGNU

AutotoolsGNU。 MakefileGNU Make。 Autotools。

Autotools

- Autoconf
- Automake<sub>make</sub>

AutotoolsUnixMakefile

```
./configure && make && make install
```

AutotoolsPOSIX。

<https://riptutorial.com/zh-TW/cplusplus/topic/8200/>

# 93:

## Examples

### std :: for\_each

```
template<class InputIterator, class Function>
 Function for_each(InputIterator first, InputIterator last, Function f);
```

ffirst [first, last)last - 1[first, last)°

first, last - f°

f - [first, last)°

f C ++ 11std::move(f) C ++ 11°

last - firstf last - first°

### C ++ 11

```
std::vector<int> v { 1, 2, 4, 8, 16 };
std::for_each(v.begin(), v.end(), [](int elem) { std::cout << elem << " "; });
```

vvstdout °

### std :: next\_permutation

```
template< class Iterator >
bool next_permutation(Iterator first, Iterator last);
template< class Iterator, class Compare >
bool next_permutation(Iterator first, Iterator last, Compare cmpFun);
```

[firstlast]° cmpFun °

first -

last -

true°

false°

Onnfirstlast°

```

std::vector< int > v { 1, 2, 3 };
do
{
 for(int i = 0; i < v.size(); i += 1)
 {
 std::cout << v[i];
 }
 std::cout << std::endl;
}while(std::next_permutation(v.begin(), v.end()));

```

1,2,3°

```

123
132
213
231
312
321

```

**std ::**

<numeric>

```

template<class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init); // (1)

template<class InputIterator, class T, class BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init, BinaryOperation f); // (2)

```

**std :: accumulate**[first, last)f[first, last)init°

```

T acc = init;
for (auto it = first; first != last; ++it)
 acc = f(acc, *it);
return acc;

```

1operator+f°

first, last - f°

init - °

f - °

f°

**Onxk n<sub>first</sub>last Ok<sub>f</sub>**°

```

std::vector<int> v { 2, 3, 4 };
auto sum = std::accumulate(v.begin(), v.end(), 1);
std::cout << sum << std::endl;

```

10

## C++ 11

```
class Converter {
public:
 int operator()(int a, int d) const { return a * 10 + d; }
};
```

```
const int ds[3] = {1, 2, 3};
int n = std::accumulate(ds, ds + 3, 0, Converter());
std::cout << n << std::endl;
```

## C++ 11

```
const std::vector<int> ds = {1, 2, 3};
int n = std::accumulate(ds.begin(), ds.end(),
 0,
 [](int a, int d) { return a * 10 + d; });
std::cout << n << std::endl;
```

123

## std::

```
template <class InputIterator, class T>
InputIterator find (InputIterator first, InputIterator last, const T& val);
```

## val [firstlast

first => iterator last => iterator val =>

==valvallast.

```
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int main(int argc, const char * argv[]) {

 //create a vector
 vector<int> intVec {4, 6, 8, 9, 10, 30, 55,100, 45, 2, 4, 7, 9, 43, 48};

 //define iterators
 vector<int>::iterator itr_9;
 vector<int>::iterator itr_43;
 vector<int>::iterator itr_50;

 //calling find
 itr_9 = find(intVec.begin(), intVec.end(), 9); //occurs twice
 itr_43 = find(intVec.begin(), intVec.end(), 43); //occurs once

 //a value not in the vector
 itr_50 = find(intVec.begin(), intVec.end(), 50); //does not occur
```

```

cout << "first occurrence of: " << *itr_9 << endl;
cout << "only occurrence of: " << *itr_43 << endl;

/*
 let's prove that itr_9 is pointing to the first occurrence
 of 9 by looking at the element after 9, which should be 10
 not 43
*/
cout << "element after first 9: " << *(itr_9 + 1) << endl;

/*
 to avoid dereferencing intVec.end(), let's look at the
 element right before the end
*/
cout << "last element: " << *(itr_50 - 1) << endl;

return 0;
}

```

```

first occurrence of: 9
only occurrence of: 43
element after first 9: 10
last element: 48

```

## std ::

```

template <class InputIterator, class T>
typename iterator_traits<InputIterator>::difference_type
count (InputIterator first, InputIterator last, const T& val);

```

## val

```

first =>
last =>
val =>

```

## ==val

```

#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int main(int argc, const char * argv[]) {

 //create vector
 vector<int> intVec{4,6,8,9,10,30,55,100,45,2,4,7,9,43,48};

 //count occurrences of 9, 55, and 101
 size_t count_9 = count(intVec.begin(), intVec.end(), 9); //occurs twice
 size_t count_55 = count(intVec.begin(), intVec.end(), 55); //occurs once
 size_t count_101 = count(intVec.begin(), intVec.end(), 101); //occurs once

 //print result
}

```



```

cout << "There are " << count_9 << " 9s"<< endl;
cout << "There is " << count_55 << " 55"<< endl;
cout << "There is " << count_101 << " 101"<< ends;

//find the first element == 4 in the vector
vector<int>::iterator itr_4 = find(intVec.begin(), intVec.end(), 4);

//count its occurrences in the vector starting from the first one
size_t count_4 = count(itr_4, intVec.end(), *itr_4); // should be 2

cout << "There are " << count_4 << " " << *itr_4 << endl;

return 0;
}

```

```

There are 2 9s
There is 1 55
There is 0 101
There are 2 4

```

## std :: count\_if

```

template <class InputIterator, class UnaryPredicate>
typename iterator_traits<InputIterator>::difference_type
count_if (InputIterator first, InputIterator last, UnaryPredicate red);

```

true

first => iterator<sub>last</sub> => iterator<sub>red</sub> => true/false

true◦

```

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

/*
 Define a few functions to use as predicates
*/

//return true if number is odd
bool isOdd(int i){
 return i%2 == 1;
}

//functor that returns true if number is greater than the value of the constructor parameter
provided
class Greater {
 int _than;
public:
 Greater(int th): _than(th){}
 bool operator()(int i){
 return i > _than;
 }
}

```

```

};

int main(int argc, const char * argv[]) {

 //create a vector
 vector<int> myvec = {1,5,8,0,7,6,4,5,2,1,5,0,6,9,7};

 //using a lambda function to count even numbers
 size_t evenCount = count_if(myvec.begin(), myvec.end(), [](int i){return i % 2 == 0;}); //
 >= C++11

 //using function pointer to count odd number in the first half of the vector
 size_t oddCount = count_if(myvec.begin(), myvec.end()- myvec.size()/2, isOdd);

 //using a functor to count numbers greater than 5
 size_t greaterCount = count_if(myvec.begin(), myvec.end(), Greater(5));

 cout << "vector size: " << myvec.size() << endl;
 cout << "even numbers: " << evenCount << " found" << endl;
 cout << "odd numbers: " << oddCount << " found" << endl;
 cout << "numbers > 5: " << greaterCount << " found"<< endl;

 return 0;
}

```

```

vector size: 15
even numbers: 7 found
odd numbers: 4 found
numbers > 5: 6 found

```

## std :: find\_if

```

template <class InputIterator, class UnaryPredicate>
InputIterator find_if (InputIterator first, InputIterator last, UnaryPredicate pred);

```

predtrue°

first => iterator last => iterator pred => true/false

predtrue° val

```

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

/*
 define some functions to use as predicates
*/

//Returns true if x is multiple of 10
bool multOf10(int x) {
 return x % 10 == 0;
}

```

```

//returns true if item greater than passed in parameter
class Greater {
 int _than;

public:
 Greater(int th):_than(th){

 }
 bool operator()(int data) const
 {
 return data > _than;
 }
};

int main()
{

 vector<int> myvec {2, 5, 6, 10, 56, 7, 48, 89, 850, 7, 456};

 //with a lambda function
 vector<int>::iterator gt10 = find_if(myvec.begin(), myvec.end(), [](int x){return x>10;});
 // >= C++11

 //with a function pointer
 vector<int>::iterator pow10 = find_if(myvec.begin(), myvec.end(), multOf10);

 //with functor
 vector<int>::iterator gt5 = find_if(myvec.begin(), myvec.end(), Greater(5));

 //not Found
 vector<int>::iterator nf = find_if(myvec.begin(), myvec.end(), Greater(1000)); // nf points
to myvec.end()

 //check if pointer points to myvec.end()
 if(nf != myvec.end()) {
 cout << "nf points to: " << *nf << endl;
 }
 else {
 cout << "item not found" << endl;
 }

 cout << "First item > 10: " << *gt10 << endl;
 cout << "First Item n * 10: " << *pow10 << endl;
 cout << "First Item > 5: " << *gt5 << endl;

 return 0;
}

```

```

item not found
First item > 10: 56
First Item n * 10: 10
First Item > 5: 6

```

## std :: min\_element

```

template <class ForwardIterator>
ForwardIterator min_element (ForwardIterator first, ForwardIterator last);

template <class ForwardIterator, class Compare>
ForwardIterator min_element (ForwardIterator first, ForwardIterator last, Compare comp);

```

first -  
last - comp - truefalse2.

o

```

#include <iostream>
#include <algorithm>
#include <vector>
#include <utility> //to use make_pair

using namespace std;

//function compare two pairs
bool pairLessThanFunction(const pair<string, int> &p1, const pair<string, int> &p2)
{
 return p1.second < p2.second;
}

int main(int argc, const char * argv[]) {

 vector<int> intVec {30,200,167,56,75,94,10,73,52,6,39,43};

 vector<pair<string, int>> pairVector = {make_pair("y", 25), make_pair("b", 2),
make_pair("z", 26), make_pair("e", 5) };

 // default using < operator
 auto minInt = min_element(intVec.begin(), intVec.end());

 //Using pairLessThanFunction
 auto minPairFunction = min_element(pairVector.begin(), pairVector.end(),
pairLessThanFunction);

 //print minimum of intVector
 cout << "min int from default: " << *minInt << endl;

 //print minimum of pairVector
 cout << "min pair from PairLessThanFunction: " << (*minPairFunction).second << endl;

 return 0;
}

```

```

min int from default: 6
min pair from PairLessThanFunction: 2

```

## std :: nth\_element

`std::nth_element`  $n^o$   $nn^o$  ; $o$

- °

$n^{\lceil n/2 \rceil}$ . 536.

°

```
std::vector<int> v{5, 1, 2, 3, 4};

std::vector<int>::iterator b = v.begin();
std::vector<int>::iterator e = v.end();

std::vector<int>::iterator med = b;
std::advance(med, v.size() / 2);

// This makes the 2nd position hold the median.
std::nth_element(b, med, e);

// The median is now at v[2].
```

*p*

```
const std::size_t pos = p * std::distance(b, e);

std::advance(nth, pos);
```

pos°

<https://riptutorial.com/zh-TW/cplusplus/topic/3177/>

# 94:

C ++ 14 ◦ ◦ ◦ `std::vector` `std::vector<int>` ◦

- `template < template-parameter-list >`
- `< template-parameter-list > /*C ++ 11 */`
- `<>`
- 
- `extern /*C ++ 11 */`
- `template < template-parameter-list > class ... opt identifier opt`
- `template < template-parameter-list > opt = id-expression`
- `template < template-parameter-list > typename ... opt identifier opt /*C ++ 17 */`
- `template < template-parameter-list > typename identifier opt = id-expression /*C ++ 17 */`
- ◦ `id-expression`
- `postfix-expression - > template id-expression`
- `nested-name-specifier` `template` `simple-template-id ::`

template C ++ ◦

1. `<>` ◦

```
template <class T>
void increment(T& x) { ++x; }
```

2. `<>` ◦

```
template <class T>
void print(T x);

template <> // <-- keyword used in this sense here
void print(const char* s) {
 // output the content of the string
 printf("%s\n", s);
}
```

3. `<>` ◦

```
template <class T>
std::set<T> make_singleton(T x) { return std::set<T>(x); }

template std::set<int> make_singleton(int x); // <-- keyword used in this sense here
```

4. ◦

```
template <class T, template <class U> class Alloc>
// ^^^^^^^^ keyword used in this sense here
class List {
 struct Node {
```

```

 T value;
 Node* next;
};
Alloc<Node> allocator;
Node* allocate_node() {
 return allocator.allocate(sizeof(T));
}
// ...
};

```

## 5. ...-> ◦

```

struct Allocator {
 template <class T>
 T* allocate();
};

template <class T, class Alloc>
class List {
 struct Node {
 T value;
 Node* next;
 }
 Alloc allocator;
 Node* allocate_node() {
 // return allocator.allocate<Node>(); // error: < and > are interpreted as
 // comparison operators
 return allocator.template allocate<Node>(); // ok; allocate is a template
 // ^^^^^^^ keyword used in this sense here
 }
};

```

## C++ 11 export ◦ ◦

foo.h

```

#ifndef FOO_H
#define FOO_H
export template <class T> T identity(T x);
#endif

```

foo.cpp

```

#include "foo.h"
template <class T> T identity(T x) { return x; }

```

main.cpp

```

#include "foo.h"
int main() {
 const int x = identity(42); // x is 42
}

```

export ◦ C++ 11; export ◦ ◦

# Examples

◦

```
// 'T' stands for the unknown type
// Both of our arguments will be of the same type.
template<typename T>
void printSum(T add1, T add2)
{
 std::cout << (add1 + add2) << std::endl;
}
```

◦

```
printSum<int>(4, 5);
printSum<float>(4.5f, 8.9f);
```

## template;C ++◦

◦

```
printSum(4, 5); // Both parameters are int.
 // This allows the compiler deduce that the type
 // T is also int.

printSum(5.0, 4); // In this case the parameters are two different types.
 // The compiler is unable to deduce the type of T
 // because there are contradictions. As a result
 // this is a compile time error.
```

◦ make\_X()template structure X

```
// The make_X pattern looks like this.
// 1) A template structure with 1 or more template types.
template<typename T1, typename T2>
struct MyPair
{
 T1 first;
 T2 second;
};
// 2) A make function that has a parameter type for
// each template parameter in the template structure.
template<typename T1, typename T2>
MyPair<T1, T2> make_MyPair(T1 t1, T2 t2)
{
 return MyPair<T1, T2>{t1, t2};
}
```

```
auto val1 = MyPair<int, float>{5, 8.7}; // Create object explicitly defining the types
auto val2 = make_MyPair(5, 8.7); // Create object using the types of the paramters.
 // In this code both val1 and val2 are the same
 // type.
```



o o o

```
template <typename T>
void f(T &&t);
```

t

```
struct X { };

X x;
f(x); // calls f<X&>(x)
f(X()); // calls f<X>(x)
```

TX X& tX TXt° X X&&°

decltype(t)T°

tstd::forward

```
template <typename T>
void f(T &&t) {
 g(std::forward<T>(t));
}
```

```
template <typename... Args>
void f(Args&&... args) {
 g(std::forward<Args>(args)...);
}
```

v

```
#include <vector>

template <typename T>
void f(std::vector<T> &&v);
```

o o o main()

```
#include <iostream>
using std::cout;

template <typename T> // A simple class to hold one number of any type
class Number {
public:
 void setNum(T n); // Sets the class field to the given number
 T plus1() const; // returns class field's "follower"
private:
 T num; // Class field
};

template <typename T> // Set the class field to the given number
void Number<T>::setNum(T n) {
 num = n;
}
```

```

template <typename T> // returns class field's "follower"
T Number<T>::plus1() const {
 return num + 1;
}

int main() {
 Number<int> anInt; // Test with an integer (int replaces T in the class)
 anInt.setNum(1);
 cout << "My integer + 1 is " << anInt.plus1() << "\n"; // Prints 2

 Number<double> aDouble; // Test with a double
 aDouble.setNum(3.1415926535897);
 cout << "My double + 1 is " << aDouble.plus1() << "\n"; // Prints 4.14159

 Number<float> aFloat; // Test with a float
 aFloat.setNum(1.4);
 cout << "My float + 1 is " << aFloat.plus1() << "\n"; // Prints 2.4

 return 0; // Successful completion
}

```

/o

```

template <typename T>
T sqrt(T t) { /* Some generic implementation */ }

```

```

template<>
int sqrt<int>(int i) { /* Highly optimized integer implementation */ }

```

sqrt(4.0) sqrt(4) o

o /

```

// Common case:
template<typename T, typename U>
struct S {
 T t_val;
 U u_val;
};

// Special case when the first template argument is fixed to int
template<typename V>
struct S<int, V> {
 double another_value;
 int foo(double arg) { // Do something }
};

```

o

o

```

template<typename T, typename U, typename V>
struct S {
 static void foo() {

```

```

 std::cout << "General case\n";
 }
};

template<typename U, typename V>
struct S<int, U, V> {
 static void foo() {
 std::cout << "T = int\n";
 }
};

template<typename V>
struct S<int, double, V> {
 static void foo() {
 std::cout << "T = int, U = double\n";
 }
};

```

```

S<std::string, int, double>::foo();
S<int, float, std::string>::foo();
S<int, double, std::string>::foo();

```

```

General case
T = int
T = int, U = double

```

```

template<typename T, typename U>
void foo(T t, U u) {
 std::cout << "General case: " << t << " " << u << std::endl;
}

// OK.
template<>
void foo<int, int>(int a1, int a2) {
 std::cout << "Two ints: " << a1 << " " << a2 << std::endl;
}

void invoke_foo() {
 foo(1, 2.1); // Prints "General case: 1 2.1"
 foo(1,2); // Prints "Two ints: 1 2"
}

// Compilation error: partial function specialization is not allowed.
template<typename U>
void foo<std::string, U>(std::string t, U u) {
 std::cout << "General case: " << t << " " << u << std::endl;
}

```

o o o

```

template <class T, size_t N = 10>
struct my_array {
 T arr[N];
};

int main() {
 /* Default parameter is ignored, N = 5 */
}

```

```

my_array<int, 5> a;

/* Print the length of a.arr: 5 */
std::cout << sizeof(a.arr) / sizeof(int) << std::endl;

/* Last parameter is omitted, N = 10 */
my_array<int> b;

/* Print the length of a.arr: 10 */
std::cout << sizeof(b.arr) / sizeof(int) << std::endl;
}

```

## C++ 11

```
template<typename T> using pointer = T*;
```

pointer<T>T\*

```
pointer<int> p = new int; // equivalent to: int* p = new int;
```

◦

```

template<typename T>
struct nonconst_pointer_helper { typedef T* type; };

template<typename T>
struct nonconst_pointer_helper<T const> { typedef T* type; };

template<typename T> using nonconst_pointer = nonconst_pointer_helper<T>::type;

```

◦ ◦

```

template <class T>
struct Tag1 { };

template <class T>
struct Tag2 { };

template <template <class> class Tag>
struct IntTag {
 typedef Tag<int> type;
};

int main() {
 IntTag<Tag1>::type t;
}

```

## C++ 11

```

#include <vector>
#include <iostream>

template <class T, template <class...> class C, class U>
C<T> cast_all(const C<U> &c) {

```

```

 C<T> result(c.begin(), c.end());
 return result;
}

int main() {
 std::vector<float> vf = {1.2, 2.6, 3.7};
 auto vi = cast_all<int>(vf);
 for(auto &&i: vi) {
 std::cout << i << std::endl;
 }
}

```

## auto

### C++ 17。

```

template <class T, T N>
struct integral_constant {
 using type = T;
 static constexpr T value = N;
};

using five = integral_constant<int, 5>;

```

decltype(expr), expr ◦ auto

### C++ 17

```

template <auto N>
struct integral_constant {
 using type = decltype(N);
 static constexpr type value = N;
};

using five = integral_constant<5>;

```

## unique\_ptr

unique\_ptr ◦ C API -

```

template <auto DeleteFn>
struct FunctionDeleter {
 template <class T>
 void operator()(T* ptr) const {
 DeleteFn(ptr);
 }
};

template <T, auto DeleteFn>
using unique_ptr_deleter = std::unique_ptr<T, FunctionDeleter<DeleteFn>>;

```

Tunique\_ptr

```

unique_ptr_deleter<std::FILE, std::fclose> p;

```

- 
- 
- 
- 
- `std::nullptr_t` ◦

## Template Argument Deduction ◦

```
#include <iostream>

template<typename T, std::size_t size>
std::size_t size_of(T (&anArray)[size]) // Pass array by reference. Requires.
{ // an exact size. We allow all sizes
 return size; // by using a template "size".
}

int main()
{
 char anArrayOfChar[15];
 std::cout << "anArrayOfChar: " << size_of(anArrayOfChar) << "\n";

 int anArrayOfData[] = {1,2,3,4,5,6,7,8,9};
 std::cout << "anArrayOfData: " << size_of(anArrayOfData) << "\n";
}
```

```
#include <array>
int main ()
{
 std::array<int, 5> foo; // int is a type parameter, 5 is non-type
}
```

◦

## C++ 14

- `std::tuple` ◦ `std::tuple` ◦

```
template<typename ... T>
struct DataStructure {};
```

`DataStructure<> data` ◦

```
template<typename T, typename ... Rest>
struct DataStructure<T, Rest ...>
{
 DataStructure(const T& first, const Rest& ... rest)
 : first(first)
 , rest(rest...)
 {}

 T first;
 DataStructure<Rest ... > rest;
};
```

`DataStructure<int, float, std::string> data(1, 2.1, "hello")`

◦

T Rest◦ T first◦ DataStructure<Rest ... > rest◦ rest◦

DataStructure<int, float> data◦ int firstDataStructure<float> rest◦ restfloat first

DataStructure<> rest◦ rest◦

```
DataStructure<int, float>
-> int first
-> DataStructure<float> rest
 -> float first
 -> DataStructure<> rest
 -> (empty)
```

DataStructure<int, float, std::string> dataDataStructure<int, float, std::string> data

data.rest.rest.first◦ getget

```
template<typename T, typename ... Rest>
struct DataStructure<T, Rest ...>
{
 ...
 template<size_t idx>
 auto get()
 {
 return GetHelper<idx, DataStructure<T, Rest...>>::get(*this);
 }
 ...
};
```

get - data.get<1>() std::tuple◦ GetHelper◦ DataStructuregetidx - ◦ **C ++ 14**auto◦

◦ ◦

```
template<size_t idx, typename T>
struct GetHelper;
```

idx==0◦ first

```
template<typename T, typename ... Rest>
struct GetHelper<0, DataStructure<T, Rest ... >>
{
 static T get(DataStructure<T, Rest...>& data)
 {
 return data.first;
 }
};
```

idxrestGetHelper

```
template<size_t idx, typename T, typename ... Rest>
struct GetHelper<idx, DataStructure<T, Rest ... >>
{
 static auto get(DataStructure<T, Rest...>& data)
```

```

 {
 return GetHelper<idx-1, DataStructure<Rest ...>>::get(data.rest);
 }
};

```

DataStructure<int, float> data data.get<1>() ◦ GetHelper<1, DataStructure<int, float>>::get(data)  
 GetHelper<0, DataStructure<float>>::get(data.rest) idx0 data.rest.first ◦

main

```

#include <iostream>

template<size_t idx, typename T>
struct GetHelper;

template<typename ... T>
struct DataStructure
{
};

template<typename T, typename ... Rest>
struct DataStructure<T, Rest ...>
{
 DataStructure(const T& first, const Rest& ... rest)
 : first(first)
 , rest(rest...)
 {}

 T first;
 DataStructure<Rest ... > rest;

 template<size_t idx>
 auto get()
 {
 return GetHelper<idx, DataStructure<T, Rest...>>::get(*this);
 }
};

template<typename T, typename ... Rest>
struct GetHelper<0, DataStructure<T, Rest ... >>
{
 static T get(DataStructure<T, Rest...>& data)
 {
 return data.first;
 }
};

template<size_t idx, typename T, typename ... Rest>
struct GetHelper<idx, DataStructure<T, Rest ... >>
{
 static auto get(DataStructure<T, Rest...>& data)
 {
 return GetHelper<idx-1, DataStructure<Rest ...>>::get(data.rest);
 }
};

int main()
{
 DataStructure<int, float, std::string> data(1, 2.1, "Hello");

```



```

std::cout << data.get<0>() << std::endl;
std::cout << data.get<1>() << std::endl;
std::cout << data.get<2>() << std::endl;

return 0;
}

```

◦ ◦ ◦

```

// print_string.h
template <class T>
void print_string(const T* str);

// print_string.cpp
#include "print_string.h"
template void print_string(const char*);
template void print_string(const wchar_t*);

```

print\_string<char> print\_string<wchar\_t> print\_string.cpp print\_string◦ ◦

## C++ 11

extern ◦ ◦ TU◦

foo.h

```

#ifndef FOO_H
#define FOO_H
template <class T> void foo(T x) {
 // complicated implementation
}
#endif

```

foo.cpp

```

#include "foo.h"
// explicit instantiation definitions for common cases
template void foo(int);
template void foo(double);

```

main.cpp

```

#include "foo.h"
// we already know foo.cpp has explicit instantiation definitions for these
extern template void foo(double);
int main() {
 foo(42); // instantiates foo<int> here;
 // wasteful since foo.cpp provides an explicit instantiation already!
 foo(3.14); // does not instantiate foo<double> here;
 // uses instantiation of foo<double> in foo.cpp instead
}

```

<https://riptutorial.com/zh-TW/cplusplus/topic/460/>

# 95:

auto

C++ 98 C++

```
int main()
{
 auto int i = 5; // removing auto has no effect
}
```

.

## Examples

auto

```
std::map< std::string, std::shared_ptr< Widget > > table;
// C++98
std::map< std::string, std::shared_ptr< Widget > >::iterator i = table.find("42");
// C++11/14/17
auto j = table.find("42");
```

## for

```
vector<int> v = {0, 1, 2, 3, 4, 5};
for(auto n: v)
 std::cout << n << ' ';
```

## lambdas

```
auto f = [](){ std::cout << "lambda\n"; };
f();
```

```
auto w = std::make_shared< Widget >();
```

```
auto myMap = std::map<int, float>();
myMap.emplace(1, 3.14);
```

```
std::pair<int, float> const& firstPair2 = *myMap.begin(); // copy!
auto const& firstPair = *myMap.begin(); // no copy!
```

```
std::pair<const int, float>
```

auto

```
auto mult(int c) {
 return c * std::valarray<int>{1};
}
```

```

}

auto v = mult(3);
std::cout << v[0]; // some value that could be, but almost certainly is not, 3.

```

valarrayoperator\*valarray auto multstd::valarray<int> 3

## autoconstreferences

autointchar const&const

sstd::string fors

```

std::vector<std::string> strings = { "stuff", "things", "misc" };
for(auto s : strings) {
 std::cout << s << std::endl;
}

```

s s.append(" and stuff") strings

sauto&std::string&

```

for(auto& s : strings) {
 std::cout << s << std::endl;
}

```

sstrings

sconst auto& constconst

```

for(const auto& s : strings) {
 std::cout << s << std::endl;
}

```

sconst

forautoconst auto&

auto

```

auto main() -> int {}

```

```

int main() {}

```

decltypestd::declval<T>

```

template <typename T1, typename T2>
auto Add(const T1& lhs, const T2& rhs) -> decltype(lhs + rhs) { return lhs + rhs; }

```

## lambdaC ++ 14

### C ++ 14

#### C ++ 14lambdaauto

```
auto print = [](const auto& arg) { std::cout << arg << std::endl; };

print(42);
print("hello world");
```

### lambda

```
struct lambda {
 template <typename T>
 auto operator ()(const T& arg) const {
 std::cout << arg << std::endl;
 }
};
```

```
lambda print;

print(42);
print("hello world");
```

auto° °

```
std::vector<bool> flags{true, true, false};
auto flag = flags[0];
flags.push_back(true);
```

flagbool std::vector<bool>::reference vectorbool operator []operator booloperator bool°

flags.push\_back(true)°

```
void foo(bool b);

std::vector<bool> getFlags();

auto flag = getFlags()[5];
foo(flag);
```

vectorflag° foo°

auto

```
auto flag = static_cast<bool>(getFlags()[5]);
```

boolauto°

° °



# 96:

operator >>operator <<◦

```
#include <iomanip> ◦
```

◦

1. `os.width(n);os << std::setw(n);`  
`is.width(n);is >> std::setw(n);`
2. `os.precision(n);os << std::setprecision(n);`  
`is.precision(n);is >> std::setprecision(n);`
3. `os.setfill(c);os << std::setfill(c);`
4. `str >> std::setbase(base);str << std::setbase(base);`

```
str.setf(base == 8 ? std::ios_base::oct :
 base == 10 ? std::ios_base::dec :
 base == 16 ? std::ios_base::hex :
 std::ios_base::fmtflags(0),
 std::ios_base::basefield);
```

5. `os.setf(std::ios_base::flag);os << std::flag;`  
`is.setf(std::ios_base::flag);is >> std::flag;`  
`os.unsetf(std::ios_base::flag);os << std::no ## flag;`  
`is.unsetf(std::ios_base::flag);is >> std::no ## flag;`  
**## -**  
flag **S boolalpha showbase showpoint showpos skipws uppercase** ◦

6. `std::ios_base::basefield` ◦

flag **S dec hexoct**

- `os.setf(std::ios_base::flag, std::ios_base::basefield);os << std::flag;`  
`is.setf(std::ios_base::flag, std::ios_base::basefield);is >> std::flag;`  
**1**
- `str.unsetf(std::ios_base::flag, std::ios_base::basefield);`  
`str.setf(std::ios_base::fmtflags(0), std::ios_base::basefield);`  
**2**

7. `std::ios_base::adjustfield`

◦

flag **S left rightinternal**

- `os.setf(std::ios_base::flag, std::ios_base::adjustfield); os << std::flag;`  
`is.setf(std::ios_base::flag, std::ios_base::adjustfield); is >> std::flag;`

**1**

- `str.unsetf(std::ios_base::flag, std::ios_base::adjustfield);`  
`str.setf(std::ios_base::fmtflags(0), std::ios_base::adjustfield);`

**2**

**1**unsetf◦

**2**flag◦

### 8. `std::ios_base::floatfield` ◦

- `os.setf(std::ios_base::flag, std::ios_base::floatfield); os << std::flag;`  
`is.setf(std::ios_base::flag, std::ios_base::floatfield); is >> std::flag;`  
flag **S fixedscientific** ◦
- `os.setf(std::ios_base::fmtflags(0), std::ios_base::floatfield); os << std::defaultfloat;`  
`is.setf(std::ios_base::fmtflags(0), std::ios_base::floatfield); is >> std::defaultfloat;`

### 9. `str.setf(std::ios_base::fmtflags(0), std::ios_base::flag); str.unsetf(std::ios_base::flag)`

flag **S basefield adjustfield floatfield** ◦

- 10.** `os.setf(mask) os << setiosflags(mask);`  
`is.setf(mask) is >> setiosflags(mask);`  
`os.unsetf(mask) os << resetiosflags(mask);`  
`is.unsetf(mask) is >> resetiosflags(mask);`  
`std::ios_base::fmtflagsmask` ◦

## Examples

`std::boolalpha` `std::noboolalpha` - ◦

```
std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" << std::boolalpha << boolValue
 << "\" was parsed as \" << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0
```

`std::showbasestd::noshowbase` - °

`std::dec std::hex std::oct` - °

```
#include <sstream>

std::cout << std::dec << 29 << ' - '
 << std::hex << 29 << ' - '
 << std::showbase << std::oct << 29 << ' - '
 << std::noshowbase << 29 << '\n';

int number;
std::istringstream("3B") >> std::hex >> number;
std::cout << std::dec << 10;
// Output: 22 - 1D - 35 - 035
// 59
```

`std::ios_base::noshowbasestd::ios_base::dec` °

`std::istringstream< sstream >` °

`std::uppercsestd::nouppercase` - ° °

```
std::cout << std::hex << std::showbase
 << "0x2a with nouppercase: " << std::nouppercase << 0x2a << '\n'
 << "1e-10 with uppercase: " << std::uppercase << 1e-10 << '\n'
}
// Output: 0x2a with nouppercase: 0x2a
// 1e-10 with uppercase: 1E-10
```

`std::nouppercase` °

`std::setw(n) - /n` °

`widthn0` °

```
std::cout << "no setw:" << 51 << '\n'
 << "setw(7): " << std::setw(7) << 51 << '\n'
 << "setw(7), more output: " << 13
 << std::setw(7) << std::setfill('*') << 67 << ' ' << 94 << '\n';

char* input = "Hello, world!";
char arr[10];
std::cin >> std::setw(6) >> arr;
std::cout << "Input from \"Hello, world!\" with setw(6) gave \"" << arr << "\"\n";

// Output: 51
// setw(7): 51
// setw(7), more output: 13*****67 94

// Input: Hello, world!
// Output: Input from "Hello, world!" with setw(6) gave "Hello"
```



std::setw(0) ◦

std::left std::rightstd::internal - std::ios\_base::adjustfieldstd::ios\_base::left  
std::ios\_base::rightstd::ios\_base::internalstd::ios\_base::internal◦ std::leftstd::right  
std::internal - ◦ ◦

```
#include <locale>
...

std::cout.imbue(std::locale("en_US.utf8"));

std::cout << std::left << std::showbase << std::setfill('*')
 << "flt: " << std::setw(15) << -9.87 << '\n'
 << "hex: " << std::setw(15) << 41 << '\n'
 << " $: " << std::setw(15) << std::put_money(367, false) << '\n'
 << "usd: " << std::setw(15) << std::put_money(367, true) << '\n'
 << "usd: " << std::setw(15)
 << std::setfill(' ') << std::put_money(367, false) << '\n';
// Output:
// flt: -9.87*****
// hex: 41*****
// $: $3.67*****
// usd: USD *3.67*****
// usd: $3.67

std::cout << std::internal << std::showbase << std::setfill('*')
 << "flt: " << std::setw(15) << -9.87 << '\n'
 << "hex: " << std::setw(15) << 41 << '\n'
 << " $: " << std::setw(15) << std::put_money(367, false) << '\n'
 << "usd: " << std::setw(15) << std::put_money(367, true) << '\n'
 << "usd: " << std::setw(15)
 << std::setfill(' ') << std::put_money(367, true) << '\n';
// Output:
// flt: -*****9.87
// hex: *****41
// $: $3.67*****
// usd: USD *****3.67
// usd: USD 3.67

std::cout << std::right << std::showbase << std::setfill('*')
 << "flt: " << std::setw(15) << -9.87 << '\n'
 << "hex: " << std::setw(15) << 41 << '\n'
 << " $: " << std::setw(15) << std::put_money(367, false) << '\n'
 << "usd: " << std::setw(15) << std::put_money(367, true) << '\n'
 << "usd: " << std::setw(15)
 << std::setfill(' ') << std::put_money(367, true) << '\n';
// Output:
// flt: *****-9.87
// hex: *****41
// $: *****$3.67
// usd: *****USD *3.67
// usd: USD 3.67
```

std::left ◦

`std::fixed` `std::scientific` `std::hexfloat` [C ++ 11] `std::defaultfloat` [C ++ 11] - /。

```
std::fixed std::ios_base::floatfield std::ios_base::fixed
std::scientific - to std::ios_base::scientific
std::hexfloat - std::ios_base::fixed | std::ios_base::scientific
std::defaultfloat - std::ios_base::fmtflags(0) 。
```

`fmtflags`

```
#include <sstream>
...

std::cout << '\n'
 << "The number 0.07 in fixed: " << std::fixed << 0.01 << '\n'
 << "The number 0.07 in scientific: " << std::scientific << 0.01 << '\n'
 << "The number 0.07 in hexfloat: " << std::hexfloat << 0.01 << '\n'
 << "The number 0.07 in default: " << std::defaultfloat << 0.01 << '\n';

double f;
std::istringstream is("0x1P-1022");
double f = std::strtod(is.str().c_str(), NULL);
std::cout << "Parsing 0x1P-1022 as hex gives " << f << '\n';

// Output:
// The number 0.01 in fixed: 0.070000
// The number 0.01 in scientific: 7.000000e-02
// The number 0.01 in hexfloat: 0x1.1eb851eb851ecp-4
// The number 0.01 in default: 0.07
// Parsing 0x1P-1022 as hex gives 2.22507e-308
```

`std::ios_base::fmtflags(0)` 。

```
double f;
std::istringstream("0x1P-1022") >> std::hexfloat >> f;
std::cout << "Parsing 0x1P-1022 as hex gives " << f << '\n';
// Output: Parsing 0x1P-1022 as hex gives 0
```

`std::showpoint` `std::noshowpoint` - 。

```
std::cout << "7.0 with showpoint: " << std::showpoint << 7.0 << '\n'
 << "7.0 with noshowpoint: " << std::noshowpoint << 7.0 << '\n';
// Output: 1.0 with showpoint: 7.00000
// 1.0 with noshowpoint: 7
```

`std::showpoint` 。

`std::showpos` `std::noshowpos` - + 。

```
std::cout << "With showpos: " << std::showpos
 << 0 << ' ' << -2.718 << ' ' << 17 << '\n'
```

```

 << "Without showpos: " << std::noshowpos
 << 0 << ' ' << -2.718 << ' ' << 17 << '\n';
// Output: With showpos: +0 -2.718 +17
// Without showpos: 0 -2.718 17

```

std::noshowpos°

std::unitbuf std::nounitbuf ° ° std::unitbuf°

std::setbase(base) °

```

std::setbase(8) std::ios_base::basefield std::ios_base::oct
std::setbase(16) ° std::ios_base::hex
std::setbase(10) ° std::ios_base::dec °

```

base8 1016std::ios\_base::basefieldstd::ios\_base::fmtflags(0)°

std::ios\_base::basefieldstd::ios\_base::dec std::setbase(10) °

std::setprecision(n) °

```

#include <cmath>
#include <limits>
...

typedef std::numeric_limits<long double> ld;
const long double pi = std::acos(-1.L);

std::cout << '\n'
 << "default precision (6): pi: " << pi << '\n'
 << " 10pi: " << 10 * pi << '\n'
 << "std::setprecision(4): 10pi: " << std::setprecision(4) << 10 * pi << '\n'
 << " 10000pi: " << 10000 * pi << '\n'
 << "std::fixed: 10000pi: " << std::fixed << 10000 * pi << std::defaultfloat
<< '\n'
 << "std::setprecision(10): pi: " << std::setprecision(10) << pi << '\n'
 << "max-1 radix precicion: pi: " << std::setprecision(ld::digits - 1) << pi <<
'\n'
 << "max+1 radix precision: pi: " << std::setprecision(ld::digits + 1) << pi <<
'\n'
 << "significant digits prec: pi: " << std::setprecision(ld::digits10) << pi << '\n';

// Output:
// default precision (6): pi: 3.14159
// 10pi: 31.4159
// std::setprecision(4): 10pi: 31.42
// 10000pi: 3.142e+04
// std::fixed: 10000pi: 31415.9265
// std::setprecision(10): pi: 3.141592654

```

```
// max-1 radix precision: pi:
3.14159265358979323851280895940618620443274267017841339111328125
// max+1 radix precision: pi:
3.14159265358979323851280895940618620443274267017841339111328125
// significant digits prec: pi: 3.14159265358979324
```

`std::setprecision(6)` ◦

`std::setiosflags(mask)std::resetiosflags(mask)` - `std::ios_base::fmtflagsmask`**setclear**◦

```
#include <sstream>
...

std::istringstream in("10 010 10 010 10 010");
int num1, num2;

in >> std::oct >> num1 >> num2;
std::cout << "Parsing \"10 010\" with std::oct gives: " << num1 << ' ' << num2 << '\n';
// Output: Parsing "10 010" with std::oct gives: 8 8

in >> std::dec >> num1 >> num2;
std::cout << "Parsing \"10 010\" with std::dec gives: " << num1 << ' ' << num2 << '\n';
// Output: Parsing "10 010" with std::oct gives: 10 10

in >> std::resetiosflags(std::ios_base::basefield) >> num1 >> num2;
std::cout << "Parsing \"10 010\" with autodetect gives: " << num1 << ' ' << num2 << '\n';
// Parsing "10 010" with autodetect gives: 10 8

std::cout << std::setiosflags(std::ios_base::hex |
 std::ios_base::uppercase |
 std::ios_base::showbase) << 42 << '\n';
// Output: OX2A
```

`std::skipwsstd::noskipws` - ◦ ◦

```
#include <sstream>
...

char c1, c2, c3;
std::istringstream("a b c") >> c1 >> c2 >> c3;
std::cout << "Default behavior: c1 = " << c1 << " c2 = " << c2 << " c3 = " << c3 << '\n';

std::istringstream("a b c") >> std::noskipws >> c1 >> c2 >> c3;
std::cout << "noskipws behavior: c1 = " << c1 << " c2 = " << c2 << " c3 = " << c3 << '\n';
// Output: Default behavior: c1 = a c2 = b c3 = c
// noskipws behavior: c1 = a c2 = c3 = b
```

`std::ios_base::skipws` ◦

`std::quoted(s[, delim[, escape]])` [C ++ 14] - ◦

```
s - °
delim - "°
escape - \ °
```

```
#include <sstream>
...

std::stringstream ss;
std::string in = "String with spaces, and embedded \"quotes\" too";
std::string out;

ss << std::quoted(in);
std::cout << "read in [" << in << "]\n"
 << "stored as [" << ss.str() << "]\n";

ss >> std::quoted(out);
std::cout << "written out [" << out << "]\n";
// Output:
// read in [String with spaces, and embedded "quotes" too]
// stored as ["String with spaces, and embedded \"quotes\" too"]
// written out [String with spaces, and embedded "quotes" too]
```

°

```
std::ends - '\0' °
```

```
template <class charT, class traits>
std::basic_ostream<charT, traits>& ends(std::basic_ostream<charT, traits>& os);
```

```
os.put(charT())
os << std::ends;
```

```
std::endlstd::flushout.flush() out° ° std::endl'\n'°
```

```
std::cout << "First line." << std::endl << "Second line. " << std::flush
 << "Still second line.";
// Output: First line.
// Second line. Still second line.
```

```
std::setfill(c) - c ° std::setw °
```

```
std::cout << "\nDefault fill: " << std::setw(10) << 79 << '\n'
 << "setfill('#'): " << std::setfill('#')
 << std::setw(10) << 42 << '\n';
// Output:
// Default fill: 79
// setfill('#'): #####79
```

`std::put_money(mon[, intl])` [C++ 11] ◦ `out << std::put_money(mon, intl)mon long double`  
`std::basic_stringstd::money_put facetout ◦ intltrue ◦`

```
long double money = 123.45;
// or std::string money = "123.45";

std::cout.imbue(std::locale("en_US.utf8"));
std::cout << std::showbase << "en_US: " << std::put_money(money)
 << " or " << std::put_money(money, true) << '\n';
// Output: en_US: $1.23 or USD 1.23

std::cout.imbue(std::locale("ru_RU.utf8"));
std::cout << "ru_RU: " << std::put_money(money)
 << " or " << std::put_money(money, true) << '\n';
// Output: ru_RU: 1.23 pyб or 1.23 RUB

std::cout.imbue(std::locale("ja_JP.utf8"));
std::cout << "ja_JP: " << std::put_money(money)
 << " or " << std::put_money(money, true) << '\n';
// Output: ja_JP: ¥123 or JPY 123
```

`std::put_time(tmb, fmt)` [C++ 11] - `fmt/std::tm` ◦

`tmb` - `localtime()`/`gmtime()` `const std::tm*` ◦  
`fmt` - `null` `const CharT*` ◦

```
#include <ctime>
...

std::time_t t = std::time(nullptr);
std::tm tm = *std::localtime(&t);

std::cout.imbue(std::locale("ru_RU.utf8"));
std::cout << "\nru_RU: " << std::put_time(&tm, "%c %Z") << '\n';
// Possible output:
// ru_RU: Вт 04 июл 2017 15:08:35 UTC
```

◦

`std::ws` - ◦ `std::skipws` ◦

```
#include <sstream>
...

std::string str;
std::istringstream(" \v\n\r\t Wow!There is no whitespaces!") >> std::ws >> str;
std::cout << str;
// Output: Wow!There is no whitespaces!
```

`std::get_money(mon[, intl])` [C++ 11] ◦ `in >> std::get_money(mon, intl)std::money_getin mon long`

doublestd::basic\_string° intltrue °

```
#include <sstream>
#include <locale>
...

std::istringstream in("$1,234.56 2.22 USD 3.33");
long double v1, v2;
std::string v3;

in.imbue(std::locale("en_US.UTF-8"));
in >> std::get_money(v1) >> std::get_money(v2) >> std::get_money(v3, true);
if (in) {
 std::cout << std::quoted(in.str()) << " parsed as: "
 << v1 << ", " << v2 << ", " << v3 << '\n';
}
// Output:
// "$1,234.56 2.22 USD 3.33" parsed as: 123456, 222, 333
```

[std::get\\_time\(tmb, fmt\) \[C++ 11\] - fmt tmb/°](#)

tmb - const std::tm\*

fmt - **null**const CharT\*

```
#include <sstream>
#include <locale>
...

std::tm t = {};
std::istringstream ss("2011-Februar-18 23:12:34");

ss.imbue(std::locale("de_DE.utf-8"));
ss >> std::get_time(&t, "%Y-%b-%d %H:%M:%S");
if (ss.fail()) {
 std::cout << "Parse failed\n";
}
else {
 std::cout << std::put_time(&t, "%c") << '\n';
}
// Possible output:
// Sun Feb 18 23:12:34 2011
```

°

<https://riptutorial.com/zh-TW/cplusplus/topic/10699/>

# 97:

◦

## Examples

switchcase◦ ◦ switchcase◦

```
char c = getchar();
bool confirmed;
switch (c) {
 case 'y':
 confirmed = true;
 break;
 case 'n':
 confirmed = false;
 break;
 default:
 std::cout << "invalid response!\n";
 abort();
}
```

## C++

switch◦

switchcasedefault◦ **switch**casedefault◦

◦

```
char c = getchar();
bool confirmed;
switch (c) {
 case 'y':
 confirmed = true;
 break;
 case 'n':
 confirmed = false;
 break;
 default:
 std::cout << "invalid response!\n";
 abort();
}
```

catch◦ catch...◦ ◦ ◦

```
try {
 std::vector<int> v(N);
 // do something
} catch (const std::bad_alloc&) {
 std::cout << "failed to allocate memory for vector!" << std::endl;
} catch (const std::runtime_error& e) {
```



```

 std::cout << "runtime error: " << e.what() << std::endl;
} catch (...) {
 std::cout << "unexpected exception!" << std::endl;
 throw;
}

```

## switchcase◦

```

char c = getchar();
bool confirmed;
switch (c) {
 case 'y':
 confirmed = true;
 break;
 case 'n':
 confirmed = false;
 break;
 default:
 std::cout << "invalid response!\n";
 abort();
}

```

## C++ 11

◦

```

class Base {
 // ...
 // we want to be able to delete derived classes through Base*,
 // but have the usual behaviour for Base's destructor.
 virtual ~Base() = default;
};

```

## if◦ if◦ ◦

```

int x;
std::cout << "Please enter a positive number." << std::endl;
std::cin >> x;
if (x <= 0) {
 std::cout << "You didn't enter a positive number!" << std::endl;
 abort();
}

```

## ifelse◦ else◦

```

int x;
std::cin >> x;
if (x%2 == 0) {
 std::cout << "The number is even\n";
} else {
 std::cout << "The number is odd\n";
}

```

◦

```

bool f(int arg) {
 bool result = false;
 hWidget widget = get_widget(arg);
 if (!g()) {
 // we can't continue, but must do cleanup still
 goto end;
 }
 // ...
 result = true;
end:
 release_widget(widget);
 return result;
}

```

◦

return◦

```

int f() {
 return 42;
}
int x = f(); // x is 42
int g() {
 return 3.14;
}
int y = g(); // y is 3

```

returnvoid◦ void void **-returning**◦

```

void f(int x) {
 if (x < 0) return;
 std::cout << sqrt(x);
}
int g() { return 42; }
void h() {
 return f(); // calls f, then returns
 return g(); // ill-formed
}

```

mainstd::exit◦ mainstd::exit◦

```

int main(int argc, char** argv) {
 if (argc < 2) {
 std::cout << "Missing argument\n";
 return EXIT_FAILURE; // equivalent to: exit(EXIT_FAILURE);
 }
}

```

1. throw◦

```

void print_asterisks(int count) {
 if (count < 0) {
 throw std::invalid_argument("count cannot be negative!");
 }
}

```

```
while (count--) { putchar('*'); }
}
```

## 2. throw ◦ std::terminate ◦

```
try {
 // something risky
} catch (const std::bad_alloc&) {
 std::cerr << "out of memory" << std::endl;
} catch (...) {
 std::cerr << "unexpected exception" << std::endl;
 // hope the caller knows how to handle this exception
 throw;
}
```

## 3. throwthrow ◦

```
// this function might propagate a std::runtime_error,
// but not, say, a std::logic_error
void risky() throw(std::runtime_error);
// this function can't propagate any exceptions
void safe() throw();
```

## C++ 11 ◦

throw ◦ **throw**void ◦

```
unsigned int predecessor(unsigned int x) {
 return (x > 0) ? (x - 1) : (throw std::invalid_argument("0 has no predecessor"));
}
```

try ◦ **trycatch** ◦ **trytrycatch** ◦

```
std::vector<int> v(N); // if an exception is thrown here,
// it will not be caught by the following catch block
try {
 std::vector<int> v(N); // if an exception is thrown here,
// it will be caught by the following catch block
 // do something with v
} catch (const std::bad_alloc&) {
 // handle bad_alloc exceptions from the try block
}
```

## ifif..else

### ifelse

### truefalse

```
if (condition) statement
```

## C++/

```
if (true) { /* code here */ } // evaluate that true is true and execute the code in the brackets
if (false) { /* code here */ } // always skip the code since false is always false
```

```
if(istrue()) { } // evaluate the function, if it returns true, the if will execute the code
if(isTrue(var)) { } //evaluate the return of the function after passing the argument var
if(a == b) { } // this will evaluate the return of the experssion (a==b) which will be true if equal and false if unequal
if(a) { } //if a is a boolean type, it will evaluate for its value, if it's an integer, any non zero value will be true,
```

```
if (a && b) { } // will be true only if both a and b are true (binary operators are outside the scope here
if (a || b) { } //true if a or b is true
```

## if / ifelse / else

```
if (a== "test") {
 //will execute if a is a string "test"
} else {
 // only if the first failed, will execute
}
```

```
if (a=='a') {
// if a is a char valued 'a'
} else if (a=='b') {
// if a is a char valued 'b'
} else if (a=='c') {
// if a is a char valued 'c'
} else {
//if a is none of the above
}
```

## “ switch ”

### breakcontinuegotoexit。

### break。

```
break;
```

### switchbreak i。

```
switch(conditon){
case 1: block1;
case 2: block2;
case 3: block3;
default: blockdefault;
}
```

## 1111231break

```
switch(condition){
case 1: block1;
 break;
case 2: block2;
 break;
case 3: block3;
 break;
default: blockdefault;
 break;
}
```

o

**break**if while for;

```
if(condition1){

 if(condition2){

 break;
 }
 ...
}
```

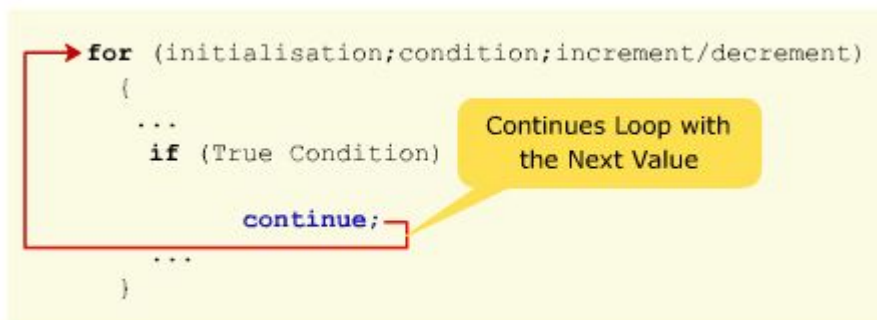
**continue**o

```
continue;
```

```
for(int i=0;i<10;i++){
if(i%2==0)
continue;
cout<<"\n @"<<i;
}
```

```
@1
@3
@5
@7
@9
```

i%2==0continue@i/o

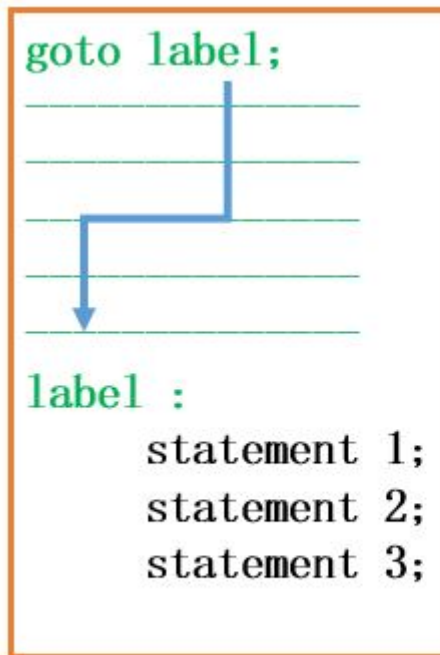


## goto

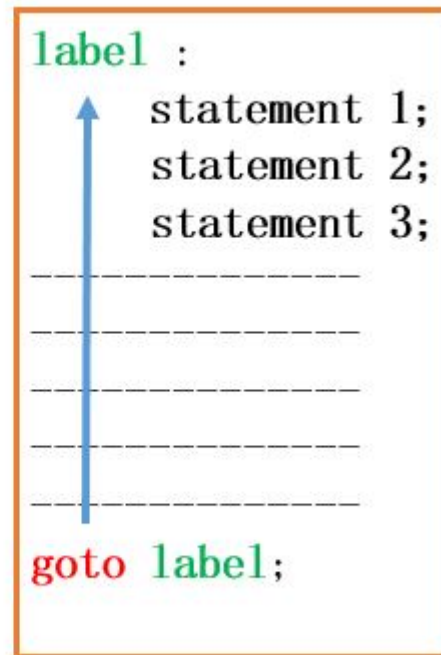
- ◦ goto ◦ :)

```
goto label;
..
.
label: statement;
```

goto ◦



Forward Reference



Backward Reference

```
int num = 1;
STEP:
do{

 if(num%2==0)
 {
 num = num + 1;
 goto STEP;
 }

 cout << "value of num : " << num << endl;
 num = num + 1;
}while(num < 10);
```

```
value of num : 1
value of num : 3
value of num : 5
value of num : 7
value of num : 9
```

num%2==0 goto do-while ◦

exitcstdlib◦ exit◦

```
void exit (int exit code);
```

cstdlibEXIT\_SUCCESSEXIT\_FAILURE ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/7837/>

# 98:

## Examples

```
float total = 0;
for(float a = 0; a != 2; a += 0.01f) {
 total += a;
}
```

0, 0.01, 0.02, 0.03, ..., 1.97, 1.98, 1.99 199 -the

1.  $a^2$
2.  $a < 2^{199}$  IEEE754<sub>201</sub>

.

```
double a = 0.1;
double b = 0.2;
double c = 0.3;
if(a + b == c)
 //This never prints on IEEE754-compliant machines
 std::cout << "This Computer is Magic!" << std::endl;
else
 std::cout << "This Computer is pretty normal, all things considered." << std::endl;
```

base10 0.1 0.2 0.310 --10-2-1/3100.333333333333333...

```
//64-bit floats have 53 digits of precision, including the whole-number-part.
double a = 0011111110111001100110011001100110011001100110011001100110011010; //imperfect
representation of 0.1
double b = 00111111110011001100110011001100110011001100110011001100110011010; //imperfect
representation of 0.2
double c = 0011111111010011001100110011001100110011001100110011001100110011; //imperfect
representation of 0.3
double a + b = 0011111111010011001100110011001100110011001100110011001100110100; //Note that
this is not quite equal to the "canonical" 0.3!
```

<https://riptutorial.com/zh-TW/cplusplus/topic/5115/>



# 99:

## Examples

Base ◦ ◦

virtual ◦ ◦

```
class Base {
public:
 virtual ~Base() = default;

private:
 // data members etc.
};

class Derived : public Base { // models Is-A relationship
public:
 // some methods

private:
 // more data members
};

// virtual destructor in Base ensures that derived destructors
// are also called when the object is destroyed
std::unique_ptr<Base> base = std::make_unique<Derived>();
base = nullptr; // safe, doesn't leak Derived's members
```

protected ◦

```
class NonPolymorphicBase {
public:
 // some methods

protected:
 ~NonPolymorphicBase() = default; // note: non-virtual

private:
 // etc.
};
```

◦

private ◦ virtual ◦ protected ◦

◦ final ◦ ◦

```
class FinalClass final { // marked final here
public:
 ~FinalClass() = default;

private:
```

```
 // etc.
};
```

◦ ◦

◦

```
class Movable {
public:
 virtual ~Movable() noexcept = default;

 // compiler won't generate these unless we tell it to
 // because we declared a destructor
 Movable(Movable&&) noexcept = default;
 Movable& operator=(Movable&&) noexcept = default;

 // declaring move operations will suppress generation
 // of copy operations unless we explicitly re-enable them
 Movable(const Movable&) = default;
 Movable& operator=(const Movable&) = default;
};
```

## Rule of Three / Five / Zero ◦

```
person(const person &other)
 : name(new char[std::strlen(other.name) + 1])
 , age(other.age)
{
 std::strcpy(name, other.name);
}

person& operator=(person const& rhs) {
 if (this != &other) {
 delete [] name;
 name = new char[std::strlen(other.name) + 1];
 std::strcpy(name, other.name);
 age = other.age;
 }

 return *this;
}
```

◦ `new[] this` ◦ ◦

```
class person {
 char* name;
 int age;
public:
 /* all the other functions ... */

 friend void swap(person& lhs, person& rhs) {
 using std::swap; // enable ADL

 swap(lhs.name, rhs.name);
 swap(lhs.age, rhs.age);
 }
}
```

```

 person& operator=(person rhs) {
 swap(*this, rhs);
 return *this;
 }
};

```

```

person p1 = ...;
person p2 = ...;
p1 = p2;

```

p2rhs ◦ operator=p1◦ \*thisrhsrhs◦ operator= this ◦ - ◦

## C++ 11

◦

```
p1 = std::move(p2);
```

p2rhs ◦ ◦

◦ ◦

```

class C{
 int i;
public:
 // the default constructor definition
 C()
 : i(0){ // member initializer list -- initialize i to 0
 // constructor function body -- can do more complex things here
 }
};

C c1; // calls default constructor of C to create object c1
C c2 = C(); // calls default constructor explicitly
C c3(); // ERROR: this intuitive version is not possible due to "most vexing parse"
C c4{}; // but in C++11 {} CAN be used in a similar way

C c5[2]; // calls default constructor for both array elements
C* c6 = new C[2]; // calls default constructor for both array elements

```

“”

```

class D{
 int i;
 int j;
public:
 // also a default constructor (can be called with no parameters)
 D(int i = 0, int j = 42)
 : i(i), j(j){
 }
};

D d; // calls constructor of D with the provided default values for the parameters

```

```

class C{
 std::string s; // note: members need to be default constructible themselves
};

C c1; // will succeed -- C has an implicitly defined default constructor

```

```

class C{
 int i;
public:
 C(int i) : i(i){}
};

C c1; // Compile ERROR: C has no (implicitly defined) default constructor

```

## C++ 11

private ° °

°

## C++ 11

C++ 11 delete °

```

class C{
 int i;
public:
 // default constructor is explicitly deleted
 C() = delete;
};

C c1; // Compile ERROR: C has its default constructor deleted

```

°

```

class C{
 int i;
public:
 // does have automatically generated default constructor (same as implicit one)
 C() = default;

 C(int i) : i(i){}
};

C c1; // default constructed
C c2(1); // constructed with the int taking constructor

```

## C++ 14

<type\_traits> std::is\_default\_constructible

```

class C1{ };
class C2{ public: C2(){} };
class C3{ public: C3(int){} };

```

```
using std::cout; using std::boolalpha; using std::endl;
using std::is_default_constructible;
cout << boolalpha << is_default_constructible<int>() << endl; // prints true
cout << boolalpha << is_default_constructible<C1>() << endl; // prints true
cout << boolalpha << is_default_constructible<C2>() << endl; // prints true
cout << boolalpha << is_default_constructible<C3>() << endl; // prints false
```

## C++ 11

### C++ 11 std::is\_default\_constructible

```
cout << boolalpha << is_default_constructible<C1>::value << endl; // prints true
```

◦ ~◦

```
class C{
 int* is;
 string s;
public:
 C()
 : is(new int[10]){
 }

 ~C(){ // destructor definition
 delete[] is;
 }
};

class C_child : public C{
 string s_ch;
public:
 C_child(){}
 ~C_child(){} // child destructor
};

void f(){
 C c1; // calls default constructor
 C c2[2]; // calls default constructor for both elements
 C* c3 = new C[2]; // calls default constructor for both array elements

 C_child c_ch; // when destructed calls destructor of s_ch and of C base (and in turn s)

 delete[] c3; // calls destructors on c3[0] and c3[1]
} // automatic variables are destroyed here -- i.e. c1, c2 and c_ch
```

```
class C{
 int i;
 string s;
};

void f(){
 C* c1 = new C;
 delete c1; // C has a destructor
}
```

```

class C{
 int m;
private:
 ~C(){} // not public destructor!
};

class C_container{
 C c;
};

void f(){
 C_container* c_cont = new C_container;
 delete c_cont; // Compile ERROR: C has no accessible destructor
}

```

## C++ 11

### C++ 11。

```

class C{
 int m;
public:
 ~C() = delete; // does NOT have implicit destructor
};

void f{
 C c1;
} // Compile ERROR: C has no destructor

```

。

```

class C{
 int m;
public:
 ~C() = default; // saying explicitly it does have implicit/empty destructor
};

void f(){
 C c1;
} // C has a destructor -- c1 properly destroyed

```

## C++ 11

<type\_traits> std::is\_destructible

```

class C1{ };
class C2{ public: ~C2() = delete };
class C3 : public C2{ };

using std::cout; using std::boolalpha; using std::endl;
using std::is_destructible;
cout << boolalpha << is_destructible<int>() << endl; // prints true
cout << boolalpha << is_destructible<C1>() << endl; // prints true
cout << boolalpha << is_destructible<C2>() << endl; // prints false
cout << boolalpha << is_destructible<C3>() << endl; // prints false

```

<https://riptutorial.com/zh-TW/cplusplus/topic/1476/>

# 100: C ++

C ++。

。

## Examples

Observer Pattern。

。 。 。

Gamma“”。

```
#include <iostream>
#include <vector>

class Subject;

class Observer
{
public:
 virtual ~Observer() = default;
 virtual void Update(Subject&) = 0;
};

class Subject
{
public:
 virtual ~Subject() = default;
 void Attach(Observer& o) { observers.push_back(&o); }
 void Detach(Observer& o)
 {
 observers.erase(std::remove(observers.begin(), observers.end(), &o));
 }
 void Notify()
 {
 for (auto* o : observers) {
 o->Update(*this);
 }
 }
private:
 std::vector<Observer*> observers;
};

class ClockTimer : public Subject
{
public:

 void SetTime(int hour, int minute, int second)
 {
 this->hour = hour;
 this->minute = minute;
 this->second = second;
 }
};
```



```

 Notify();
 }

 int GetHour() const { return hour; }
 int GetMinute() const { return minute; }
 int GetSecond() const { return second; }

private:
 int hour;
 int minute;
 int second;
};

class DigitalClock: public Observer
{
public:
 explicit DigitalClock(ClockTimer& s) : subject(s) { subject.Attach(*this); }
 ~DigitalClock() { subject.Detach(*this); }
 void Update(Subject& theChangedSubject) override
 {
 if (&theChangedSubject == &subject) {
 Draw();
 }
 }

 void Draw()
 {
 int hour = subject.GetHour();
 int minute = subject.GetMinute();
 int second = subject.GetSecond();

 std::cout << "Digital time is " << hour << ":"
 << minute << ":"
 << second << std::endl;
 }

private:
 ClockTimer& subject;
};

class AnalogClock: public Observer
{
public:
 explicit AnalogClock(ClockTimer& s) : subject(s) { subject.Attach(*this); }
 ~AnalogClock() { subject.Detach(*this); }
 void Update(Subject& theChangedSubject) override
 {
 if (&theChangedSubject == &subject) {
 Draw();
 }
 }

 void Draw()
 {
 int hour = subject.GetHour();
 int minute = subject.GetMinute();
 int second = subject.GetSecond();

 std::cout << "Analog time is " << hour << ":"
 << minute << ":"
 << second << std::endl;
 }
}

```

```
private:
 ClockTimer& subject;
};

int main()
{
 ClockTimer timer;

 DigitalClock digitalClock(timer);
 AnalogClock analogClock(timer);

 timer.SetTime(14, 41, 36);
}
```

```
Digital time is 14:41:36
Analog time is 14:41:36
```

1. DigitalClockAnalogClock **Subject** Attach()Detach() subject.Attach(\*this);  
subject.Detach(\*this);
2. vector<Observer\*> observers; ◦
3. **Observer** ◦ Update() **Subject** Update(Subject &)
4. Attach()Detach()Notify() ◦ Subject Subject::Attach (Observer&) void  
Subject::Detach(Observer&)void Subject::Notify() ◦
5. Concrete ◦
6. **Observer** ◦ subject.Attach(\*this); ◦
7. ◦

QtObserver ◦ ◦ Qt ◦ / ◦ // ◦ //QtMOC ◦

C ◦ Qt ◦

◦ ◦ ◦

1. ◦
2. Adapter ◦
3. Adaptee ◦
4. STLpushvector :: push\_back ◦

```
#include <iostream>

// Desired interface (Target)
class Rectangle
{
public:
```

```

 virtual void draw() = 0;
};

// Legacy component (Adaptee)
class LegacyRectangle
{
public:
 LegacyRectangle(int x1, int y1, int x2, int y2) {
 x1_ = x1;
 y1_ = y1;
 x2_ = x2;
 y2_ = y2;
 std::cout << "LegacyRectangle(x1,y1,x2,y2)\n";
 }
 void oldDraw() {
 std::cout << "LegacyRectangle: oldDraw(). \n";
 }
private:
 int x1_;
 int y1_;
 int x2_;
 int y2_;
};

// Adapter wrapper
class RectangleAdapter: public Rectangle, private LegacyRectangle
{
public:
 RectangleAdapter(int x, int y, int w, int h):
 LegacyRectangle(x, y, x + w, y + h) {
 std::cout << "RectangleAdapter(x,y,x+w,x+h)\n";
 }

 void draw() {
 std::cout << "RectangleAdapter: draw().\n";
 oldDraw();
 }
};

int main()
{
 int x = 20, y = 50, w = 300, h = 200;
 Rectangle *r = new RectangleAdapter(x,y,w,h);
 r->draw();
}

//Output:
//LegacyRectangle(x1,y1,x2,y2)
//RectangleAdapter(x,y,x+w,x+h)

```

1. Rectangle

2. Rectangle° °

```

Rectangle *r = new RectangleAdapter(x,y,w,h);
r->draw();

```

3. °

```
class RectangleAdapter: public Rectangle, private LegacyRectangle {
 ...
}
```

4. Adapter `RectangleAdapterLegacyRectangle` **BOTH** `Rectangle` `draw()` ◦

5. `LegacyRectangleRectangle` `draw()` `Adapter(RectangleAdapter)RectangleLegacyRectangle` `oldDraw()`

◦

```
class RectangleAdapter: public Rectangle, private LegacyRectangle {
public:
 RectangleAdapter(int x, int y, int w, int h):
 LegacyRectangle(x, y, x + w, y + h) {
 std::cout << "RectangleAdapter(x,y,x+w,x+h)\n";
 }

 void draw() {
 std::cout << "RectangleAdapter: draw().\n";
 oldDraw();
 }
};
```

◦ ◦ ◦

API◦ API◦

**Bridge** `Bridge` ◦ ◦

```
class Animal{
public:
 virtual std::shared_ptr<Animal> clone() const = 0;
 virtual std::string getname() const = 0;
};

class Bear: public Animal{
public:
 virtual std::shared_ptr<Animal> clone() const override
 {
 return std::make_shared<Bear>(*this);
 }
 virtual std::string getname() const override
 {
 return "bear";
 }
};

class Cat: public Animal{
public:
 virtual std::shared_ptr<Animal> clone() const override
 {
 return std::make_shared<Cat>(*this);
 }
 virtual std::string getname() const override
 {
 return "cat";
 }
};
```

```

 }
};

class AnimalFactory{
public:
 static std::shared_ptr<Animal> getAnimal(const std::string& name)
 {
 if (name == "bear")
 return std::make_shared<Bear>();
 if (name == "cat")
 return std::shared_ptr<Cat>();

 return nullptr;
 }
};

```

## Fluent API Builder

Builder Pattern. . .

[CEmail Builder](#) C++. Email .

```

#include <iostream>
#include <sstream>
#include <string>

using namespace std;

// Forward declaring the builder
class EmailBuilder;

class Email
{
public:
 friend class EmailBuilder; // the builder can access Email's privates

 static EmailBuilder make();

 string to_string() const {
 stringstream stream;
 stream << "from: " << m_from
 << "\nto: " << m_to
 << "\nsubject: " << m_subject
 << "\nbody: " << m_body;
 return stream.str();
 }

private:
 Email() = default; // restrict construction to builder

 string m_from;
 string m_to;
 string m_subject;
 string m_body;
};

```

```

class EmailBuilder
{
public:
 EmailBuilder& from(const string &from) {
 m_email.m_from = from;
 return *this;
 }

 EmailBuilder& to(const string &to) {
 m_email.m_to = to;
 return *this;
 }

 EmailBuilder& subject(const string &subject) {
 m_email.m_subject = subject;
 return *this;
 }

 EmailBuilder& body(const string &body) {
 m_email.m_body = body;
 return *this;
 }

 operator Email&&() {
 return std::move(m_email); // notice the move
 }

private:
 Email m_email;
};

EmailBuilder Email::make()
{
 return EmailBuilder();
}

// Bonus example!
std::ostream& operator <<(std::ostream& stream, const Email& email)
{
 stream << email.to_string();
 return stream;
}

int main()
{
 Email mail = Email::make().from("me@mail.com")
 .to("you@mail.com")
 .subject("C++ builders")
 .body("I like this API, don't you?");

 cout << mail << endl;
}

```

**C++** `std::move&&`

`operator Email&&()` ◦ ◦ `Email EmailBuilder::build() {...}` ◦

## Builder Pattern actoractor . .

```
void add_addresses(EmailBuilder& builder)
{
 builder.from("me@mail.com")
 .to("you@mail.com");
}

void compose_mail(EmailBuilder& builder)
{
 builder.subject("I know the subject")
 .body("And the body. Someone else knows the addresses.");
}

int main()
{
 EmailBuilder builder;
 add_addresses(builder);
 compose_mail(builder);

 Email mail = builder;
 cout << mail << endl;
}
```

---

. .

## Email . . setter . setter . .

**C ++** <https://riptutorial.com/zh-TW/cplusplus/topic/4335/c-plusplus>

# 101:

## Examples

```
#include <iostream>

long double operator"" _km(long double val)
{
 return val * 1000.0;
}

long double operator"" _mi(long double val)
{
 return val * 1609.344;
}

int main()
{
 std::cout << "3 km = " << 3.0_km << " m\n";
 std::cout << "3 mi = " << 3.0_mi << " m\n";
 return 0;
}
```

```
3 km = 3000 m
3 mi = 4828.03 m
```

## C++ 14

```
namespace std::literals::chrono_literals literals chrono_literals ◦ using namespace std::literals
using namespace std::chrono_literals using namespace std::literals::chrono_literals ◦
```

```
#include <chrono>
#include <iostream>

int main()
{
 using namespace std::literals::chrono_literals;

 std::chrono::nanoseconds t1 = 600ns;
 std::chrono::microseconds t2 = 42us;
 std::chrono::milliseconds t3 = 51ms;
 std::chrono::seconds t4 = 61s;
 std::chrono::minutes t5 = 88min;
 auto t6 = 2 * 0.5h;

 auto total = t1 + t2 + t3 + t4 + t5 + t6;

 std::cout.precision(13);
 std::cout << total.count() << " nanoseconds" << std::endl; // 8941051042600 nanoseconds
 std::cout << std::chrono::duration_cast<std::chrono::hours>(total).count()
 << " hours" << std::endl; // 2 hours
}
```

## C++ 14



```
namespace std::literals::string_literals literalsstring_literals ◦ using namespace std::literals
using namespace std::string_literals using namespace std::literals::string_literals◦
```

```
#include <codecvt>
#include <iostream>
#include <locale>
#include <string>

int main()
{
 using namespace std::literals::string_literals;

 std::string s = "hello world"s;
 std::u16string s16 = u"hello world"s;
 std::u32string s32 = U"hello world"s;
 std::wstring ws = L"hello world"s;

 std::cout << s << std::endl;

 std::wstring_convert<std::codecvt_utf8_utf16<char16_t>, char16_t> utf16conv;
 std::cout << utf16conv.to_bytes(s16) << std::endl;

 std::wstring_convert<std::codecvt_utf8_utf16<char32_t>, char32_t> utf32conv;
 std::cout << utf32conv.to_bytes(s32) << std::endl;

 std::wcout << ws << std::endl;
}
```

\0

```
std::string s1 = "foo\0\0bar"; // constructor from C-string: results in "foo"s
std::string s2 = "foo\0\0bar"s; // That string contains 2 '\0' in its middle
```

## C++ 14

```
namespace std::literals::complex_literals literalscomplex_literals ◦ using namespace
std::literals using namespace std::complex_literals using namespace
std::literals::complex_literals ◦
```

```
#include <complex>
#include <iostream>

int main()
{
 using namespace std::literals::complex_literals;

 std::complex<double> c = 2.0 + 1i; // {2.0, 1.}
 std::complex<float> cf = 2.0f + 1if; // {2.0f, 1.f}
 std::complex<long double> cl = 2.0L + 1il; // {2.0L, 1.L}

 std::cout << "abs" << c << " = " << abs(c) << std::endl; // abs(2,1) = 2.23607
 std::cout << "abs" << cf << " = " << abs(cf) << std::endl; // abs(2,1) = 2.23607
 std::cout << "abs" << cl << " = " << abs(cl) << std::endl; // abs(2,1) = 2.23607
}
```

## C++ 14

```
int number =0b0001'0101; // ==21
```

o

```
template< char FIRST, char... REST > struct binary
{
 static_assert(FIRST == '0' || FIRST == '1', "invalid binary digit");
 enum { value = ((FIRST - '0') << sizeof...(REST)) + binary<REST...>::value };
};

template<> struct binary<'0'> { enum { value = 0 } ; };
template<> struct binary<'1'> { enum { value = 1 } ; };

// raw literal operator
template< char... LITERAL > inline
constexpr unsigned int operator "" _b() { return binary<LITERAL...>::value ; }

// raw literal operator
template< char... LITERAL > inline
constexpr unsigned int operator "" _B() { return binary<LITERAL...>::value ; }

#include <iostream>

int main()
{
 std::cout << 10101_B << ", " << 011011000111_b << '\n' ; // prints 21, 1735
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2745/>

# 102: std :: integer\_sequence

std::integer\_sequence<Type, Values...>Type Type° ° ""°

## Examples

### std :: tuple

std::tuple<T...>° °

```
#include <array>
#include <iostream>
#include <string>
#include <tuple>
#include <utility>

// -----
// Example functions to be called:
void f(int i, std::string const& s) {
 std::cout << "f(" << i << ", " << s << ")\n";
}
void f(int i, double d, std::string const& s) {
 std::cout << "f(" << i << ", " << d << ", " << s << ")\n";
}
void f(char c, int i, double d, std::string const& s) {
 std::cout << "f(" << c << ", " << i << ", " << d << ", " << s << ")\n";
}
void f(int i, int j, int k) {
 std::cout << "f(" << i << ", " << j << ", " << k << ")\n";
}

// -----
// The actual function expanding the tuple:
template <typename Tuple, std::size_t... I>
void process(Tuple const& tuple, std::index_sequence<I...>) {
 f(std::get<I>(tuple)...);
}

// The interface to call. Sadly, it needs to dispatch to another function
// to deduce the sequence of indices created from std::make_index_sequence<N>
template <typename Tuple>
void process(Tuple const& tuple) {
 process(tuple, std::make_index_sequence<std::tuple_size<Tuple>::value>());
}

// -----
int main() {
 process(std::make_tuple(1, 3.14, std::string("foo")));
 process(std::make_tuple('a', 2, 2.71, std::string("bar")));
 process(std::make_pair(3, std::string("pair")));
 process(std::array<int, 3>{ 1, 2, 3 });
}
```

std::get<I>(object)std::tuple\_size<T>::value process()° °

std::integer\_sequence

```
#include <iostream>
#include <initializer_list>
#include <utility>

template <typename T, T... I>
void print_sequence(std::integer_sequence<T, I...>) {
 std::initializer_list<bool>{ bool(std::cout << I << ' ')... };
 std::cout << '\n';
}

template <int Offset, typename T, T... I>
void print_offset_sequence(std::integer_sequence<T, I...>) {
 print_sequence(std::integer_sequence<T, T(I + Offset)...>());
}

int main() {
 // explicitly specify sequences:
 print_sequence(std::integer_sequence<int, 1, 2, 3>());
 print_sequence(std::integer_sequence<char, 'f', 'o', 'o'>());

 // generate sequences:
 print_sequence(std::make_index_sequence<10>());
 print_sequence(std::make_integer_sequence<short, 10>());
 print_offset_sequence<'A'>(std::make_integer_sequence<char, 26>());
}
```

print\_sequence() std::initializer\_list<bool>[array]

◦ [gccclang gccvoid](#)

```
#include <algorithm>
#include <array>
#include <iostream>
#include <iterator>
#include <string>
#include <utility>

template <typename T, std::size_t... I>
std::array<T, sizeof...(I)> make_array(T const& value, std::index_sequence<I...>) {
 return std::array<T, sizeof...(I)>{ (I, value)... };
}

template <int N, typename T>
std::array<T, N> make_array(T const& value) {
 return make_array(value, std::make_index_sequence<N>());
}

int main() {
 auto array = make_array<20>(std::string("value"));
 std::copy(array.begin(), array.end(),
 std::ostream_iterator<std::string>(std::cout, " "));
 std::cout << "\n";
}
```

[std::integer\\_sequence](#) <https://riptutorial.com/zh-TW/cplusplus/topic/8315/std---integer-sequence>

---

# 103: std :: string

◦ `stringchar` ◦ C++ `stringstd1998` ◦

- `//`

```
std :: string s;
```

- `//const char *c-string`

```
std :: string s"Hello";
```

```
std :: string s = "";
```

- `//`

```
std :: string s1"Hello";
```

```
std :: string s2s1;
```

- `//`

```
std :: string s1"Hello";
```

```
std :: string s2s1,0,4; //s10s2
```

- `//`

```
std :: string s1"Hello World";
```

```
std :: string s2s1,5; //s15s2
```

- `//fillchar`

```
std :: string s5'a'; //saaaaa
```

- `//`

```
std :: string s1"Hello World";
```

```
std :: string s2s1.begins1.begin+ 5; //s15s2
```

```
std::stringstring iostream //◦
```

---

`const char *nullptr` ◦

```
std::string oops(nullptr);
std::cout << oops << "\n";
```

---

```
index >= size() atstd::out_of_range
```

```
operator[]index > size() index == size()
```

## C++ 11

1. `const` ;
2. `constCharT()` ◦

## C++ 11

1. `CharT()` ◦
2. ◦

---

**C++ 14** `"foo" "foo"s s const char* "foo"std::string "foo"` ◦

`std::string_literalsstd::literalss` ◦

## Examples

`std::string::substr` ◦

◦ `(0, str.length())`

```
std::string str = "Hello foo, bar and world!";
std::string newstr = str.substr(11); // "bar and world!"
```

◦

```
std::string str = "Hello foo, bar and world!";
std::string newstr = str.substr(15, 3); // "and"
```

`substr`

```
std::string str = "Hello foo, bar and world!";
std::string newstr = str.substr(); // "Hello foo, bar and world!"
```

---

`std::stringreplacestd::string` ◦

`replace`

```
//Define string
std::string str = "Hello foo, bar and world!";
std::string alternate = "Hello foobar";

//1)
str.replace(6, 3, "bar"); // "Hello bar, bar and world!"
```

```
//2)
str.replace(str.begin() + 6, str.end(), "nobody!"); //"Hello nobody!"

//3)
str.replace(19, 5, alternate, 6, 6); //"Hello foo, bar and foobar!"
```

## C++ 14

```
//4)
str.replace(19, 5, alternate, 6); //"Hello foo, bar and foobar!"
```

```
//5)
str.replace(str.begin(), str.begin() + 5, str.begin() + 6, str.begin() + 9);
//"foo foo, bar and world!"
```

```
//6)
str.replace(0, 5, 3, 'z'); //"zzz foo, bar and world!"
```

```
//7)
str.replace(str.begin() + 6, str.begin() + 9, 3, 'x'); //"Hello xxx, bar and world!"
```

## C++ 11

```
//8)
str.replace(str.begin(), str.begin() + 5, { 'x', 'y', 'z' }); //"xyz foo, bar and world!"
```

---

replacewithstr

```
std::string replaceString(std::string str,
 const std::string& replace,
 const std::string& with){
 std::size_t pos = str.find(replace);
 if (pos != std::string::npos)
 str.replace(pos, replace.length(), with);
 return str;
}
```

replacewithstr

```
std::string replaceStringAll(std::string str,
 const std::string& replace,
 const std::string& with) {
 if(!replace.empty()) {
 std::size_t pos = 0;
 while ((pos = str.find(replace, pos)) != std::string::npos) {
 str.replace(pos, replace.length(), with);
 pos += with.length();
 }
 }
 return str;
}
```

+=std::string ◦ +

```
std::string hello = "Hello";
std::string world = "world";
std::string helloworld = hello + world; // "Helloworld"
```

+=

```
std::string hello = "Hello";
std::string world = "world";
hello += world; // "Helloworld"
```

## C

```
std::string hello = "Hello";
std::string world = "world";
const char *comma = ", ";
std::string newhelloworld = hello + comma + world + "!"; // "Hello, world!"
```

push\_back() char

```
std::string s = "a, b, ";
s.push_back('c'); // "a, b, c"
```

append() +=

```
std::string app = "test and ";
app.append("test"); // "test and test"
```

std::string°

```
std::string str("Hello world!");
```

---

## []n

n°

std::string::operator[]°

```
char c = str[6]; // 'w'
```

---

n

n°

std::string::at std::out\_of\_range

```
char c = str.at(7); // 'o'
```

## C++ 11



```
char c = str.front(); // 'H'
```

```
char c = str.back(); // 'I'
```

### 1. `std::strtok` C++3

- `std::strtok` strings [strings](#) `strtok_s`
- `std::strtok` [Visual Studio](#)
- `std::strtok` `std::string` `const string` `S` `const char* S` `std::strtok` `std::string`

`std::strtok` ◦

```
// String to tokenize
std::string str{ "The quick brown fox" };
// Vector to store tokens
vector<std::string> tokens;

for (auto i = strtok(&str[0], " "); i != NULL; i = strtok(NULL, " "))
 tokens.push_back(i);
```

### 2. `std::istream_iterator` ◦ `std::string` `std::strtok` `const vector<string>`

```
// String to tokenize
const std::string str("The quick \tbrown \nfox");
std::istringstream is(str);
// Vector to store tokens
const std::vector<std::string> tokens = std::vector<std::string>(
 std::istream_iterator<std::string>(is),
 std::istream_iterator<std::string>());
```

### 3. `std::regex_token_iterator` `std::regex` ◦ ◦

## C++ 11

```
// String to tokenize
const std::string str{ "The ,qu\,ick ,\tbrown, fox" };
const std::regex re{ "\\s*(?:[^\s\\,]|\\\\\\\\.)*?\\s*(?:,|$)" };
// Vector to store tokens
const std::vector<std::string> tokens{
 std::sregex_token_iterator(str.begin(), str.end(), re, 1),
 std::sregex_token_iterator()
};
```

[regex\\_token\\_iterator](#) ◦

## `const char *`

`const char*` `std::stringc_str()` ◦ `std::string` `const` ◦

## C++ 17

```
data()char* std::string
```

## C++ 11

```
char* &s[0] ◦ C++ 11null ◦ &s[0]s&s.front()s ◦
```

## C++ 11

```
std::string str("This is a string.");
const char* cstr = str.c_str(); // cstr points to: "This is a string.\0"
const char* data = str.data(); // data points to: "This is a string.\0"
```

```
std::string str("This is a string.");

// Copy the contents of str to untie lifetime from the std::string object
std::unique_ptr<char []> cstr = std::make_unique<char[]>(str.size() + 1);

// Alternative to the line above (no exception safety):
// char* cstr_unsafe = new char[str.size() + 1];

std::copy(str.data(), str.data() + str.size(), cstr);
cstr[str.size()] = '\0'; // A null-terminator needs to be added

// delete[] cstr_unsafe;
std::cout << cstr.get();
```

[std::string::find](#) ◦ ◦ [std::string::npos](#)

```
std::string str = "Curiosity killed the cat";
auto it = str.find("cat");

if (it != std::string::npos)
 std::cout << "Found at position: " << it << '\n';
else
 std::cout << "Not found!\n";
```

## 21

---

```
find_first_of // Find first occurrence of characters
find_first_not_of // Find first absence of characters
find_last_of // Find last occurrence of characters
find_last_not_of // Find last absence of characters
```

◦

```
std::string str = "dog dog cat cat";
std::cout << "Found at position: " << str.find_last_of("gzx") << '\n';
```

## 6

◦ 6'g' ◦

/

<algorithm> <locale><utility> ◦

## C++ 11

◦ ◦ std::basic\_string std::string std::wstring std::vector std::list ◦

```
template <typename Sequence, // any basic_string, vector, list etc.
 typename Pred> // a predicate on the element (character) type
Sequence& trim(Sequence& seq, Pred pred) {
 return trim_start(trim_end(seq, pred), pred);
}
```

```
template <typename Sequence, typename Pred>
Sequence& trim_end(Sequence& seq, Pred pred) {
 auto last = std::find_if_not(seq.rbegin(),
 seq.rend(),
 pred);
 seq.erase(last.base(), seq.end());
 return seq;
}
```

```
template <typename Sequence, typename Pred>
Sequence& trim_start(Sequence& seq, Pred pred) {
 auto first = std::find_if_not(seq.begin(),
 seq.end(),
 pred);
 seq.erase(seq.begin(), first);
 return seq;
}
```

std::string std::isspace()

```
std::string& trim(std::string& str, const std::locale& loc = std::locale()) {
 return trim(str, [&loc](const char c){ return std::isspace(c, loc); });
}

std::string& trim_start(std::string& str, const std::locale& loc = std::locale()) {
 return trim_start(str, [&loc](const char c){ return std::isspace(c, loc); });
}

std::string& trim_end(std::string& str, const std::locale& loc = std::locale()) {
 return trim_end(str, [&loc](const char c){ return std::isspace(c, loc); });
}
```

std::wstring std::iswspace() ◦

```
template <typename Sequence, typename Pred>
Sequence trim_copy(Sequence seq, Pred pred) { // NOTE: passing seq by value
 trim(seq, pred);
 return seq;
}
```

== != < <= > >=std::string S

```
std::string str1 = "Foo";
std::string str2 = "Bar";

assert(!(str1 < str2));
assert(str > str2);
assert(!(str1 <= str2));
assert(str1 >= str2);
assert(!(str1 == str2));
assert(str1 != str2);
```

std::string::compare()◦

- operator ==

str1.length() == str2.length() true false ◦

- operator !=

str1.length() != str2.length() true false ◦

- operator <operator >

◦

- operator <=operator >=

◦

◦ str1str2 nm str1[i]str2[i]  $i = 0, 1, 2$  str2[i] ..maxnm ◦  $i$  nm ◦

---

<

```
std::string str1 = "Barr";
std::string str2 = "Bar";

assert(str2 < str1);
```

1. 'B' == 'B' - ◦
2. 'a' == 'a' - ◦
3. 'r' == 'r' - ◦
4. str2str1◦ str2 < str1 ◦

## std :: wstring

C++std::basic\_string◦ std::stringstd::wstring

- std::stringchar
- std::wstringwchar\_t

wstring\_convert

```
#include <string>
#include <codecvt>
#include <locale>

std::string input_str = "this is a -string-, which is a sequence based on the -char- type.";
std::wstring input_wstr = L"this is a -wide- string, which is based on the -wchar_t- type.";

// conversion
std::wstring str_turned_to_wstr =
std::wstring_convert<std::codecvt_utf8<wchar_t>>().from_bytes(input_str);

std::string wstr_turned_to_str =
std::wstring_convert<std::codecvt_utf8<wchar_t>>().to_bytes(input_wstr);
```

/

```
#include <string>
#include <codecvt>
#include <locale>

using convert_t = std::codecvt_utf8<wchar_t>;
std::wstring_convert<convert_t, wchar_t> strconverter;

std::string to_string(std::wstring wstr)
{
 return strconverter.to_bytes(wstr);
}

std::wstring to_wstring(std::string str)
{
 return strconverter.from_bytes(str);
}
```

```
std::wstring a_wide_string = to_wstring("Hello World!");
```

```
std::wstring_convert<std::codecvt_utf8<wchar_t>>().from_bytes("Hello World!").
```

---

charwchar\_t° wchar\_t2WindowsUTF-16Windows 2000UCS-2UTF-32° Linux° char16\_tchar32\_tC++ 11UTF16UTF32"" °

## std :: string\_view

C++ 17

C++ 17std::string\_view const char ° ° C++ 17

```
void foo(std::string const& s); // pre-C++17, single argument, could incur
 // allocation if caller's data was not in a string
 // (e.g. string literal or vector<char>)

void foo(const char* s, size_t len); // pre-C++17, two arguments, have to pass them
```

```

// both everywhere

void foo(const char* s); // pre-C++17, single argument, but need to call
// strlen()

template <class StringT>
void foo(StringT const& s); // pre-C++17, caller can pass arbitrary char data
// provider, but now foo() has to live in a header

```

```

void foo(std::string_view s); // post-C++17, single argument, tighter coupling
// zero copies regardless of how caller is storing
// the data

```

`std::string_view` ◦

`string_view` ◦

`std::string`

```

std::string str = "lllloooonnnngggg sssstttrrrriinnnggg"; //A really long string

//Bad way - 'string::substr' returns a new string (expensive if the string is long)
std::cout << str.substr(15, 10) << '\n';

//Good way - No copies are created!
std::string_view view = str;

// string_view::substr returns a new string_view
std::cout << view.substr(15, 10) << '\n';

```

## C++ 11

`std::string`

```

std::string str = "Hello World!";
for (auto c : str)
 std::cout << c;

```

“” for

```

std::string str = "Hello World!";
for (std::size_t i = 0; i < str.length(); ++i)
 std::cout << str[i];

```

/

`std::string` ◦

`"123abc"123` ◦

`std::ato*`C

```
std::string ten = "10";

double num1 = std::atof(ten.c_str());
int num2 = std::atoi(ten.c_str());
long num3 = std::atol(ten.c_str());
```

## C++ 11

```
long long num4 = std::atoll(ten.c_str());
```

0 ◦ 0“0”◦

std::sto\*std::string S◦

## C++ 11

```
std::string ten = "10";

int num1 = std::stoi(ten);
long num2 = std::stol(ten);
long long num3 = std::stoll(ten);

float num4 = std::stof(ten);
double num5 = std::stod(ten);
long double num6 = std::stold(ten);
```

std::ato\*◦ ◦ 000x0X

```
std::string ten = "10";
std::string ten_octal = "12";
std::string ten_hex = "0xA";

int num1 = std::stoi(ten, 0, 2); // Returns 2
int num2 = std::stoi(ten_octal, 0, 8); // Returns 10
long num3 = std::stol(ten_hex, 0, 16); // Returns 10
long num4 = std::stol(ten_hex); // Returns 0
long num5 = std::stol(ten_hex, 0, 0); // Returns 10 as it detects the leading 0x
```

## C++ 11<codecvt><locale>◦

```
#include <iostream>
#include <codecvt>
#include <locale>
#include <string>
using namespace std;

int main() {
 // converts between wstring and utf8 string
 wstring_convert<codecvt_utf8_utf16<wchar_t>> wchar_to_utf8;
 // converts between u16string and utf8 string
 wstring_convert<codecvt_utf8_utf16<char16_t>, char16_t> utf16_to_utf8;

 wstring wstr = L"foobar";
 string utf8str = wchar_to_utf8.to_bytes(wstr);
 wstring wstr2 = wchar_to_utf8.from_bytes(utf8str);
```

```

wcout << wstr << endl;
cout << utf8str << endl;
wcout << wstr2 << endl;

u16string u16str = u"foobar";
string utf8str2 = utf16_to_utf8.to_bytes(u16str);
u16string u16str2 = utf16_to_utf8.from_bytes(utf8str2);

return 0;
}

```

## Visual Studio 2015 `char16_twstring_convert`

```

using utf16_char = unsigned short;
wstring_convert<codecvt_utf8_utf16<utf16_char>, utf16_char> conv_utf8_utf16;

void strings::utf16_to_utf8(const std::u16string& utf16, std::string& utf8)
{
 std::basic_string<utf16_char> tmp;
 tmp.resize(utf16.length());
 std::copy(utf16.begin(), utf16.end(), tmp.begin());
 utf8 = conv_utf8_utf16.to_bytes(tmp);
}

void strings::utf8_to_utf16(const std::string& utf8, std::u16string& utf16)
{
 std::basic_string<utf16_char> tmp = conv_utf8_utf16.from_bytes(utf8);
 utf16.clear();
 utf16.resize(tmp.length());
 std::copy(tmp.begin(), tmp.end(), utf16.begin());
}

```

## C++ 14

### C++ 14 `std::mismatch`

```

std::string prefix = "foo";
std::string string = "foobar";

bool isPrefix = std::mismatch(prefix.begin(), prefix.end(),
 string.begin(), string.end()).first == prefix.end();

```

### C++ 14 `mismatch()` ◦

## C++ 14

`std::mismatch()`

```

bool isPrefix = prefix.size() <= string.size() &&
 std::mismatch(prefix.begin(), prefix.end(),
 string.begin(), string.end()).first == prefix.end();

```

## C++ 17

`std::string_view`



```
bool isPrefix(std::string_view prefix, std::string_view full)
{
 return prefix == full.substr(0, prefix.size());
}
```

## std :: string

std::ostringstream<< std::ostringstreamstd::ostringstream [std::ostringstream](#)std::string ◦

int

```
#include <sstream>

int main()
{
 int val = 4;
 std::ostringstream str;
 str << val;
 std::string converted = str.str();
 return 0;
}
```

```
template<class T>
std::string toString(const T& x)
{
 std::ostringstream ss;
 ss << x;
 return ss.str();
}
```

◦

```
std::ostream operator<<(std::ostream& out, const A& a)
{
 // write a string representation of a to out
 return out;
}
```

## C ++ 11

C ++ 11 [std::to\\_string](#) [std::to\\_wstring](#) ◦

```
std::string s = to_string(0x12f3); // after this the string s contains "4851"
```

[std :: string](#) <https://riptutorial.com/zh-TW/cplusplus/topic/488/std----string>

# 104: std :: iomanip

## Examples

### std ::

```
int val = 10;
// val will be printed to the extreme left end of the output console:
std::cout << val << std::endl;
// val will be printed in an output field of length 10 starting from right end of the field:
std::cout << std::setw(10) << val << std::endl;
```

```
10
 10
1234567890
```

- 
- **std::setw** **std :: iomanip** ◦

std::setw

```
std::setw(int n)
```

n

### std :: setprecision

out << setprecision(n) in >> setprecision(n) out in ◦ ◦

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <limits>
int main()
{
 const long double pi = std::acos(-1.L);
 std::cout << "default precision (6): " << pi << '\n'
 << "std::precision(10): " << std::setprecision(10) << pi << '\n'
 << "max precision: "
 << std::setprecision(std::numeric_limits<long double>::digits10 + 1)
 << pi << '\n';
}
//Output
//default precision (6): 3.14159
//std::precision(10): 3.141592654
//max precision: 3.141592653589793239
```

### std :: setfill

```
out << setfill(c) c ◦
```

```
std::ostream::fill◦
```

```
#include <iostream>
#include <iomanip>
int main()
{
 std::cout << "default fill: " << std::setw(10) << 42 << '\n'
 << "setfill('*'): " << std::setfill('*')
 << std::setw(10) << 42 << '\n';
}
//output::
//default fill: 42
//setfill('*'): *****42
```

## std :: setiosflags

```
out << setiosflags(mask) in >> setiosflags(mask) ◦
```

```
std::ios_base::fmtflags
```

- dec - **I/O**
- oct - **I/O**
- hex - **I/O**
- basefield - dec|oct|hex|0
- left -
- right -
- internal -
- adjustfield - left|right|internal ◦
- scientific -
- fixed -
- floatfield - scientific|fixed|(scientific|fixed)|0 ◦
- boolalpha - bool
- showbase - **I/O**
- showpoint -
- showpos - +
- skipws -
- unitbuf
- uppercase -

```
#include <iostream>
#include <string>
#include <iomanip>
int main()
{
 int l_iTemp = 47;
 std::cout << std::resetiosflags(std::ios_base::basefield);
 std::cout << std::setiosflags(std::ios_base::oct) << l_iTemp << std::endl;
 //output: 57
 std::cout << std::resetiosflags(std::ios_base::basefield);
```

```

std::cout<<std::setiosflags(std::ios_base::hex)<<l_iTemp<<std::endl;
//output: 2f
std::cout<<std::setiosflags(std::ios_base::uppercase)<<l_iTemp<<std::endl;
//output 2F
std::cout<<std::setfill('0')<<std::setw(12);
std::cout<<std::resetiosflags(std::ios_base::uppercase);
std::cout<<std::setiosflags(std::ios_base::right)<<l_iTemp<<std::endl;
//output: 00000000002f

std::cout<<std::resetiosflags(std::ios_base::basefield|std::ios_base::adjustfield);
std::cout<<std::setfill('.')<<std::setw(10);
std::cout<<std::setiosflags(std::ios_base::left)<<l_iTemp<<std::endl;
//output: 47.....

std::cout<<std::resetiosflags(std::ios_base::adjustfield)<<std::setfill('#');
std::cout<<std::setiosflags(std::ios_base::internal|std::ios_base::showpos);
std::cout<<std::setw(10)<<l_iTemp<<std::endl;
//output +#####47

double l_dTemp = -1.2;
double pi = 3.14159265359;
std::cout<<pi<<" "<<l_dTemp<<std::endl;
//output +3.14159 -1.2
std::cout<<std::setiosflags(std::ios_base::showpoint)<<l_dTemp<<std::endl;
//output -1.20000
std::cout<<setiosflags(std::ios_base::scientific)<<pi<<std::endl;
//output: +3.141593e+00
std::cout<<std::resetiosflags(std::ios_base::floatfield);
std::cout<<setiosflags(std::ios_base::fixed)<<pi<<std::endl;
//output: +3.141593
bool b = true;
std::cout<<std::setiosflags(std::ios_base::unitbuf|std::ios_base::boolalpha)<<b;
//output: true
return 0;
}

```

**std::iosmanip** <https://riptutorial.com/zh-TW/cplusplus/topic/6936/std---iosmanip>

# 105: std ::

std::any°

## Examples

```
std::any an_object{ std::string("hello world") };
if (an_object.has_value()) {
 std::cout << std::any_cast<std::string>(an_object) << '\n';
}

try {
 std::any_cast<int>(an_object);
} catch (std::bad_any_cast&) {
 std::cout << "Wrong type\n";
}

std::any_cast<std::string&>(an_object) = "42";
std::cout << std::any_cast<std::string>(an_object) << '\n';
```

```
hello world
Wrong type
42
```

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/7894/std--->

# 106: std ::Modifiers

std::forward\_list ◦ C ◦ std::list ◦

◦ erase\_after ◦ std :: forward\_listContainerSize==AllocatorAwareContainerSequenceContainer ◦

## Examples

```
#include <forward_list>
#include <string>
#include <iostream>

template<typename T>
std::ostream& operator<<(std::ostream& s, const std::forward_list<T>& v) {
 s.put(' ');
 char comma[3] = {'\0', ' ', '\0'};
 for (const auto& e : v) {
 s << comma << e;
 comma[0] = ',';
 }
 return s << ' ';
}

int main()
{
 // c++11 initializer list syntax:
 std::forward_list<std::string> words1 {"the", "frogurt", "is", "also", "cursed"};
 std::cout << "words1: " << words1 << '\n';

 // words2 == words1
 std::forward_list<std::string> words2(words1.begin(), words1.end());
 std::cout << "words2: " << words2 << '\n';

 // words3 == words1
 std::forward_list<std::string> words3(words1);
 std::cout << "words3: " << words3 << '\n';

 // words4 is {"Mo", "Mo", "Mo", "Mo", "Mo"}
 std::forward_list<std::string> words4(5, "Mo");
 std::cout << "words4: " << words4 << '\n';
}
```

```
words1: [the, frogurt, is, also, cursed]
words2: [the, frogurt, is, also, cursed]
words3: [the, frogurt, is, also, cursed]
words4: [Mo, Mo, Mo, Mo, Mo]
```

operator=

assign

get\_allocator

|                       |                     |
|-----------------------|---------------------|
| -----                 | -----               |
| front                 |                     |
| -----                 | -----               |
|                       |                     |
| before_begin <b>n</b> |                     |
| cbefore_begin         |                     |
| begin                 |                     |
| cbegin                | <b>const</b>        |
| end                   |                     |
| cend                  |                     |
|                       |                     |
| empty                 |                     |
| max_size              |                     |
|                       |                     |
| clear                 |                     |
| insert_after          |                     |
| emplace_after         |                     |
| erase_after           |                     |
| push_front            |                     |
| emplace_front         |                     |
| pop_front             |                     |
| resize                |                     |
| swap                  |                     |
|                       |                     |
| merge                 |                     |
| splice_after          | <b>forward_list</b> |
| remove                |                     |
| remove_if             |                     |
| reverse               |                     |
| unique                |                     |
| sort                  |                     |

[std ::Modifiers](https://riptutorial.com/zh-TW/cplusplus/topic/9703/std---modifiers) <https://riptutorial.com/zh-TW/cplusplus/topic/9703/std---modifiers>



# 107: std ::

## Examples

`std::atomic`

`std::memory_order`

`std :: atomic` TriviallyCopyable type T. `std::atomic` TriviallyCopyable type T. `std::atomic`

`std :: atomic`

1. `bool` typedef `std::atomic<T>`

| Typedef                       |                                      |
|-------------------------------|--------------------------------------|
| <code>std::atomic_bool</code> | <code>std::atomic&lt;bool&gt;</code> |

2 typedef

| Typedef                           |                                                    |
|-----------------------------------|----------------------------------------------------|
| <code>std::atomic_char</code>     | <code>std::atomic&lt;char&gt;</code>               |
| <code>std::atomic_char</code>     | <code>std::atomic&lt;char&gt;</code>               |
| <code>std::atomic_schar</code>    | <code>std::atomic&lt;signed char&gt;</code>        |
| <code>std::atomic_uchar</code>    | <code>std::atomic&lt;unsigned char&gt;</code>      |
| <code>std::atomic_short</code>    | <code>std::atomic&lt;short&gt;</code>              |
| <code>std::atomic_ushort</code>   | <code>std::atomic&lt;unsigned short&gt;</code>     |
| <code>std::atomic_int</code>      | <code>std::atomic&lt;int&gt;</code>                |
| <code>std::atomic_uint</code>     | <code>std::atomic&lt;unsigned int&gt;</code>       |
| <code>std::atomic_long</code>     | <code>std::atomic&lt;long&gt;</code>               |
| <code>std::atomic_ulong</code>    | <code>std::atomic&lt;unsigned long&gt;</code>      |
| <code>std::atomic_llong</code>    | <code>std::atomic&lt;long long&gt;</code>          |
| <code>std::atomic_ullong</code>   | <code>std::atomic&lt;unsigned long long&gt;</code> |
| <code>std::atomic_char16_t</code> | <code>std::atomic&lt;char16_t&gt;</code>           |
| <code>std::atomic_char32_t</code> | <code>std::atomic&lt;char32_t&gt;</code>           |
| <code>std::atomic_wchar_t</code>  | <code>std::atomic&lt;wchar_t&gt;</code>            |
| <code>std::atomic_int8_t</code>   | <code>std::atomic&lt;std::int8_t&gt;</code>        |

| Typedef                                 |                                                     |
|-----------------------------------------|-----------------------------------------------------|
| <code>std::atomic_uint8_t</code>        | <code>std::atomic&lt;std::uint8_t&gt;</code>        |
| <code>std::atomic_int16_t</code>        | <code>std::atomic&lt;std::int16_t&gt;</code>        |
| <code>std::atomic_uint16_t</code>       | <code>std::atomic&lt;std::uint16_t&gt;</code>       |
| <code>std::atomic_int32_t</code>        | <code>std::atomic&lt;std::int32_t&gt;</code>        |
| <code>std::atomic_uint32_t</code>       | <code>std::atomic&lt;std::uint32_t&gt;</code>       |
| <code>std::atomic_int64_t</code>        | <code>std::atomic&lt;std::int64_t&gt;</code>        |
| <code>std::atomic_uint64_t</code>       | <code>std::atomic&lt;std::uint64_t&gt;</code>       |
| <code>std::atomic_int_least8_t</code>   | <code>std::atomic&lt;std::int_least8_t&gt;</code>   |
| <code>std::atomic_uint_least8_t</code>  | <code>std::atomic&lt;std::uint_least8_t&gt;</code>  |
| <code>std::atomic_int_least16_t</code>  | <code>std::atomic&lt;std::int_least16_t&gt;</code>  |
| <code>std::atomic_uint_least16_t</code> | <code>std::atomic&lt;std::uint_least16_t&gt;</code> |
| <code>std::atomic_int_least32_t</code>  | <code>std::atomic&lt;std::int_least32_t&gt;</code>  |
| <code>std::atomic_uint_least32_t</code> | <code>std::atomic&lt;std::uint_least32_t&gt;</code> |
| <code>std::atomic_int_least64_t</code>  | <code>std::atomic&lt;std::int_least64_t&gt;</code>  |
| <code>std::atomic_uint_least64_t</code> | <code>std::atomic&lt;std::uint_least64_t&gt;</code> |
| <code>std::atomic_int_fast8_t</code>    | <code>std::atomic&lt;std::int_fast8_t&gt;</code>    |
| <code>std::atomic_uint_fast8_t</code>   | <code>std::atomic&lt;std::uint_fast8_t&gt;</code>   |
| <code>std::atomic_int_fast16_t</code>   | <code>std::atomic&lt;std::int_fast16_t&gt;</code>   |
| <code>std::atomic_uint_fast16_t</code>  | <code>std::atomic&lt;std::uint_fast16_t&gt;</code>  |
| <code>std::atomic_int_fast32_t</code>   | <code>std::atomic&lt;std::int_fast32_t&gt;</code>   |
| <code>std::atomic_uint_fast32_t</code>  | <code>std::atomic&lt;std::uint_fast32_t&gt;</code>  |
| <code>std::atomic_int_fast64_t</code>   | <code>std::atomic&lt;std::int_fast64_t&gt;</code>   |
| <code>std::atomic_uint_fast64_t</code>  | <code>std::atomic&lt;std::uint_fast64_t&gt;</code>  |
| <code>std::atomic_intptr_t</code>       | <code>std::atomic&lt;std::intptr_t&gt;</code>       |
| <code>std::atomic_uintptr_t</code>      | <code>std::atomic&lt;std::uintptr_t&gt;</code>      |
| <code>std::atomic_size_t</code>         | <code>std::atomic&lt;std::size_t&gt;</code>         |
| <code>std::atomic_ptrdiff_t</code>      | <code>std::atomic&lt;std::ptrdiff_t&gt;</code>      |
| <code>std::atomic_intmax_t</code>       | <code>std::atomic&lt;std::intmax_t&gt;</code>       |
| <code>std::atomic_uintmax_t</code>      | <code>std::atomic&lt;std::uintmax_t&gt;</code>      |

`std :: atomic_int`

```

#include <iostream> // std::cout
#include <atomic> // std::atomic, std::memory_order_relaxed
#include <thread> // std::thread

std::atomic_int foo (0);

void set_foo(int x) {
 foo.store(x, std::memory_order_relaxed); // set value atomically
}

void print_foo() {
 int x;
 do {
 x = foo.load(std::memory_order_relaxed); // get value atomically
 } while (x==0);
 std::cout << "foo: " << x << '\n';
}

int main ()
{
 std::thread first (print_foo);
 std::thread second (set_foo,10);
 first.join();
 //second.join();
 return 0;
}
//output: foo: 10

```

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/7475/std--->

# 108: std ::

## Examples

OptionalsMaybe C ++ 17 `std::optional<int>`

Optionals

```
std::optionalpair<bool, T>
```

**VS**

```
nullptr - optional
```

**vs Sentinel**

◦ `0-1 nullptr` ◦ `00`

**VS** `std::pair<bool, T>`

`bool`

◦ `optional<T>`

**C ++ 17** `nullptr` ◦ `int` ◦ `std::optional`

```
struct Person ◦ ◦ Personpetstd::optional
```

```
#include <iostream>
#include <optional>
#include <string>

struct Animal {
 std::string name;
};

struct Person {
 std::string name;
 std::optional<Animal> pet;
};

int main() {
 Person person;
 person.name = "John";

 if (person.pet) {
 std::cout << person.name << "'s pet's name is " <<
 person.pet->name << std::endl;
 }
 else {
```

```

 std::cout << person.name << " is alone." << std::endl;
 }
}

```

## C++ 17

- ◦
  - AppDelegate \*App::get\_delegate() nullptr ◦
  - ◦
- ◦
  - unsigned shortest\_path\_distance(Vertex a, Vertex b) unsigned shortest\_path\_distance(Vertex a, Vertex b) ◦
- bool ◦
  - std::pair<int, bool> parse(const std::string &str) undefined int bool false ◦

John Fluffy Furball ◦ Person::pet\_with\_name() John ◦ John Whiskers std::nullopt ◦

```

#include <iostream>
#include <optional>
#include <string>
#include <vector>

struct Animal {
 std::string name;
};

struct Person {
 std::string name;
 std::vector<Animal> pets;

 std::optional<Animal> pet_with_name(const std::string &name) {
 for (const Animal &pet : pets) {
 if (pet.name == name) {
 return pet;
 }
 }
 return std::nullopt;
 }
};

int main() {
 Person john;
 john.name = "John";

 Animal fluffy;
 fluffy.name = "Fluffy";
 john.pets.push_back(fluffy);

 Animal furball;
 furball.name = "Furball";
 john.pets.push_back(furball);

 std::optional<Animal> whiskers = john.pet_with_name("Whiskers");
 if (whiskers) {
 std::cout << "John has a pet named Whiskers." << std::endl;
 }
}

```

```

else {
 std::cout << "Whiskers must not belong to John." << std::endl;
}
}

```

```

std::optional<float> divide(float a, float b) {
 if (b!=0.f) return a/b;
 return {};
}

```

a/b ◦

```

template<class Range, class Pred>
auto find_if(Range&& r, Pred&& p) {
 using std::begin; using std::end;
 auto b = begin(r), e = end(r);
 auto r = std::find_if(b, e , p);
 using iterator = decltpe(r);
 if (r==e)
 return std::optional<iterator>();
 return std::optional<iterator>(r);
}
template<class Range, class T>
auto find(Range&& r, T const& t) {
 return find_if(std::forward<Range>(r), [&t](auto&& x){return x==t;});
}

```

find( some\_range, 7 )some\_range7 ◦ find\_iffind\_if ◦

◦

```

if (find(vec, 7)) {
 // code
}

```

```

if (auto oit = find(vec, 7)) {
 vec.erase(*oit);
}

```

/◦

## value\_or

```

void print_name(std::ostream& os, std::optional<std::string> const& name) {
 std::cout << "Name is: " << name.value_or("<name missing>") << '\n';
}

```

value\_oroptional ◦

maybe-null. "" ◦

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/2423/std--->

# 109: std ::

- `std::map``std::multimap` <map> ◦
- `std::map``std::multimap` ◦ `std::multimap` ◦
- `std::map``std::multimap` `std::map``std::multimap` ◦
- ◦ `search()` `insert()` `erase()`  $\Theta \log n$  ◦ `std::unordered_map` ◦
- `size()` `empty()`  $\Theta 1$  ◦

## Examples

`std::map`(key, value) ◦

`std::map`

```
std::map < std::string, int > ranking { std::make_pair("stackoverflow", 2),
 std::make_pair("docs-beta", 1) };
```

`std::map`

```
ranking["stackoverflow"]=2;
ranking["docs-beta"]=1;
```

`stackoverflow`2. ◦

`std::map`

```
std::cout << ranking["stackoverflow"] << std::endl;
```

`operator[]` ◦ `const std::map` ◦ `find()` `at()` ◦

## C++ 11

`at()` `std::map`

```
std::cout << ranking.at("stackoverflow") << std::endl;
```

`at()` `std::out_of_range` ◦

`std::map` `std::multimap`

## C++ 11

```
// Example using begin()
std::multimap < int, std::string > mmp { std::make_pair(2, "stackoverflow"),
```

```

 std::make_pair(1, "docs-beta"),
 std::make_pair(2, "stackexchange") };

auto it = mmp.begin();
std::cout << it->first << " : " << it->second << std::endl; // Output: "1 : docs-beta"
it++;
std::cout << it->first << " : " << it->second << std::endl; // Output: "2 : stackoverflow"
it++;
std::cout << it->first << " : " << it->second << std::endl; // Output: "2 : stackexchange"

// Example using rbegin()
std::map < int, std::string > mp { std::make_pair(2, "stackoverflow"),
 std::make_pair(1, "docs-beta"),
 std::make_pair(2, "stackexchange") };

auto it2 = mp.rbegin();
std::cout << it2->first << " : " << it2->second << std::endl; // Output: "2 : stackoverflow"
it2++;
std::cout << it2->first << " : " << it2->second << std::endl; // Output: "1 : docs-beta"

```

## std::mapstd::multimap

std::mapstd::multimap {key, value}std::make\_pair(key, value) std::map

```

std::multimap < int, std::string > mmp { std::make_pair(2, "stackoverflow"),
 std::make_pair(1, "docs-beta"),
 std::make_pair(2, "stackexchange") };

// 1 docs-beta
// 2 stackoverflow
// 2 stackexchange

std::map < int, std::string > mp { std::make_pair(2, "stackoverflow"),
 std::make_pair(1, "docs-beta"),
 std::make_pair(2, "stackexchange") };

// 1 docs-beta
// 2 stackoverflow

```

o

```

// From std::map or std::multimap iterator
std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {6, 8}, {3, 4},
 {6, 7} };
 // {1, 2}, {3, 4}, {3, 4}, {6, 5}, {6, 8}, {6, 7}, {8, 9}
auto it = mmp.begin();
std::advance(it,3); //moved cursor on first {6, 5}
std::map< int, int > mp(it, mmp.end()); // {6, 5}, {8, 9}

//From std::pair array
std::pair< int, int > arr[10];
arr[0] = {1, 3};
arr[1] = {1, 5};
arr[2] = {2, 5};
arr[3] = {0, 1};
std::map< int, int > mp(arr,arr+4); //{0 , 1}, {1, 3}, {2, 5}

//From std::vector of std::pair
std::vector< std::pair<int, int> > v{ {1, 5}, {5, 1}, {3, 6}, {3, 2} };
std::multimap< int, int > mp(v.begin(), v.end());
 // {1, 5}, {3, 6}, {3, 2}, {5, 1}

```



```
std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
mmp.clear(); //empty multimap
```

```
std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
// {1, 2}, {3, 4}, {3, 4}, {6, 5}, {6, 7}, {8, 9}
auto it = mmp.begin();
std::advance(it,3); // moved cursor on first {6, 5}
mmp.erase(it); // {1, 2}, {3, 4}, {3, 4}, {6, 7}, {8, 9}
```

```
std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
// {1, 2}, {3, 4}, {3, 4}, {6, 5}, {6, 7}, {8, 9}
auto it = mmp.begin();
auto it2 = it;
it++; //moved first cursor on first {3, 4}
std::advance(it2,3); //moved second cursor on first {6, 5}
mmp.erase(it,it2); // {1, 2}, {6, 5}, {6, 7}, {8, 9}
```

```
std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
// {1, 2}, {3, 4}, {3, 4}, {6, 5}, {6, 7}, {8, 9}
mmp.erase(6); // {1, 2}, {3, 4}, {3, 4}, {8, 9}
```

pred

```
std::map<int,int> m;
auto it = m.begin();
while (it != m.end())
{
 if (pred(*it))
 it = m.erase(it);
 else
 ++it;
}
```

std::map **map**◦

```
std::map< std::string, size_t > fruits_count;
```

- insert()std::map ◦ pair

```
fruits_count.insert({"grapes", 20});
fruits_count.insert(make_pair("orange", 30));
fruits_count.insert(pair<std::string, size_t>("banana", 40));
fruits_count.insert(map<std::string, size_t>::value_type("cherry", 50));
```

insert()boolpair

- booltrue ◦
- key◦ boolfalse ◦

```
auto success = fruits_count.insert({"grapes", 20});
if (!success.second) { // we already have 'grapes' in the map
 success.first->second += 20; // access the iterator to update the value
}
```

```
}
```

- `std::map`

```
fruits_count["apple"] = 10;
```

- `std::map::operator[]`

- `insert()` ◦ `insertvoid`

```
fruits_count.insert({{"apricot", 1}, {"jackfruit", 1}, {"lime", 1}, {"mango", 7}});
```

- `insert()` `value_type`

```
std::map< std::string, size_t > fruit_list{ {"lemon", 0}, {"olive", 0}, {"plum", 0}};
fruits_count.insert(fruit_list.begin(), fruit_list.end());
```

```
std::map<std::string, size_t> fruits_count;
std::string fruit;
while(std::cin >> fruit){
 // insert an element with 'fruit' as key and '1' as value
 // (if the key is already stored in fruits_count, insert does nothing)
 auto ret = fruits_count.insert({fruit, 1});
 if(!ret.second){ // 'fruit' is already in the map
 ++ret.first->second; // increment the counter
 }
}
```

**Olog n** `std::map` ◦

## C++ 11

`make_pair()` `emplace()` `pair`

```
std::map< std::string , int > runs;
runs.emplace("Babe Ruth", 714);
runs.insert(make_pair("Barry Bonds", 762));
```

`emplace_hint()` `hint` ◦ `hint` ◦ `emplace()`

```
std::map< std::string , int > runs;
auto it = runs.emplace("Barry Bonds", 762); // get iterator to the inserted element
// the next element will be before "Barry Bonds", so it is inserted before 'it'
runs.emplace_hint(it, "Babe Ruth", 714);
```

## `std::map` `std::multimap`

`std::map` `std::multimap`

```
std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
```

```

//Range based loop - since C++11
for(const auto &x: mmp)
 std::cout<< x.first <<" "<< x.second << std::endl;

//Forward iterator for loop: it would loop through first element to last element
//it will be a std::map< int, int >::iterator
for (auto it = mmp.begin(); it != mmp.end(); ++it)
 std::cout<< it->first <<" "<< it->second << std::endl; //Do something with iterator

//Backward iterator for loop: it would loop through last element to first element
//it will be a std::map< int, int >::reverse_iterator
for (auto it = mmp.rbegin(); it != mmp.rend(); ++it)
 std::cout<< it->first <<" "<< it->second << std::endl; //Do something with iterator

```

std::mapstd::multimapautoSO°

## std :: mapstd :: multimap

std::mapstd::multimap°

- find()° end()°

```

std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
auto it = mmp.find(6);
if(it!=mmp.end())
 std::cout << it->first << ", " << it->second << std::endl; //prints: 6, 5
else
 std::cout << "Value does not exist!" << std::endl;

it = mmp.find(66);
if(it!=mmp.end())
 std::cout << it->first << ", " << it->second << std::endl;
else
 std::cout << "Value does not exist!" << std::endl; // This line would be executed.

```

- std::mapstd::multimapcount()° std::mapcount()01° std::multimap count()1°

```

std::map< int , int > mp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
if(mp.count(3) > 0) // 3 exists as a key in map
 std::cout << "The key exists!" << std::endl; // This line would be executed.
else
 std::cout << "The key does not exist!" << std::endl;

```

findmultimaps°

- std::multimap° equal\_range()std::pair° end()°

```

auto eqr = mmp.equal_range(6);
auto st = eqr.first, en = eqr.second;
for(auto it = st; it != en; ++it){
 std::cout << it->first << ", " << it->second << std::endl;
}

// prints: 6, 5
// 6, 7

```

std::mapempty() truefalse map◦ size()std::map

```
std::map<std::string , int> rank {"facebook.com", 1} , {"google.com", 2}, {"youtube.com", 3};
if(!rank.empty()){
 std::cout << "Number of elements in the rank map: " << rank.size() << std::endl;
}
else{
 std::cout << "The rank map is empty" << std::endl;
}
```

◦

```
#include <string>
#include <map>
std::map<std::string, size_t> fruits_count;
```

std::string size\_t ◦

◦ ◦

- multimap

```
#include <string>
#include <map>
#include <cstring>
struct StrLess {
 bool operator()(const std::string& a, const std::string& b) {
 return strncmp(a.c_str(), b.c_str(), 8)<0;
 //compare only up to 8 first characters
 }
}
std::map<std::string, size_t, StrLess> fruits_count2;
```

StrLessfalse ◦

Multimap◦ ◦

```
#include <string>
#include <map>
std::multimap<std::string, size_t> fruits_count;
std::multimap<std::string, size_t, StrLess> fruits_count2;
```

◦ ◦ ◦

```
#include <string>
#include <unordered_map>
std::unordered_map<std::string, size_t> fruits_count;
```

◦ unordered\_map◦

**std :: map**

copiableassignable ◦ ◦ std::less<KeyType> <◦

```
struct CmpMyType
{
 bool operator()(MyType const& lhs, MyType const& rhs) const
 {
 // ...
 }
};
```

C ++“compare” ◦ compare(X,X) Xfalse ◦ CmpMyType() (a, b) trueCmpMyType() (b, a) falsefalse◦

————

◦

fxfyxyxy◦ ◦

C ++<◦

```
X a;
X b;
```

| Condition:            | Test: | Result |
|-----------------------|-------|--------|
| a is equivalent to b: | a < b | false  |
| a is equivalent to b  | b < a | false  |
| a is less than b      | a < b | true   |
| a is less than b      | b < a | false  |
| b is less than a      | a < b | false  |
| b is less than a      | b < a | true   |

/◦

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/681/std--->

# 110: std ::

## Examples

Pair ◦ std::make\_pair ◦

firstsecond ◦

```
#include <iostream>
#include <utility>

int main()
{
 std::pair<int,int> p = std::make_pair(1,2); //Creating the pair
 std::cout << p.first << " " << p.second << std::endl; //Accessing the elements

 //We can also create a pair and assign the elements later
 std::pair<int,int> p1;
 p1.first = 3;
 p1.second = 4;
 std::cout << p1.first << " " << p1.second << std::endl;

 //We can also create a pair using a constructor
 std::pair<int,int> p2 = std::pair<int,int>(5, 6);
 std::cout << p2.first << " " << p2.second << std::endl;

 return 0;
}
```

lhsrhs

- operator==lhsrhs ◦ lhs.first == rhs.firstlhs.second == rhs.second true false

```
std::pair<int, int> p1 = std::make_pair(1, 2);
std::pair<int, int> p2 = std::make_pair(2, 2);

if (p1 == p2)
 std::cout << "equals";
else
 std::cout << "not equal"//statement will show this, because they are not identical
```

- operator!=lhsrhs ◦ true lhs.first != rhs.first **OR** lhs.second != rhs.second false ◦
- operator<lhs.first<rhs.first true ◦ rhs.first<lhs.firstfalse ◦ lhs.second<rhs.secondtrue false ◦
- operator<= **return** !(rhs<lhs)
- operator>rhs<lhs

- `operator>=` returns `!(lhs<rhs)`

- `operator<`

```
#include <iostream>
#include <utility>
#include <vector>
#include <algorithm>
#include <string>

int main()
{
 std::vector<std::pair<int, std::string>> v = { {2, "baz"},
 {2, "bar"},
 {1, "foo"} };

 std::sort(v.begin(), v.end());

 for(const auto& p: v) {
 std::cout << "(" << p.first << "," << p.second << ") ";
 //output: (1,foo) (2,bar) (2,baz)
 }
}
```

`std` :: <https://riptutorial.com/zh-TW/cplusplus/topic/4834/std--->

# 111: std ::

◦ ◦

```
; C-arrays std::array◦
```

```
std::vector#include <vector><vector>◦
```

```
std::vector◦ std::vector<std::vector<int> > ◦
```

## Examples

### std :: vector

`std::vector`

### C++ 11

```
std::vector<int> v{ 1, 2, 3 }; // v becomes {1, 2, 3}

// Different from std::vector<int> v(3, 6)
std::vector<int> v{ 3, 6 }; // v becomes {3, 6}
```

```
// Different from std::vector<int> v{3, 6} in C++11
std::vector<int> v(3, 6); // v becomes {6, 6, 6}

std::vector<int> v(4); // v becomes {0, 0, 0, 0}
```

v2

```
std::vector<int> v(v2);
std::vector<int> v = v2;
```

### C++ 11

v2

```
std::vector<int> v(std::move(v2));
std::vector<int> v = std::move(v2);
```

v

```
// from another vector
std::vector<int> v(v2.begin(), v2.begin() + 3); // v becomes {v2[0], v2[1], v2[2]}

// from an array
int z[] = { 1, 2, 3, 4 };
std::vector<int> v(z, z + 3); // v becomes {1, 2, 3}

// from a list
```



```
std::list<int> list1{ 1, 2, 3 };
std::vector<int> v(list1.begin(), list1.end()); // v becomes {1, 2, 3}
```

## C++ 11

### Iterator move-construction `std::make_move_iterator` v

```
// from another vector
std::vector<int> v(std::make_move_iterator(v2.begin()),
 std::make_move_iterator(v2.end()));

// from a list
std::list<int> list1{ 1, 2, 3 };
std::vector<int> v(std::make_move_iterator(list1.begin()),
 std::make_move_iterator(list1.end()));
```

### `assign()` `std::vector`

```
v.assign(4, 100); // v becomes {100, 100, 100, 100}

v.assign(v2.begin(), v2.begin() + 3); // v becomes {v2[0], v2[1], v2[2]}

int z[] = { 1, 2, 3, 4 };
v.assign(z + 1, z + 4); // v becomes {2, 3, 4}
```

/

```
struct Point {
 double x, y;
 Point(double x, double y) : x(x), y(y) {}
};
std::vector<Point> v;
Point p(10.0, 2.0);
v.push_back(p); // p is copied into the vector.
```

## C++ 11

```
std::vector<Point> v;
v.emplace_back(10.0, 2.0); // The arguments are passed to the constructor of the
 // given type (here Point). The object is constructed
 // in the vector, avoiding a copy.
```

`std::vector` `push_front()` ◦ ◦ `std::list` `std::deque` ◦

```
std::vector<int> v{ 1, 2, 3 };
v.insert(v.begin(), 9); // v now contains {9, 1, 2, 3}
```

## C++ 11

```
std::vector<int> v{ 1, 2, 3 };
v.emplace(v.begin()+1, 9); // v now contains {1, 9, 2, 3}
```

```
std::vector<int> v(4); // contains: 0, 0, 0, 0
std::vector<int> v2(2, 10); // contains: 10, 10
v.insert(v.begin()+2, v2.begin(), v2.end()); // contains: 0, 0, 10, 10, 0, 0
```

```
std::vector<int> v(4); // contains: 0, 0, 0, 0
int a [] = {1, 2, 3}; // contains: 1, 2, 3
v.insert(v.begin()+1, a, a+sizeof(a)/sizeof(a[0])); // contains: 0, 1, 2, 3, 0, 0, 0
```

### reserve()

```
std::vector<int> v;
v.reserve(100);
for(int i = 0; i < 100; ++i)
 v.emplace_back(i);
```

### resize() 200°

## std :: vector

std::vector ° v

```
std::vector<int> v;
```

## C++ 11

```
// Range based for
for(const auto& value: v) {
 std::cout << value << "\n";
}

// Using a for loop with iterator
for(auto it = std::begin(v); it != std::end(v); ++it) {
 std::cout << *it << "\n";
}

// Using for_each algorithm, using a function or functor:
void fun(int const& value) {
 std::cout << value << "\n";
}

std::for_each(std::begin(v), std::end(v), fun);

// Using for_each algorithm. Using a lambda:
std::for_each(std::begin(v), std::end(v), [](int const& value) {
 std::cout << value << "\n";
});
```

## C++ 11

```
// Using a for loop with iterator
```

```
for(std::vector<int>::iterator it = std::begin(v); it != std::end(v); ++it) {
 std::cout << *it << "\n";
}
```

```
// Using a for loop with index
for(std::size_t i = 0; i < v.size(); ++i) {
 std::cout << v[i] << "\n";
}
```

---

## C++ 14

```
// There is no standard way to use range based for for this.
// See below for alternatives.

// Using for_each algorithm
// Note: Using a lambda for clarity. But a function or functor will work
std::for_each(std::rbegin(v), std::rend(v), [](auto const& value) {
 std::cout << value << "\n";
});

// Using a for loop with iterator
for(auto rit = std::rbegin(v); rit != std::rend(v); ++rit) {
 std::cout << *rit << "\n";
}
```

```
// Using a for loop with index
for(std::size_t i = 0; i < v.size(); ++i) {
 std::cout << v[v.size() - 1 - i] << "\n";
}
```

;◦ begin()end()◦

## C++ 14

```
template<class C>
struct ReverseRange {
 C c; // could be a reference or a copy, if the original was a temporary
 ReverseRange(C&& cin): c(std::forward<C>(cin)) {}
 ReverseRange(ReverseRange&&)=default;
 ReverseRange& operator=(ReverseRange&&)=delete;
 auto begin() const {return std::rbegin(c);}
 auto end() const {return std::rend(c);}
};

// C is meant to be deduced, and perfect forwarded into
template<class C>
ReverseRange<C> make_ReverseRange(C&& c) {return {std::forward<C>(c)};}

int main() {
 std::vector<int> v { 1,2,3,4};
 for(auto const& value: make_ReverseRange(v)) {
 std::cout << value << "\n";
 }
}
```

# const

C++ 11 `cbegin()` `cend()` `const` `const`

C++ 11

```
// forward iteration
for (auto pos = v.cbegin(); pos != v.cend(); ++pos) {
 // type of pos is vector<T>::const_iterator
 // *pos = 5; // Compile error - can't write via const iterator
}

// reverse iteration
for (auto pos = v.crbegin(); pos != v.crend(); ++pos) {
 // type of pos is vector<T>::const_iterator
 // *pos = 5; // Compile error - can't write via const iterator
}

// expects Functor::operand() (T&)
for_each(v.begin(), v.end(), Functor());

// expects Functor::operand() (const T&)
for_each(v.cbegin(), v.cend(), Functor())
```

C++ 17

`as_const`

```
for (auto const& e : std::as_const(v)) {
 std::cout << e << '\n';
}
```

C++

C++ 14

```
template <class T>
constexpr std::add_const_t<T>& as_const(T& t) noexcept {
 return t;
}
```

---

`std::vector` C++ `std::vector` C++ `std::vector`

`std::vector`

- 
- 

---

operator `[]at()`

`std::vector<bool> const` ◦

`[]at()` `[]at()` ◦ `index < 0` `index >= size()` `at()` `std::out_of_range` ◦

**C++11** ◦

**C++11**

```
std::vector<int> v{ 1, 2, 3 };
```

```
// using []
int a = v[1]; // a is 2
v[1] = 4; // v now contains { 1, 4, 3 }

// using at()
int b = v.at(2); // b is 3
v.at(2) = 5; // v now contains { 1, 4, 5 }
int c = v.at(3); // throws std::out_of_range exception
```

`at()` `[]` ◦ `[]` ◦ ◦

```
for (std::size_t i = 0; i < v.size(); ++i) {
 v[i] = 1;
}
```

`operator[]` `iCPU` ◦

`front()` `back()` ◦ `[]`

```
std::vector<int> v{ 4, 5, 6 }; // In pre-C++11 this is more verbose

int a = v.front(); // a is 4, v.front() is equivalent to v[0]
v.front() = 3; // v now contains {3, 5, 6}
int b = v.back(); // b is 6, v.back() is equivalent to v[v.size() - 1]
v.back() = 7; // v now contains {3, 5, 7}
```

`front()` `back()` ◦ `front()` `back()` `empty()` ◦ `'empty'`

```
int main ()
{
 std::vector<int> v;
 int sum (0);

 for (int i=1;i<=10;i++) v.push_back(i); //create and initialize the vector

 while (!v.empty()) //loop through until the vector tests to be empty
 {
 sum += v.back(); //keep a running total
 v.pop_back(); //pop out the element which removes it from the vector
 }

 std::cout << "total: " << sum << '\n'; //output the total to the user

 return 0;
}
```

```
}
```

110. 'empty'.

## C++ 11

`data()` `std::vector` C

```
std::vector<int> v{ 1, 2, 3, 4 }; // v contains {1, 2, 3, 4}
int* p = v.data(); // p points to 1
*p = 4; // v now contains {4, 2, 3, 4}
++p; // p points to 2
*p = 3; // v now contains {4, 3, 3, 4}
p[1] = 2; // v now contains {4, 3, 2, 4}
*(p + 2) = 1; // v now contains {4, 3, 2, 1}
```

## C++ 11

C++ 11 `front()` `data()`

```
std::vector<int> v(4);
int* ptr = &(v.front()); // or &v[0]
```

operator& pre-C++ 11 `std::addressof`.

“`std::vector`” **Iterators**.

## C++ 11

```
std::vector<int> v{ 4, 5, 6 };

auto it = v.begin();
int i = *it; // i is 4
++it;
i = *it; // i is 5
*it = 6; // v contains { 4, 6, 6 }
auto e = v.end(); // e points to the element after the end of v. It can be
// used to check whether an iterator reached the end of the vector:
++it;
it == v.end(); // false, it points to the element at position 2 (with value 6)
++it;
it == v.end(); // true
```

`std::vector<T> T* S`.

◦ `vector/vector` `vector`.

## C++ 11

```
std::vector<int> v{ 1, 2, 3 };
int* p = v.data() + 1; // p points to 2
```

```
v.insert(v.begin(), 0); // p is now invalid, accessing *p is a undefined behavior.
p = v.data() + 1; // p points to 1
v.reserve(10); // p is now invalid, accessing *p is a undefined behavior.
p = v.data() + 1; // p points to 1
v.erase(v.begin()); // p is now invalid, accessing *p is a undefined behavior.
```

## std::vectorC

std::vectorCC° °

### C++ 11

```
std::vector<int> v{ 1, 2, 3 };
int* p = v.data();
```

C++.data()°

### C++ 11

```
int* p = &v[0]; // combine subscript operator and 0 literal
int* p = &v.front(); // explicitly reference the first element
```

v[0]v.front()°

push\_back resize °

```
std::vector<int> v;
int* p = v.data();
v.resize(42); // internal memory location changed; value of p is now invalid
```

/

std::vector° /°

/

- vector capacity/

```
vector<int> v(5); // Vector has a size of 5; capacity is unknown.
int *p1 = &v[0];
v.push_back(2); // p1 may have been invalidated, since the capacity was unknown.

v.reserve(20); // Capacity is now at least 20.
int *p2 = &v[0];
v.push_back(4); // p2 is *not* invalidated, since the size of `v` is now 7.
v.insert(v.end(), 30, 9); // Inserts 30 elements at the end. The size exceeds the
// requested capacity of 20, so `p2` is (probably) invalidated.
int *p3 = &v[0];
v.reserve(v.capacity() + 20); // Capacity exceeded, thus `p3` is invalid.
```

### C++ 11

```

auto old_cap = v.capacity();
v.shrink_to_fit();
if(old_cap != v.capacity())
 // Iterators were invalidated.

```

- `/o end`

```

vector<int> v(5);
v.reserve(20); // Capacity is at least 20.
int *p1 = &v[0];
int *p2 = &v[3];
v.insert(v.begin() + 2, 5, 0); // `p2` is invalidated, but since the capacity
// did not change, `p1` remains valid.
int *p3 = &v[v.size() - 1];
v.push_back(10); // The capacity did not change, so `p3` and `p1` remain valid.

```

- `/o end`

```

vector<int> v(10);
int *p1 = &v[0];
int *p2 = &v[5];
v.erase(v.begin() + 3, v.end()); // `p2` is invalid, but `p1` remains valid.

```

- `operator= copymove clear() vector/o`

---

```

std::vector<int> v{ 1, 2, 3 };
v.pop_back(); // v becomes {1, 2}

```

---

```

std::vector<int> v{ 1, 2, 3 };
v.clear(); // v becomes an empty vector

```

---

```

std::vector<int> v{ 1, 2, 3, 4, 5, 6 };
v.erase(v.begin() + 3); // v becomes {1, 2, 3, 5, 6}

```

vector [std :: list](#) ◦

---

```

std::vector<int> v{ 1, 2, 3, 4, 5, 6 };
v.erase(v.begin() + 1, v.begin() + 5); // v becomes {1, 6}

```

◦ ◦

`erase` ◦ `std::remove` `erase` ◦ [std :: list](#) ◦



```
std::vector<int> v{ 1, 1, 2, 2, 3, 3 };
int value_to_remove = 2;
v.erase(std::remove(v.begin(), v.end(), value_to_remove), v.end()); // v becomes {1, 1, 3, 3}
```

```
// std::remove_if needs a function, that takes a vector element as argument and returns true,
// if the element shall be removed
bool _predicate(const int& element) {
 return (element > 3); // This will cause all elements to be deleted that are larger than 3
}
...
std::vector<int> v{ 1, 2, 3, 4, 5, 6 };
v.erase(std::remove_if(v.begin(), v.end(), _predicate), v.end()); // v becomes {1, 2, 3}
```

## lambda

### C++ 11

```
std::vector<int> v{ 1, 2, 3, 4, 5, 6 };
v.erase(std::remove_if(v.begin(), v.end(),
 [](auto& element){return element > 3;}), v.end()
);
```

```
std::vector<int> v{ 1, 2, 3, 4, 5, 6 };
std::vector<int>::iterator it = v.begin();
while (it != v.end()) {
 if (condition)
 it = v.erase(it); // after erasing, 'it' will be set to the next element in v
 else
 ++it; // manually set 'it' to the next element in v
}
```

it<sup>◦</sup> remove\_if<sup>◦</sup>

```
std::vector<int> v{ -1, 0, 1, 2, 3, 4, 5, 6 };
typedef std::vector<int>::reverse_iterator rev_itr;
rev_itr it = v.rbegin();

while (it != v.rend()) { // after the loop only '0' will be in v
 int value = *it;
 if (value) {
 ++it;
 // See explanation below for the following line.
 it = rev_itr(v.erase(it.base()));
 } else
 ++it;
}
```

- [itbase<sup>◦</sup>](#)

- `vector::erase(iterator)`◦
- `reverse_iterator::reverse_iterator(iterator)`◦

```
it = rev_itr(v.erase(it.base()))it v;it ◦
```

---

```
v.clear()capacity()◦
```

```
std::vector<int>().swap(v);
```

## C++ 11

```
shrink_to_fit()
```

```
v.shrink_to_fit();
```

```
shrink_to_fit◦
```

## std :: vector

```
<algorithm>std::findstd::vector◦
```

```
std::findoperator==◦ ◦
```

```
std::findstd::vector::end conststd::vector::end ◦
```

## C++ 11

```
static const int arr[] = {5, 4, 3, 2, 1};
std::vector<int> v (arr, arr + sizeof(arr) / sizeof(arr[0]));

std::vector<int>::iterator it = std::find(v.begin(), v.end(), 4);
std::vector<int>::difference_type index = std::distance(v.begin(), it);
// `it` points to the second element of the vector, `index` is 1

std::vector<int>::iterator missing = std::find(v.begin(), v.end(), 10);
std::vector<int>::difference_type index_missing = std::distance(v.begin(), missing);
// `missing` is v.end(), `index_missing` is 5 (ie. size of the vector)
```

## C++ 11

```
std::vector<int> v { 5, 4, 3, 2, 1 };

auto it = std::find(v.begin(), v.end(), 4);
auto index = std::distance(v.begin(), it);
// `it` points to the second element of the vector, `index` is 1

auto missing = std::find(v.begin(), v.end(), 10);
auto index_missing = std::distance(v.begin(), missing);
// `missing` is v.end(), `index_missing` is 5 (ie. size of the vector)
```

```
binary_search◦
```

---

```
std::find_if ◦ std::find std::find_if ◦ bool
```

## C++ 11

```
bool isEven(int val) {
 return (val % 2 == 0);
}

struct moreThan {
 moreThan(int limit) : _limit(limit) {}

 bool operator()(int val) {
 return val > _limit;
 }

 int _limit;
};

static const int arr[] = {1, 3, 7, 8};
std::vector<int> v (arr, arr + sizeof(arr) / sizeof(arr[0]));

std::vector<int>::iterator it = std::find_if(v.begin(), v.end(), isEven);
// `it` points to 8, the first even element

std::vector<int>::iterator missing = std::find_if(v.begin(), v.end(), moreThan(10));
// `missing` is v.end(), as no element is greater than 10
```

## C++ 11

```
// find the first value that is even
std::vector<int> v = {1, 3, 7, 8};
auto it = std::find_if(v.begin(), v.end(), [](int val){return val % 2 == 0;});
// `it` points to 8, the first even element

auto missing = std::find_if(v.begin(), v.end(), [](int val){return val > 10;});
// `missing` is v.end(), as no element is greater than 10
```

## std::vector

```
std::beginstd::endstd::vector
```

## C++ 11

```
int values[5] = { 1, 2, 3, 4, 5 }; // source array

std::vector<int> v(std::begin(values), std::end(values)); // copy array to new vector

for(auto &x: v)
 std::cout << x << " ";
std::cout << std::endl;
```

1 2 3 4 5

```
int main(int argc, char* argv[]) {
 // convert main arguments into a vector of strings.
 std::vector<std::string> args(argv, argv + argc);
```

```
}
```

## C++ 11 initializer\_list <>

```
initializer_list<int> arr = { 1,2,3,4,5 };
vector<int> vec1 {arr};

for (auto & i : vec1)
 cout << i << endl;
```

### 23.3.7 vector<bool>bool C++ vector<bool>

- vector<bool>bool C API
- at() operator []bool bool std::vector<bool>

## C++ 11

```
std::vector<bool> v = {true, false};
for (auto &b: v) { } // error
```

bool&operator []at()vector<bool>

```
void f(bool& b);
f(v[0]); // error
f(*v.begin()); // error
```

std::vector<bool> n° n° 18° 8° 11°

vector<bool> ° ° [xxxxxxxx]°

std::vector<char>8

## C++ 11

```
std::vector<char> trad_vect = {true, false, false, false, true, false, true, true};
```

```
[0,0,0,0,0,0,0,1], [0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,1], [0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,1], [0,0,0,0,0,0,0,1]
```

std::vector<bool>8

## C++ 11

```
std::vector<bool> optimized_vect = {true, false, false, false, true, false, true, true};
```

[1,0,0,0,1,0,1,1]

std::vector<bool> 88std::vector<bool> 1° ° vector<bool>CAPIAPI°

1. size() ° size0Convenience empty()true

```
vector<int> v = { 1, 2, 3 }; // size is 3
const vector<int>::size_type size = v.size();
cout << size << endl; // prints 3
cout << boolalpha << v.empty() << endl; // prints false
```

## 2.0

```
vector<int> v; // size is 0
cout << v.size() << endl; // prints 0
```

3. `push_back()` `insert()` `resize()` ◦

4. `pop_back()` `N pop_back()` `erase()` `clear()` ◦

## 5. VectorRAM

```
vector<int> v;
const vector<int>::size_type max_size = v.max_size();
cout << max_size << endl; // prints some large number
v.resize(max_size); // probably won't work
v.push_back(1); // definitely won't work
```

int

```
// !!!bad!!!evil!!!
vector<int> v_bad(N, 1); // constructs large N size vector
for(int i = 0; i < v_bad.size(); ++i) { // size is not supposed to be int!
 do_something(v_bad[i]);
}
```

◦ /◦ ◦

1. `capacity()` ◦

```
vector<int> v = { 1, 2, 3 }; // size is 3, capacity is >= 3
const vector<int>::size_type capacity = v.capacity();
cout << capacity << endl; // prints number >= 3
```

2. `reserve( N )` N

```
// !!!bad!!!evil!!!
vector<int> v_bad;
for(int i = 0; i < 10000; ++i) {
 v_bad.push_back(i); // possibly lot of reallocations
}

// good
vector<int> v_good;
v_good.reserve(10000); // good! only one allocation
for(int i = 0; i < 10000; ++i) {
 v_good.push_back(i); // no allocations needed anymore
}
```

3.

```
shrink_to_fit()◦
```

```
vector<int> v = { 1, 2, 3, 4, 5 }; // size is 5, assume capacity is 6
v.shrink_to_fit(); // capacity is 5 (or possibly still 6)
cout << boolalpha << v.capacity() == v.size() << endl; // prints likely true (but
possibly false)
```

## Vector◦ 21.5 - ◦ ◦

```
vector<int> v; // capacity is possibly (but not guaranteed) to be 0
v.push_back(1); // capacity is some starter value, likely 1
v.clear(); // size is 0 but capacity is still same as before!

v = { 1, 2, 3, 4 }; // size is 4, and lets assume capacity is 4.
v.push_back(5); // capacity grows - let's assume it grows to 6 (1.5 factor)
v.push_back(6); // no change in capacity
v.push_back(7); // capacity grows - let's assume it grows to 9 (1.5 factor)
// and so on
v.pop_back(); v.pop_back(); v.pop_back(); v.pop_back(); // capacity stays the same
```

```
std::vector::insert()
```

```
std::vector<int> a = {0, 1, 2, 3, 4};
std::vector<int> b = {5, 6, 7, 8, 9};

a.insert(a.end(), b.begin(), b.end());
```

```
insert()◦
```

## C++ 11

```
std::begin() std::end()
```

```
a.insert(std::end(a), std::begin(b), std::end(b));
```

```
b◦ ◦
```

```
if (b.size() < a.size())
 a.insert(a.end(), b.begin(), b.end());
else
 b.insert(b.end(), a.begin(), a.end());
```

```
std::vector◦
```

```
// Initialize a vector with 100 elements
std::vector<int> v(100);

// The vector's capacity is always at least as large as its size
auto const old_capacity = v.capacity();
// old_capacity >= 100

// Remove half of the elements
v.erase(v.begin() + 50, v.end()); // Reduces the size from 100 to 50 (v.size() == 50),
```

```
// but not the capacity (v.capacity() == old_capacity)
```

◦ ◦ ◦

```
std::vector<int>(v).swap(v);
```

## C++ 11

### C++ 11 shrink\_to\_fit()

```
v.shrink_to_fit();
```

shrink\_to\_fit()◦

<algorithm>◦

<◦

std::sort()

```
std::vector<int> v;
// add some code here to fill v with some elements
std::sort(v.begin(), v.end());
```

std::lower\_bound()◦ std::find() std::find()◦

```
// search the vector for the first element with value 42
std::vector<int>::iterator it = std::lower_bound(v.begin(), v.end(), 42);
if (it != v.end() && *it == 42) {
 // we found the element!
}
```

std::lower\_bound()◦

```
int const new_element = 33;
v.insert(std::lower_bound(v.begin(), v.end(), new_element), new_element);
```

push\_back() std::sort()◦ ◦

std::lower\_bound()◦ std::upper\_bound()

```
v.insert(std::upper_bound(v.begin(), v.end(), new_element), new_element);
```

std::equal\_range()

```
std::pair<std::vector<int>::iterator,
 std::vector<int>::iterator> rg = std::equal_range(v.begin(), v.end(), 42);
std::vector<int>::iterator lower_bound = rg.first;
std::vector<int>::iterator upper_bound = rg.second;
```

`std::binary_search()`

```
bool exists = std::binary_search(v.begin(), v.end(), value_to_find);
```

## C++ 11

### C++ 11. ◦

```
#include <vector>
#include <iostream>

// If the compiler is unable to perform named return value optimization (NRVO)
// and elide the move altogether, it is required to move from v into the return value.
std::vector<int> fillVector(int a, int b) {
 std::vector<int> v;
 v.reserve(b-a+1);
 for (int i = a; i <= b; i++) {
 v.push_back(i);
 }
 return v; // implicit move
}

int main() { // declare and fill vector
 std::vector<int> vec = fillVector(1, 10);

 // print vector
 for (auto value : vec)
 std::cout << value << " "; // this will print "1 2 3 4 5 6 7 8 9 10 "

 std::cout << std::endl;

 return 0;
}
```

## C++ 11

### C++ 11 copy elision. ◦

```
#include <vector>
#include <iostream>

// passing a std::vector by reference
void fillVectorFrom_By_Ref(int a, int b, std::vector<int> &v) {
 assert(v.empty());
 v.reserve(b-a+1);
 for (int i = a; i <= b; i++) {
 v.push_back(i);
 }
}

int main() { // declare vector
 std::vector<int> vec;

 // fill vector
 fillVectorFrom_By_Ref(1, 10, vec);
 // print vector
 for (std::vector<int>::const_iterator it = vec.begin(); it != vec.end(); ++it)
```



```

 std::cout << *it << " "; // this will print "1 2 3 4 5 6 7 8 9 10 "
 std::cout << std::endl;
 return 0;
}

```

`std::max_element` `std::min_element` `<algorithm>` `v.end()`

```

std::vector<int> v = {5, 2, 8, 10, 9};
int maxElementIndex = std::max_element(v.begin(),v.end()) - v.begin();
int maxElement = *std::max_element(v.begin(), v.end());

int minElementIndex = std::min_element(v.begin(),v.end()) - v.begin();
int minElement = *std::min_element(v.begin(), v.end());

std::cout << "maxElementIndex:" << maxElementIndex << ", maxElement:" << maxElement << '\n';
std::cout << "minElementIndex:" << minElementIndex << ", minElement:" << minElement << '\n';

```

maxElementIndex3maxElement10  
minElementIndex1minElement2

## C++ 11

`std::minmax_element<algorithm>`

```

std::vector<int> v = {5, 2, 8, 10, 9};
auto minmax = std::minmax_element(v.begin(), v.end());

std::cout << "minimum element: " << *minmax.first << '\n';
std::cout << "maximum element: " << *minmax.second << '\n';

```

2  
10

2D

340

```

std::vector<std::vector<int> > matrix(3, std::vector<int>(4));

```

## C++ 11

。

```

std::vector<std::vector<int>> matrix = { {0,1,2,3},
 {4,5,6,7},
 {8,9,10,11}
};

```

2D

```

int var = matrix[0][2];

```

◦

```
for(int i = 0; i < 3; ++i)
{
 for(int j = 0; j < 4; ++j)
 {
 std::cout << matrix[i][j] << std::endl;
 }
}
```

## C++ 11

```
for(auto& row: matrix)
{
 for(auto& col : row)
 {
 std::cout << col << std::endl;
 }
}
```

◦

◦ ◦

**std ::** <https://riptutorial.com/zh-TW/cplusplus/topic/511/std--->

# 112: std ::

Variant<sup>union</sup> ◦

◦

std::tuple std::optional ◦

std::get std::get\_if ◦ std::visit ◦ if constexpr visit visit visit ◦

## Examples

### std :: variant

int string ◦

```
std::variant< int, std::string > var;
```

```
var = "hello"s;
```

std::visit

```
// Prints "hello\n":
visit([](auto&& e) {
 std::cout << e << '\n';
}, var);
```

lambda ◦

```
auto str = std::get<std::string>(var);
```

◦ get\_if

```
auto* str = std::get_if<std::string>(&var);
```

nullptr ◦

◦ ◦

◦ union ◦

◦

◦

```
template<class F>
```

```

struct pseudo_method {
 F f;
 // enable C++17 class type deduction:
 pseudo_method(F&& fin):f(std::move(fin)) {}

 // Koenig lookup operator->*, as this is a pseudo-method it is appropriate:
 template<class Variant> // maybe add SFINAE test that LHS is actually a variant.
 friend decltype(auto) operator->*(Variant&& var, pseudo_method const& method) {
 // var->*method returns a lambda that perfect forwards a function call,
 // behaving like a method pointer basically:
 return [&](auto&&...args)->decltype(auto) {
 // use visit to get the type of the variant:
 return std::visit(
 [&](auto&& self)->decltype(auto) {
 // decltype(x)(x) is perfect forwarding in a lambda:
 return method.f(decltype(self)(self), decltype(args)(args)...);
 },
 std::forward<Var>(var)
);
 };
 }
};

```

operator->\*Variant ◦

```

// C++17 class type deduction to find template argument of `print` here.
// a pseudo-method lambda should take `self` as its first argument, then
// the rest of the arguments afterwards, and invoke the action:
pseudo_method print = [](auto&& self, auto&&...args)->decltype(auto) {
 return decltype(self)(self).print(decltype(args)(args)...);
};

```

print

```

struct A {
 void print(std::ostream& os) const {
 os << "A";
 }
};
struct B {
 void print(std::ostream& os) const {
 os << "B";
 }
};

```

◦

```

std::variant<A,B> var = A{};

(var->*print)(std::cout);

```

A::print(std::cout) ◦ B{}var B::print(std::cout) ◦

## C

```

struct C {};

```

```
std::variant<A,B,C> var = A{};
(var->*print) (std::cout);
```

C.print(std::cout)◦

print printf constexpr ◦

boost::variantstd::variant ◦

## `std :: variant`

◦

```
struct A {};
struct B { B()=default; B(B const&)=default; B(int){}; };
struct C { C()=delete; C(int) {}; C(C const&)=default; };
struct D { D(std::initializer_list<int>) {}; D(D const&)=default; D()=default; };

std::variant<A,B> var_ab0; // contains a A()
std::variant<A,B> var_ab1 = 7; // contains a B(7)
std::variant<A,B> var_ab2 = var_ab1; // contains a B(7)
std::variant<A,B,C> var_abc0{ std::in_place_type<C>, 7 }; // contains a C(7)
std::variant<C> var_c0; // illegal, no default ctor for C
std::variant<A,D> var_ad0(std::in_place_type<D>, {1,3,3,4}); // contains D{1,3,3,4}
std::variant<A,D> var_ad1(std::in_place_index<0>); // contains A{}
std::variant<A,D> var_ad2(std::in_place_index<1>, {1,3,3,4}); // contains D{1,3,3,4}
```

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/5239/std--->

# 113: std ::

|               |  |
|---------------|--|
| class T       |  |
| std::size_t N |  |

```
std::array#include <array><array>°
```

## Examples

### std :: array

**std::array<T, N> T NT**

Tstd::array

```
// 1) Using aggregate-initialization
std::array<int, 3> a{ 0, 1, 2 };
// or equivalently
std::array<int, 3> a = { 0, 1, 2 };

// 2) Using the copy constructor
std::array<int, 3> a{ 0, 1, 2 };
std::array<int, 3> a2(a);
// or equivalently
std::array<int, 3> a2 = a;

// 3) Using the move constructor
std::array<int, 3> a = std::array<int, 3>{ 0, 1, 2 };
```

**std::array<T, N> T NT**

Tstd::array

```
struct A { int values[3]; }; // An aggregate type

// 1) Using aggregate initialization with brace elision
// It works only if T is an aggregate type!
std::array<A, 2> a{ 0, 1, 2, 3, 4, 5 };
// or equivalently
std::array<A, 2> a = { 0, 1, 2, 3, 4, 5 };

// 2) Using aggregate initialization with brace initialization of sub-elements
std::array<A, 2> a{ A{ 0, 1, 2 }, A{ 3, 4, 5 } };
// or equivalently
std::array<A, 2> a = { A{ 0, 1, 2 }, A{ 3, 4, 5 } };

// 3)
std::array<A, 2> a{{ { 0, 1, 2 }, { 3, 4, 5 } }};
// or equivalently
```

```

std::array<A, 2> a = {{ { 0, 1, 2 }, { 3, 4, 5 } }};

// 4) Using the copy constructor
std::array<A, 2> a{ 1, 2, 3 };
std::array<A, 2> a2(a);
// or equivalently
std::array<A, 2> a2 = a;

// 5) Using the move constructor
std::array<A, 2> a = std::array<A, 2>{ 0, 1, 2, 3, 4, 5 };

```

## at (pos)

pos° posstd::out\_of\_range°

### O1.

```

#include <array>

int main()
{
 std::array<int, 3> arr;

 // write values
 arr.at(0) = 2;
 arr.at(1) = 4;
 arr.at(2) = 6;

 // read values
 int a = arr.at(0); // a is now 2
 int b = arr.at(1); // b is now 4
 int c = arr.at(2); // c is now 6

 return 0;
}

```

## 2 operator[pos]

pos° pos° at(pos)°

### O1.

```

#include <array>

int main()
{
 std::array<int, 3> arr;

 // write values
 arr[0] = 2;
 arr[1] = 4;
 arr[2] = 6;

 // read values
 int a = arr[0]; // a is now 2
 int b = arr[1]; // b is now 4
 int c = arr[2]; // c is now 6
}

```

```
 return 0;
}
```

### 3 std::get<pos>

pos° pos°

O1°

```
#include <array>

int main()
{
 std::array<int, 3> arr;

 // write values
 std::get<0>(arr) = 2;
 std::get<1>(arr) = 4;
 std::get<2>(arr) = 6;

 // read values
 int a = std::get<0>(arr); // a is now 2
 int b = std::get<1>(arr); // b is now 4
 int c = std::get<2>(arr); // c is now 6

 return 0;
}
```

### 4 front ()

° front () °

O1°

Cc.front()\*c.begin() °

```
#include <array>

int main()
{
 std::array<int, 3> arr{ 2, 4, 6 };

 int a = arr.front(); // a is now 2

 return 0;
}
```

### 5 back ()

° back () °

O1°



```

#include <array>

int main()
{
 std::array<int, 3> arr{ 2, 4, 6 };

 int a = arr.back(); // a is now 6

 return 0;
}

```

## 6 data()

◦ range [data(); data() + size())data()◦

O1◦

```

#include <iostream>
#include <cstring>
#include <array>

int main ()
{
 const char* cstr = "Test string";
 std::array<char, 12> arr;

 std::memcpy(arr.data(), cstr, 12); // copy cstr to arr

 std::cout << arr.data(); // outputs: Test string

 return 0;
}

```

C std::arraysize()size()

```

int main() {
 std::array<int, 3> arr = { 1, 2, 3 };
 cout << arr.size() << endl;
}

```

std::arraySTLforvector

```

int main() {
 std::array<int, 3> arr = { 1, 2, 3 };
 for (auto i : arr)
 cout << i << '\n';
}

```

fill()std::array

```

int main() {

 std::array<int, 3> arr = { 1, 2, 3 };
 // change all elements of the array to 100
}

```

```
arr.fill(100);
}
```

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/2712/std--->

# 114:

## Examples

C++

T&& T T&

std::move(obj)

std::move std::move(obj) **obj** auto obj2 = std::move(obj)

```
class A {
public:
 int a;
 int b;

 A(const A &other) {
 this->a = other.a;
 this->b = other.b;
 }
};
```

AA

A(const A &) = default;

```
class Wallet {
public:
 int nrOfDollars;

 Wallet() = default; //default ctor

 Wallet(Wallet &&other) {
 this->nrOfDollars = other.nrOfDollars;
 other.nrOfDollars = 0;
 }
};
```

zero Wallet(Wallet&&) = default; nrOfDollars **POD**

“”

```
Wallet a;
```

```

a.nrofDollars = 1;
Wallet b (std::move(a)); //calling B(B&& other);
std::cout << a.nrofDollars << std::endl; //0
std::cout << b.nrofDollars << std::endl; //1

```

◦

◦ ◦

```

// Manages operations involving a specified type.
// Owns a helper on the heap, and one in its memory (presumably on the stack).
// Both helpers are DefaultConstructible, CopyConstructible, and MoveConstructible.
template<typename T,
 template<typename> typename HeapHelper,
 template<typename> typename StackHelper>
class OperationsManager {
 using MyType = OperationsManager<T, HeapHelper, StackHelper>;

 HeapHelper<T>* h_helper;
 StackHelper<T> s_helper;
 // ...

public:
 // Default constructor & Rule of Five.
 OperationsManager() : h_helper(new HeapHelper<T>) {}
 OperationsManager(const MyType& other)
 : h_helper(new HeapHelper<T>(*other.h_helper)), s_helper(other.s_helper) {}
 MyType& operator=(MyType copy) {
 swap(*this, copy);
 return *this;
 }
 ~OperationsManager() {
 if (h_helper) { delete h_helper; }
 }

 // Move constructor (without swap()).
 // Takes other's HeapHelper<T>*.
 // Takes other's StackHelper<T>, by forcing the use of StackHelper<T>'s move
constructor.
 // Replaces other's HeapHelper<T>* with nullptr, to keep other from deleting our shiny
 // new helper when it's destroyed.
 OperationsManager(MyType&& other) noexcept
 : h_helper(other.h_helper),
 s_helper(std::move(other.s_helper)) {
 other.h_helper = nullptr;
 }

 // Move constructor (with swap()).
 // Places our members in the condition we want other's to be in, then switches members
 // with other.
 // OperationsManager(MyType&& other) noexcept : h_helper(nullptr) {
 // swap(*this, other);
 // }

 // Copy/move helper.
 friend void swap(MyType& left, MyType& right) noexcept {
 std::swap(left.h_helper, right.h_helper);
 std::swap(left.s_helper, right.s_helper);
 }

```

```
 }
};
```

◦ ◦ ◦

operator = ◦

```
class A {
 int a;
 A& operator= (A&& other) {
 this->a = other.a;
 other.a = 0;
 return *this;
 }
};
```

◦ operator =◦

```
A a;
a.a = 1;
A b;
b = std::move(a); //calling A& operator= (A&& other)
std::cout << a.a << std::endl; //0
std::cout << b.a << std::endl; //1
```

◦

## std :: moveOn<sup>2</sup>On

C++ 11. ◦ ◦ ◦

- && std::string&&std::string
- T( T&& )
- auto operator=(T&&) -> T&◦

<utility>std::move◦ ◦

On nO1. nÓN<sup>2</sup>On ◦

[2013919Dobbs Journal](#) Andrew Koenig. ◦ Collatz

```
// Based on an example by Andrew Koenig in his Dr. Dobbs Journal article
// "Containers That Never Change" September 19, 2013, available at
// <url: http://www.drdobbs.com/cpp/containers-that-never-change/240161543>

// Includes here, e.g. <vector>

namespace my {
 template< class Item >
 using Vector_ = /* E.g. std::vector<Item> */;
```

```

auto concat(Vector_<int> const& v, int const x)
 -> Vector_<int>
{
 auto result{ v };
 result.push_back(x);
 return result;
}

auto collatz_aux(int const n, Vector_<int> const& result)
 -> Vector_<int>
{
 if(n == 1)
 {
 return result;
 }
 auto const new_result = concat(result, n);
 if(n % 2 == 0)
 {
 return collatz_aux(n/2, new_result);
 }
 else
 {
 return collatz_aux(3*n + 1, new_result);
 }
}

auto collatz(int const n)
 -> Vector_<int>
{
 assert(n != 0);
 return collatz_aux(n, Vector_<int>());
}
} // namespace my

#include <iostream>
using namespace std;
auto main() -> int
{
 for(int const x : my::collatz(42))
 {
 cout << x << ' ';
 }
 cout << '\n';
}

```

```
42 21 64 32 16 8 4 2
```

$O n 21 + 2 + 3 + \dots n$ .

**g ++Visual C ++**collatz(42)collatz(42) **8Collatz368 \* collatz(42) = 28**.

◦ const ◦ ◦ std::move

```

using std::move;

auto concat(Vector_<int> v, int const x)
 -> Vector_<int>

```

```

{
 v.push_back(x);
 // warning: moving a local object in a return statement prevents copy elision [-
Wpessimizing-move]
 // See https://stackoverflow.com/documentation/c%2b%2b/2489/copy-elision
 // return move(v);
 return v;
}

auto collatz_aux(int const n, Vector_<int> result)
 -> Vector_<int>
{
 if(n == 1)
 {
 return result;
 }
 auto new_result = concat(move(result), n);
 struct result; // Make absolutely sure no use of `result` after this.
 if(n % 2 == 0)
 {
 return collatz_aux(n/2, move(new_result));
 }
 else
 {
 return collatz_aux(3*n + 1, move(new_result));
 }
}

auto collatz(int const n)
 -> Vector_<int>
{
 assert(n != 0);
 return collatz_aux(n, Vector_<int>());
}

```

g ++Visual C ++0。

OnCollatz  $On^2 \rightarrow On$  。

---

◦ C ++ 14C ++ ◦ struct move struct result; ; ◦

C ++ const ◦

```

template< class Item >
class Copy_tracking_vector
{
private:
 static auto n_copy_ops()
 -> int&
 {
 static int value;
 return value;
 }

 vector<Item> items_;

```

```

public:
 static auto n() -> int { return n_copy_ops(); }

 void push_back(Item const& o) { items_.push_back(o); }
 auto begin() const { return items_.begin(); }
 auto end() const { return items_.end(); }

 Copy_tracking_vector(){}

 Copy_tracking_vector(Copy_tracking_vector const& other)
 : items_(other.items_)
 { n_copy_ops() += items_.size(); }

 Copy_tracking_vector(Copy_tracking_vector&& other)
 : items_(move(other.items_))
 {}
};

```

```

void print(const std::vector<int>& vec) {
 for (auto&& val : vec) {
 std::cout << val << ", ";
 }
 std::cout << std::endl;
}

int main() {
 // initialize vec1 with 1, 2, 3, 4 and vec2 as an empty vector
 std::vector<int> vec1{1, 2, 3, 4};
 std::vector<int> vec2;

 // The following line will print 1, 2, 3, 4
 print(vec1);

 // The following line will print a new line
 print(vec2);

 // The vector vec2 is assigned with move assignment.
 // This will "steal" the value of vec1 without copying it.
 vec2 = std::move(vec1);

 // Here the vec1 object is in an indeterminate state, but still valid.
 // The object vec1 is not destroyed,
 // but there's is no guarantees about what it contains.

 // The following line will print 1, 2, 3, 4
 print(vec2);
}

```

```

void consumingFunction(std::vector<int> vec) {
 // Some operations
}

int main() {
 // initialize vec with 1, 2, 3, 4
 std::vector<int> vec{1, 2, 3, 4};

 // Send the vector by move
 consumingFunction(std::move(vec));
}

```



```
// Here the vec object is in an indeterminate state.
// Since the object is not destroyed, we can assign it a new content.
// We will, in this case, assign an empty value to the vector,
// making it effectively empty
vec = {};

// Since the vector as gained a determinate value, we can use it normally.
vec.push_back(42);

// Send the vector by move again.
consumingFunction(std::move(vec));
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2129/>

# 115:

- 
- &&
- && funcArgs && ... args

|       |             |
|-------|-------------|
|       |             |
| other | other other |
| func  |             |
| args  | func        |

- `std::thread`
- `std::threadjoin`

## Examples

◦

- ◦

```
int n;
std::thread thread{ calculateSomething, std::ref(n) };

//Doing some other stuff

//We need 'n' now!
//Wait for the thread to finish - if it is not already done
thread.join();

//Now 'n' has the result of the calculation done in the separate thread
std::cout << n << '\n';
```

detach

```
std::thread thread{ doSomething };

//Detaching the thread, we don't need it anymore (for whatever reason)
thread.detach();

//The thread will terminate when it is done, or when the main thread returns
```

`const std::thread` ◦ `std::reference_wrapper`

```
void foo(int& b)
{
 b = 10;
}
```

```
int a = 1;
std::thread thread{ foo, std::ref(a) }; //'a' is now really passed as reference

thread.join();
std::cout << a << '\n'; //Outputs 10
```

```
void bar(const ComplexObject& co)
{
 co.doCalculations();
}

ComplexObject object;
std::thread thread{ bar, std::cref(object) }; //'object' is passed as const&

thread.join();
std::cout << object.getResult() << '\n'; //Outputs the result
```

## std :: thread

C++std :: thread ; . . .

- 
- 
- Functor
- Lambda

```
#include <iostream>
#include <thread>

void foo(int a)
{
 std::cout << a << '\n';
}

int main()
{
 // Create and execute the thread
 std::thread thread(foo, 10); // foo is the function to execute, 10 is the
 // argument to pass to it

 // Keep going; the thread is executed separately

 // Wait for the thread to finish; we stay here until it is done
 thread.join();

 return 0;
}
```

```
#include <iostream>
```

```

#include <thread>

class Bar
{
public:
 void foo(int a)
 {
 std::cout << a << '\n';
 }
};

int main()
{
 Bar bar;

 // Create and execute the thread
 std::thread thread(&Bar::foo, &bar, 10); // Pass 10 to member function

 // The member function will be executed in a separate thread

 // Wait for the thread to finish, this is a blocking operation
 thread.join();

 return 0;
}

```

---

## Functor

```

#include <iostream>
#include <thread>

class Bar
{
public:
 void operator()(int a)
 {
 std::cout << a << '\n';
 }
};

int main()
{
 Bar bar;

 // Create and execute the thread
 std::thread thread(bar, 10); // Pass 10 to functor object

 // The functor object will be executed in a separate thread

 // Wait for the thread to finish, this is a blocking operation
 thread.join();

 return 0;
}

```

---

## Lambda

```

#include <iostream>
#include <thread>

int main()
{
 auto lambda = [](int a) { std::cout << a << '\n'; };

 // Create and execute the thread
 std::thread thread(lambda, 10); // Pass 10 to the lambda expression

 // The lambda expression will be executed in a separate thread

 // Wait for the thread to finish, this is a blocking operation
 thread.join();

 return 0;
}

```

std::this\_threadnamespace °

|             |    |
|-------------|----|
| get_id      | id |
| sleep_for   |    |
| sleep_until |    |
| yield       |    |

std::this\_thread::get\_id**id**

```

void foo()
{
 //Print this threads id
 std::cout << std::this_thread::get_id() << '\n';
}

std::thread thread{ foo };
thread.join(); //'threads' id has now been printed, should be something like 12556

foo(); //The id of the main thread is printed, should be something like 2420

```

std::this\_thread::sleep\_for**3**

```

void foo()
{
 std::this_thread::sleep_for(std::chrono::seconds(3));
}

std::thread thread{ foo };
foo.join();

std::cout << "Waited for 3 seconds!\n";

```

std::this\_thread::sleep\_untilstd::this\_thread::sleep\_until 3

```
void foo()
{
 std::this_thread::sleep_until(std::chrono::system_clock::now() + std::chrono::hours(3));
}

std::thread thread{ foo };
thread.join();

std::cout << "We are now located 3 hours after the thread has been called\n";
```

std::this\_thread::yield

```
void foo(int a)
{
 for (int i = 0; i < a; ++i)
 std::this_thread::yield(); //Now other threads take priority, because this thread
 //isn't doing anything important

 std::cout << "Hello World!\n";
}

std::thread thread{ foo, 10 };
thread.join();
```

## std :: asyncstd :: thread

std::async° std::thread°

```
#include <future>
#include <iostream>

unsigned int square(unsigned int i){
 return i*i;
}

int main() {
 auto f = std::async(std::launch::async, square, 8);
 std::cout << "square currently running\n"; //do something while square is running
 std::cout << "result is " << f.get() << '\n'; //getting the result from square
}
```

- std::asyncstd::future ° future°

```
std::async(std::launch::async, square, 5);
//thread already completed at this point, because the returning future got destroyed
```

- std::asyncstd::async(square, 5);° ° ° std::launch::async°
- °

std::thread join()detach() ° std::terminate ° [RAII](#)

```

class thread_joiner
{
public:

 thread_joiner(std::thread t)
 : t_(std::move(t))
 { }

 ~thread_joiner()
 {
 if(t_.joinable()) {
 t_.join();
 }
 }

private:

 std::thread t_;
}

```

```

void perform_work()
{
 // Perform some work
}

void t()
{
 thread_joiner j{std::thread(perform_work)};
 // Do some other calculations while thread is running
} // Thread is automatically joined here

```

};t() join()。

。

joinablestd::terminate 。

```

#include <thread>

void foo()
{
 std::this_thread::sleep_for(std::chrono::seconds(3));
}

//create 100 thread objects that do nothing
std::thread executors[100];

// Some code

// I want to create some threads now

for (int i = 0;i < 100;i++)
{
 // If this object doesn't have a thread assigned
 if (!executors[i].joinable())
 executors[i] = std::thread(foo);
}

```

◦ `std::mutex` ◦ ◦ `std::lock_guard`

```
std::mutex m;
void worker() {
 std::lock_guard<std::mutex> guard(m); // Acquires a lock on the mutex
 // Synchronized code here
} // the mutex is automatically released when guard goes out of scope
```

`std::lock_guard` ◦ `std::unique_lock`

```
std::mutex m;
void worker() {
 // by default, constructing a unique_lock from a mutex will lock the mutex
 // by passing the std::defer_lock as a second argument, we
 // can construct the guard in an unlocked state instead and
 // manually lock later.
 std::unique_lock<std::mutex> guard(m, std::defer_lock);
 // the mutex is not locked yet!
 guard.lock();
 // critical section
 guard.unlock();
 // mutex is again released
}
```

◦ ◦

`std::unique_lock<std::mutex>std::condition_variable` ◦ ◦

- `std::thread std::condition_variable std::mutex`◦

```
#include <condition_variable>
#include <cstdint>
#include <iostream>
#include <mutex>
#include <queue>
#include <random>
#include <thread>

int main()
{
 std::condition_variable cond;
 std::mutex mtx;
 std::queue<int> intq;
 bool stopped = false;

 std::thread producer{[&]()
 {
 // Prepare a random number generator.
 // Our producer will simply push random numbers to intq.
 //
 std::default_random_engine gen{};
 std::uniform_int_distribution<int> dist{};

 std::size_t count = 4006;
 while(count--)
 {
```



```

 // Always lock before changing
 // state guarded by a mutex and
 // condition_variable (a.k.a. "condvar").
 std::lock_guard<std::mutex> L{mtx};

 // Push a random int into the queue
 intq.push(dist(gen));

 // Tell the consumer it has an int
 cond.notify_one();
}

// All done.
// Acquire the lock, set the stopped flag,
// then inform the consumer.
std::lock_guard<std::mutex> L{mtx};

std::cout << "Producer is done!" << std::endl;

stopped = true;
cond.notify_one();
});

std::thread consumer{[&]()
{
 do{
 std::unique_lock<std::mutex> L{mtx};
 cond.wait(L, [&]()
 {
 // Acquire the lock only if
 // we've stopped or the queue
 // isn't empty
 return stopped || !intq.empty();
 });

 // We own the mutex here; pop the queue
 // until it empties out.

 while(! intq.empty())
 {
 const auto val = intq.front();
 intq.pop();

 std::cout << "Consumer popped: " << val << std::endl;
 }

 if(stopped){
 // producer has signaled a stop
 std::cout << "Consumer is done!" << std::endl;
 break;
 }

 }while(true);
});

consumer.join();
producer.join();

std::cout << "Example Completed!" << std::endl;

return 0;

```

```
}
```

## C++ 11。

## C++ 14

```
struct tasks {
 // the mutex, condition variable and deque form a single
 // thread-safe triggered queue of tasks:
 std::mutex m;
 std::condition_variable v;
 // note that a packaged_task<void> can store a packaged_task<R>:
 std::deque<std::packaged_task<void()>> work;

 // this holds futures representing the worker threads being done:
 std::vector<std::future<void>> finished;

 // queue(lambda) will enqueue the lambda into the tasks for the threads
 // to use. A future of the type the lambda returns is given to let you get
 // the result out.
 template<class F, class R=std::result_of_t<F&()>>
 std::future<R> queue(F&& f) {
 // wrap the function object into a packaged task, splitting
 // execution from the return value:
 std::packaged_task<R()> p(std::forward<F>(f));

 auto r=p.get_future(); // get the return value before we hand off the task
 {
 std::unique_lock<std::mutex> l(m);
 work.emplace_back(std::move(p)); // store the task<R()> as a task<void()>
 }
 v.notify_one(); // wake a thread to work on the task

 return r; // return the future result of the task
 }

 // start N threads in the thread pool.
 void start(std::size_t N=1){
 for (std::size_t i = 0; i < N; ++i)
 {
 // each thread is a std::async running this->thread_task():
 finished.push_back(
 std::async(
 std::launch::async,
 [this]{ thread_task(); }
)
);
 }
 }

 // abort() cancels all non-started tasks, and tells every working thread
 // stop running, and waits for them to finish up.
 void abort() {
 cancel_pending();
 finish();
 }

 // cancel_pending() merely cancels all non-started tasks:
 void cancel_pending() {
 std::unique_lock<std::mutex> l(m);
 work.clear();
 }
}
```

```

// finish enques a "stop the thread" message for every thread, then waits for them:
void finish() {
 {
 std::unique_lock<std::mutex> l(m);
 for(auto&&unused:finished){
 work.push_back({});
 }
 }
 v.notify_all();
 finished.clear();
}
~tasks() {
 finish();
}
private:
// the work that a worker thread does:
void thread_task() {
 while(true){
 // pop a task off the queue:
 std::packaged_task<void()> f;
 {
 // usual thread-safe queue code:
 std::unique_lock<std::mutex> l(m);
 if (work.empty()){
 v.wait(l, [&]{return !work.empty();});
 }
 f = std::move(work.front());
 work.pop_front();
 }
 // if the task is invalid, it means we are asked to abort:
 if (!f.valid()) return;
 // otherwise, run the task:
 f();
 }
}
};

```

tasks.queue( []{ return "hello world"s; } )std::future<std::string> **task**hello world ◦

tasks.start(10) **10**◦

packaged\_task<void()>std::function◦ packaged\_task<void()>◦

◦

## C++ 11

C++ 11 result\_of\_t<blah>typename result\_of<blah>::type ◦

◦

thread\_local ◦ thread\_local◦

- ◦
- ◦ extern ◦
- ◦
-

- 
- ◦ ◦

```
void debug_counter() {
 thread_local int count = 0;
 Logger::log("This function has been called %d times by this thread", ++count);
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/699/>

# 116:

◦ ◦

## Examples

### std :: shared\_lock

shared\_lock◦

```
#include <unordered_map>
#include <mutex>
#include <shared_mutex>
#include <thread>
#include <string>
#include <iostream>

class PhoneBook {
public:
 string getPhoneNo(const std::string & name)
 {
 shared_lock<shared_timed_mutex> r(_protect);
 auto it = _phonebook.find(name);
 if (it == _phonebook.end())
 return (*it).second;
 return "";
 }
 void addPhoneNo (const std::string & name, const std::string & phone)
 {
 unique_lock<shared_timed_mutex> w(_protect);
 _phonebook[name] = phone;
 }

 shared_timed_mutex _protect;
 unordered_map<string, string> _phonebook;
};
```

### std :: call\_oncstd :: once\_flag

std::call\_once◦ std::system\_error◦

Std::once\_flag◦

```
#include <mutex>
#include <iostream>

std::once_flag flag;
void do_something() {
 std::call_once(flag, []() {std::cout << "Happens once" << std::endl;});

 std::cout << "Happens every time" << std::endl;
}
```

◦  
◦ ◦ ◦

```
class text_buffer
{
 // for readability/maintainability
 using mutex_type = std::shared_timed_mutex;
 using reading_lock = std::shared_lock<mutex_type>;
 using updates_lock = std::unique_lock<mutex_type>;

public:
 // This returns a scoped lock that can be shared by multiple
 // readers at the same time while excluding any writers
 [[nodiscard]]
 reading_lock lock_for_reading() const { return reading_lock(mtx); }

 // This returns a scoped lock that is excluding to one
 // writer preventing any readers
 [[nodiscard]]
 updates_lock lock_for_updates() { return updates_lock(mtx); }

 char* data() { return buf; }
 char const* data() const { return buf; }

 char* begin() { return buf; }
 char const* begin() const { return buf; }

 char* end() { return buf + sizeof(buf); }
 char const* end() const { return buf + sizeof(buf); }

 std::size_t size() const { return sizeof(buf); }

private:
 char buf[1024];
 mutable mutex_type mtx; // mutable allows const objects to be locked
};
```

◦

```
std::size_t checksum(text_buffer const& buf)
{
 std::size_t sum = 0xA44944A4;

 // lock the object for reading
 auto lock = buf.lock_for_reading();

 for(auto c: buf)
 sum = (sum << 8) | (((unsigned char) ((sum & 0xFF000000) >> 24)) ^ c);

 return sum;
}
```

◦

```
void clear(text_buffer& buf)
{
```

```
auto lock = buf.lock_for_updates(); // exclusive lock
std::fill(std::begin(buf), std::end(buf), '\0');
}
```

◦

```
void transfer(text_buffer const& input, text_buffer& output)
{
 auto lock1 = input.lock_for_reading();
 auto lock2 = output.lock_for_updates();

 std::copy(std::begin(input), std::end(input), std::begin(output));
}
```

[std :: deferred :: lockstd :: lock](#)

## std :: condition\_variable\_anystd :: cv\_status

std::condition\_variable std::condition\_variable\_any **BasicLockable**◦

std::cv\_status

- std :: cv\_status :: no\_timeout
- std :: cv\_status :: no\_timeout

<https://riptutorial.com/zh-TW/cplusplus/topic/9794/>

# 117:

C++。

- GCCGNU g++
- clangLLVM clang ++ C
- MSVCMicrosoft Visual C ++Visual Studio visual-c ++
- C ++ BuilderEmbarcadero C ++ Builder RAD Studio c ++ builder

C++。

。

## Examples

### GCC

main.cppGCC。

```
g++ -o app -Wall main.cpp -O0
```

```
-O -O1 -O2 -O3 -Os -Ofast
```

```
g++ -o app -Wall -O2 main.cpp
```

-O-O0-O01。

O -O2-O3

```
g++ -o app -Wall -O2 -ftree-partial-pre main.cpp
```

```
g++ -o app -Wall -O2 -march=native main.cpp
```

.\app.exeWindows./appLinuxMac OS。

-o GCCWindowsa.exe Unixa.out -c

```
g++ -o file.o -Wall -c file.cpp
```

file.o

```
g++ -o app file.o otherfile.o
```



[gcc.gnu.org](http://gcc.gnu.org) ◦ -Og - - -Ofast ◦

-Wall ◦ -Wextra -Wall -Wextra ◦

C ++-std= ◦ ISO C ++ ◦ GCC 6.1.0 std= flag c++98 / c++03 c++11 c++14c++17 / c++1z ◦ ◦

```
g++ -std=c++11 <file>
```

**GCC**-std= flag ◦ gnu++XX XXc++ ◦

◦ 6.1.0GCC -std=gnu++03 ;GCC 6.1.0 -std=gnu++14 ◦

**GCC** -pthreadGCC ++ 11C ++std::threadstd::wait\_for ◦ ◦

-l

```
g++ main.cpp -lpcr2-8
#pcr2-8 is the PCRE2 library for 8bit code units (UTF-8)
```

-L

```
g++ main.cpp -L/my/custom/path/ -lmylib
```

```
g++ main.cpp -lmylib1 -lmylib2 -lmylib3
```

```
g++ main.cpp -lchild-lib -lbase-lib
```

--start-group--end-group

```
g++ main.cpp -Wl,--start-group -lbase-lib -lchild-lib -Wl,--end-group
```

## Visual C ++

**GCC**ClangVisual StudioIDEVisual C ++ ◦

Visual Studio ◦ [Visual Studio Command Prompt / Developer Command Prompt / x86 Native Tools Command Prompt / x64 Native Tools Command Prompt](#) Visual StudioVC\Program Files (x86)\Microsoft Visual Studio x\VC x201010.0201514.0 VCVARSALL ◦

**GCC**Visual Studio cl.exe link.exe ◦ cl.exelink.execllink ◦ cllinkcllink ◦ Visual Studio link ◦

cllink ◦

[Windows shell"" %cd% ◦ ◦ clC:\src> %cd%C:\src\ ◦ ]

main.cpp

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".

cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

## “niam.cpp”

```
cl main.cpp niam.cpp
// Generates object files "main.obj" and "niam.obj".
// Performs linking with "main.obj" and "niam.obj".
// Generates executable "main.exe".
```

```
cl main.cpp src*.cpp
// Generates object file "main.obj", plus one object file for each ".cpp" file in folder
// "%cd%\src".
// Performs linking with "main.obj", and every additional object file generated.
// All object files will be in the current folder.
// Generates executable "main.exe".
```

```
cl /o name main.cpp
// Generates executable named "name.exe".

cl /o folder\ main.cpp
// Generates executable named "main.exe", in folder "%cd%\folder".

cl /o folder\name main.cpp
// Generates executable named "name.exe", in folder "%cd%\folder".

cl /Fename main.cpp
// Same as "/o name".

cl /Fefolder\ main.cpp
// Same as "/o folder\".

cl /Fefolder\name main.cpp
// Same as "/o folder\name".
```

/o/Feo-param link/OUT:o-param .exe.dll "" o-param /o/FeVisual Studio /oGCCClang

/o//Fe/

```
cl /O1 main.cpp
// Optimise for executable size. Produces small programs, at the possible expense of slower
// execution.

cl /O2 main.cpp
// Optimise for execution speed. Produces fast programs, at the possible expense of larger
// file size.
```

```
cl /GL main.cpp other.cpp
// Generates special object files used for whole-program optimisation, which allows CL to
// take every module (translation unit) into consideration during optimisation.
// Passes the option "/LTCG" (Link-Time Code Generation) to LINK, telling it to call CL during
// the linking phase to perform additional optimisations. If linking is not performed at
// this
// time, the generated object files should be linked with "/LTCG".
// Can be used with other CL optimisation options.
```

---

`VCVARSALL` ◦ `link;/MACHINE` ◦

```
// If compiling for x64, and LINK doesn't automatically detect target platform:
cl main.cpp /link /machine:X64
```

`/o/Fe` ◦

```
cl a.cpp b.cpp c.cpp
// Generates "a.exe".

cl d.obj a.cpp q.cpp
// Generates "d.exe".

cl y.lib n.cpp o.obj
// Generates "n.exe".

cl /o yo zp.obj pz.cpp
// Generates "yo.exe".
```

---

```
cl /c main.cpp
// Generates object file "main.obj".
```

`cllink` ◦

```
cl main.obj niam.cpp
// Generates object file "niam.obj".
// Performs linking with "main.obj" and "niam.obj".
// Generates executable "main.exe".

link main.obj niam.obj
// Performs linking with "main.obj" and "niam.obj".
// Generates executable "main.exe".
```

---

```
cl /EHsc main.cpp
// "/EHsc" specifies that only standard C++ ("synchronous") exceptions will be caught,
// and `extern "C"` functions will not throw exceptions.
// This is recommended when writing portable, platform-independent code.

cl /clr main.cpp
// "/clr" specifies that the code should be compiled to use the common language runtime,
// the .NET Framework's virtual machine.
// Enables the use of Microsoft's C++/CLI language in addition to standard ("native") C++,
// and creates an executable that requires .NET to run.
```

```

cl /Za main.cpp
// "/Za" specifies that Microsoft extensions should be disabled, and code should be
// compiled strictly according to ISO C++ specifications.
// This is recommended for guaranteeing portability.

cl /Zi main.cpp
// "/Zi" generates a program database (PDB) file for use when debugging a program, without
// affecting optimisation specifications, and passes the option "/DEBUG" to LINK.

cl /LD dll.cpp
// "/LD" tells CL to configure LINK to generate a DLL instead of an executable.
// LINK will output a DLL, in addition to an LIB and EXP file for use when linking.
// To use the DLL in other programs, pass its associated LIB to CL or LINK when compiling
// those
// programs.

cl main.cpp /link /LINKER_OPTION
// "/link" passes everything following it directly to LINK, without parsing it in any way.
// Replace "/LINKER_OPTION" with any desired LINK option(s).

```

\* nix/GCC / Clang cl link Visual Studio -c /c Windows\* nix g++clang++cl

```

g++ -o app src/main.cpp
cl -o app src/main.cpp

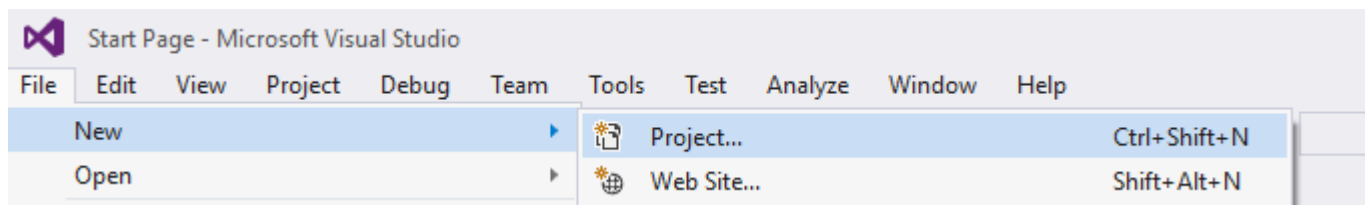
```

g++clang++/

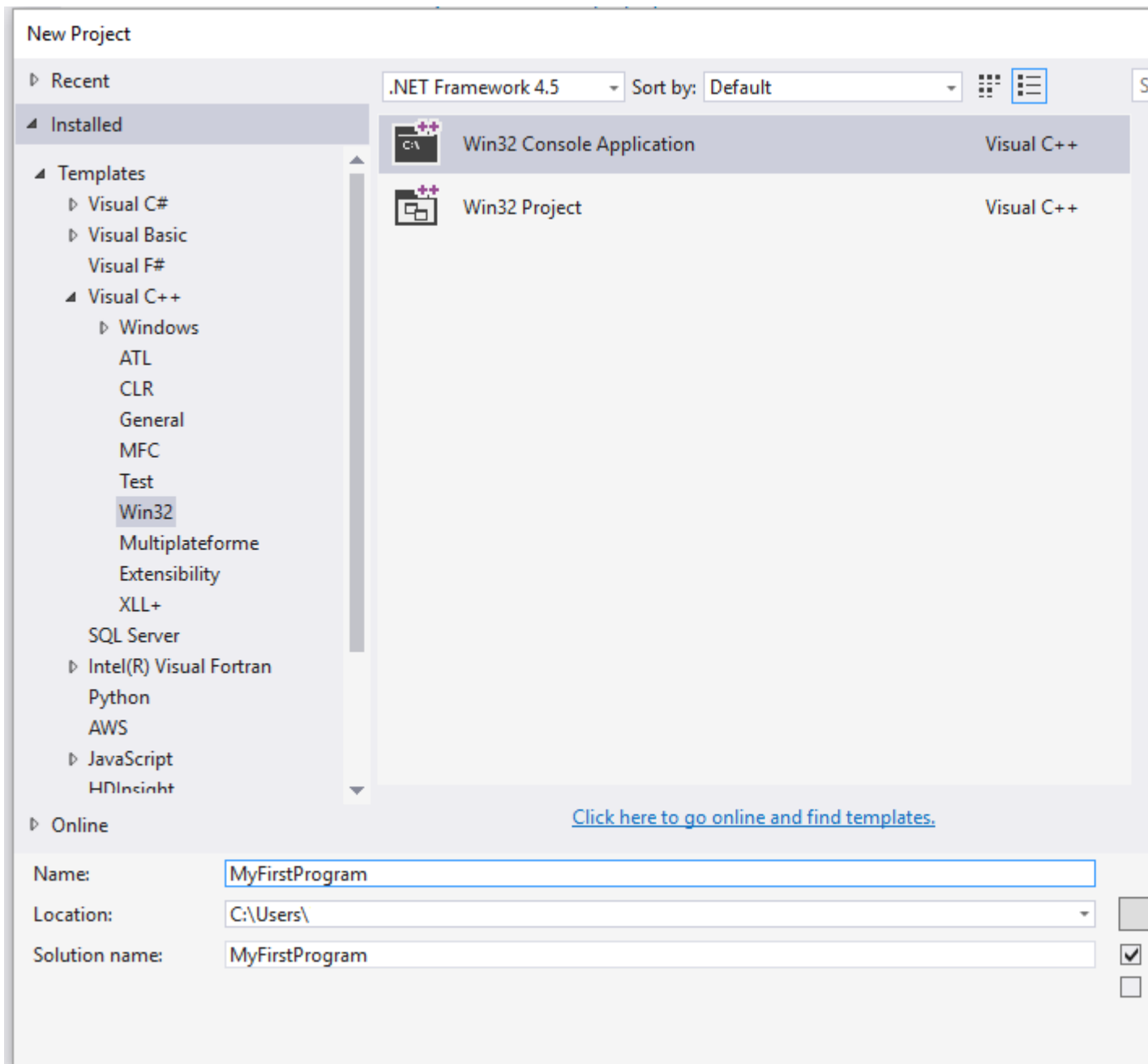
MSVC Visual C ++ 2015 Update 3/std /std:c++14/std:c++latest /std:c++17

## Visual Studio - Hello World

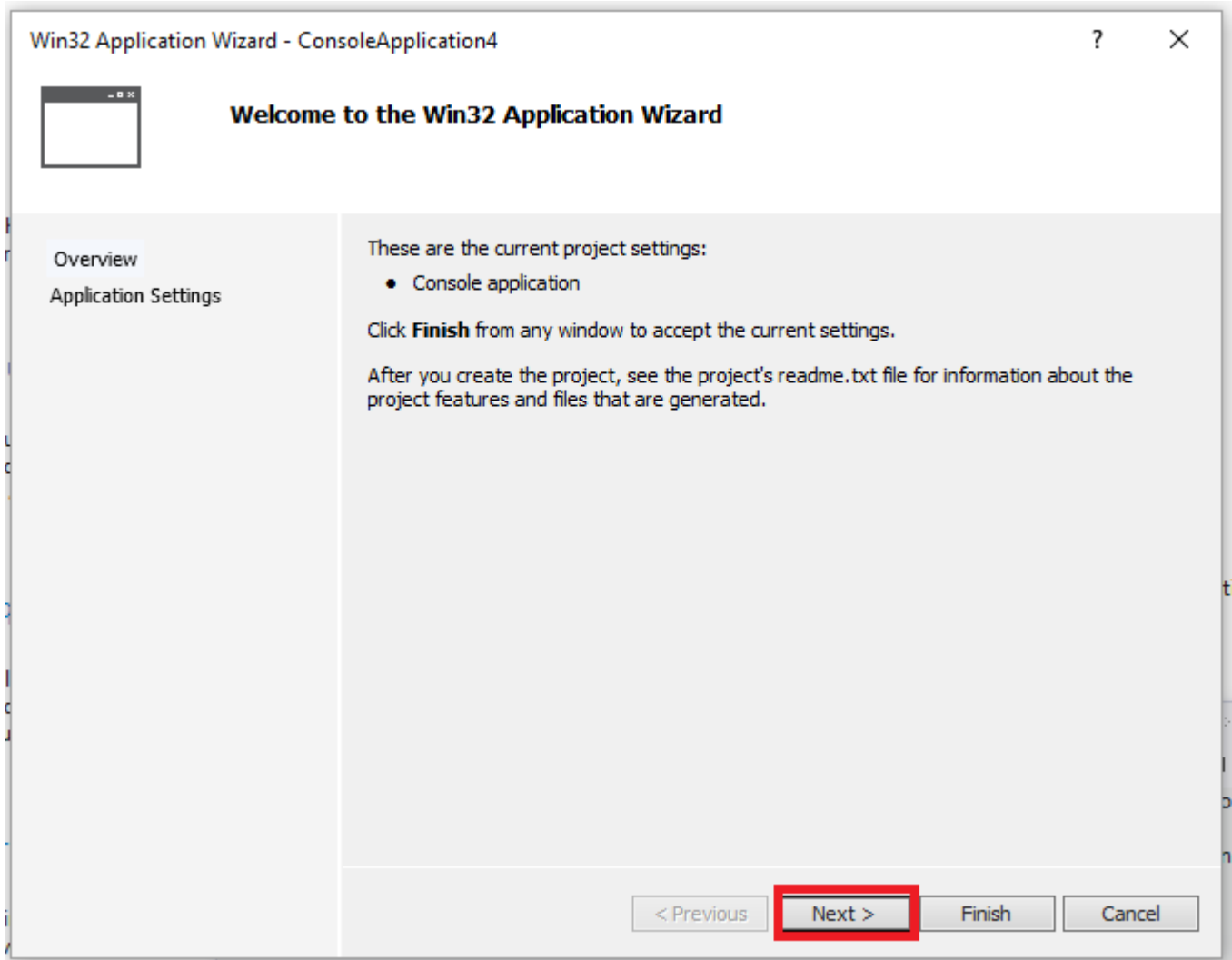
1. [Visual Studio Community 2015](#)
2. Visual Studio
3. -> ->



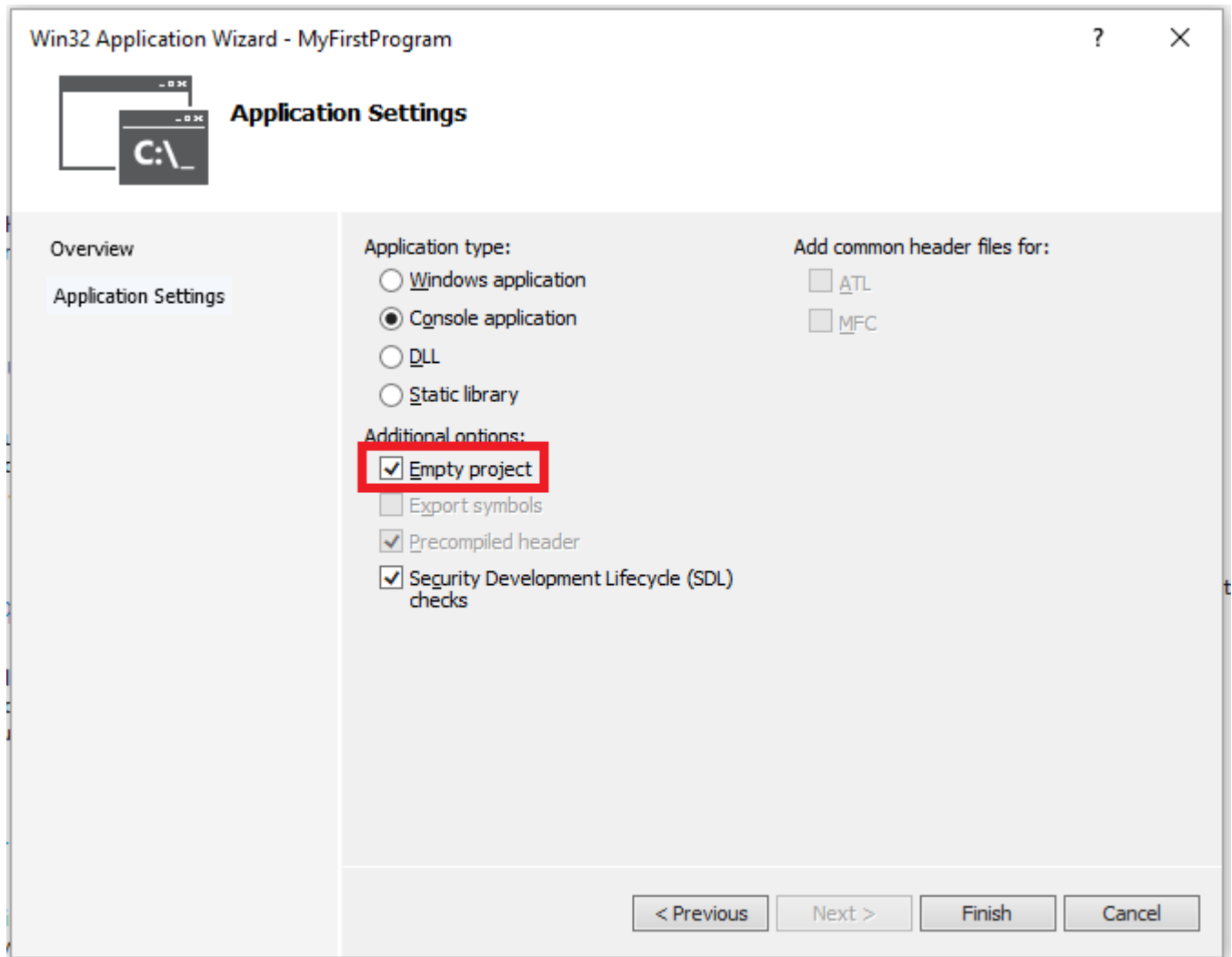
4. -> Visual C ++ -> Win32MyFirstProgram



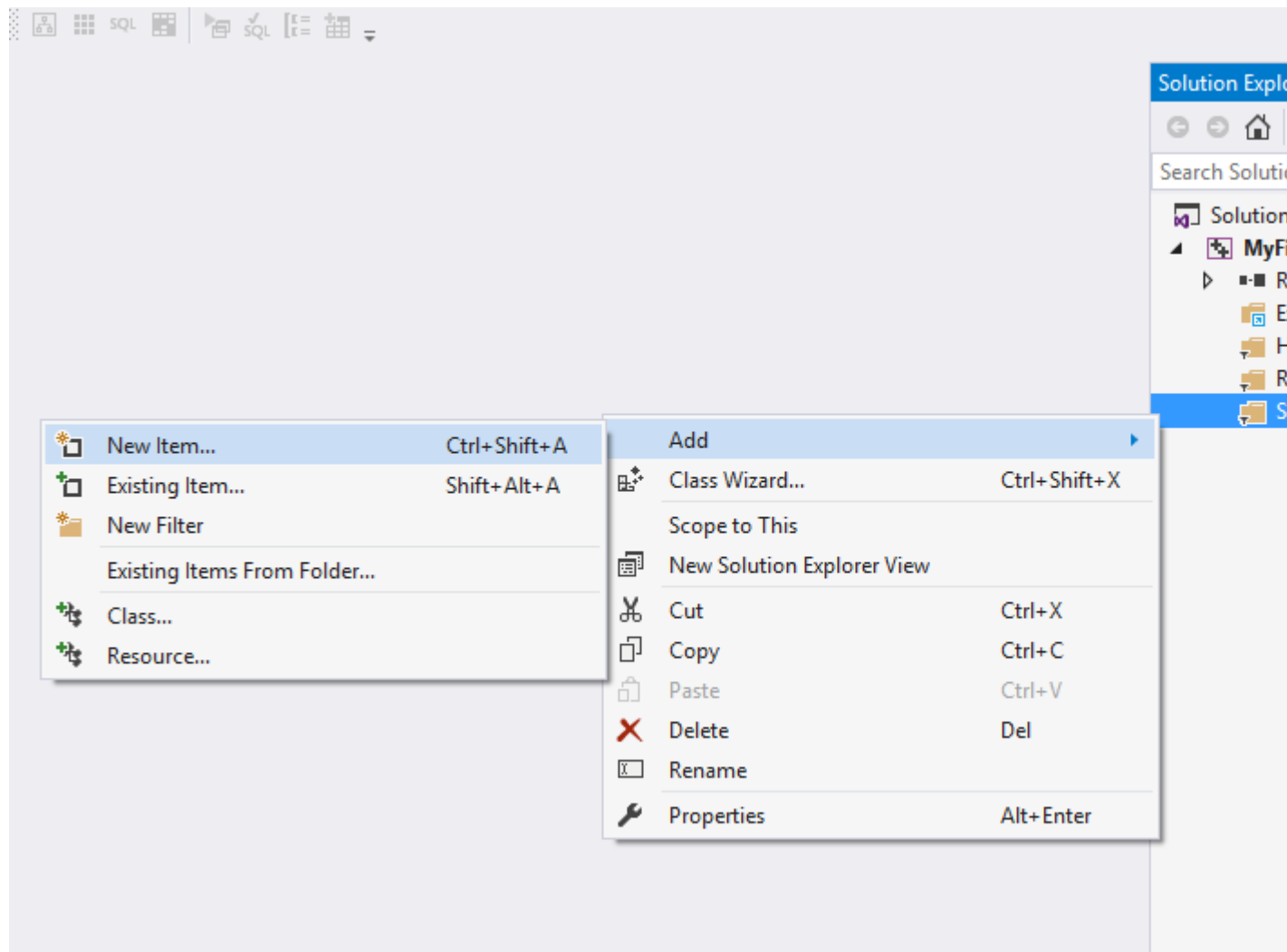
- 5.
6. ""。



7. Empty project **Finish**

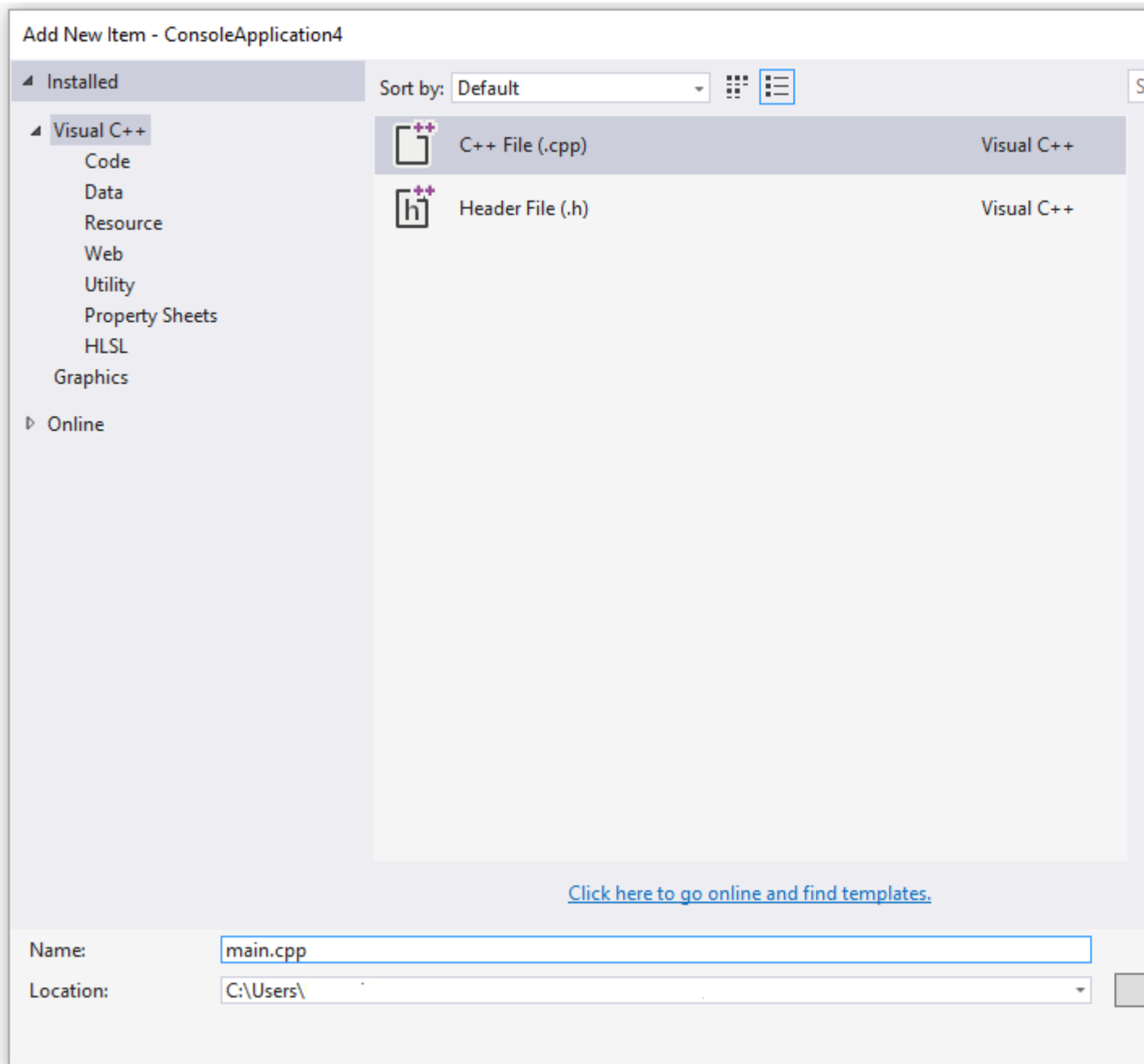


8. -> ->



## 9. C ++ Filemain.cppAdd

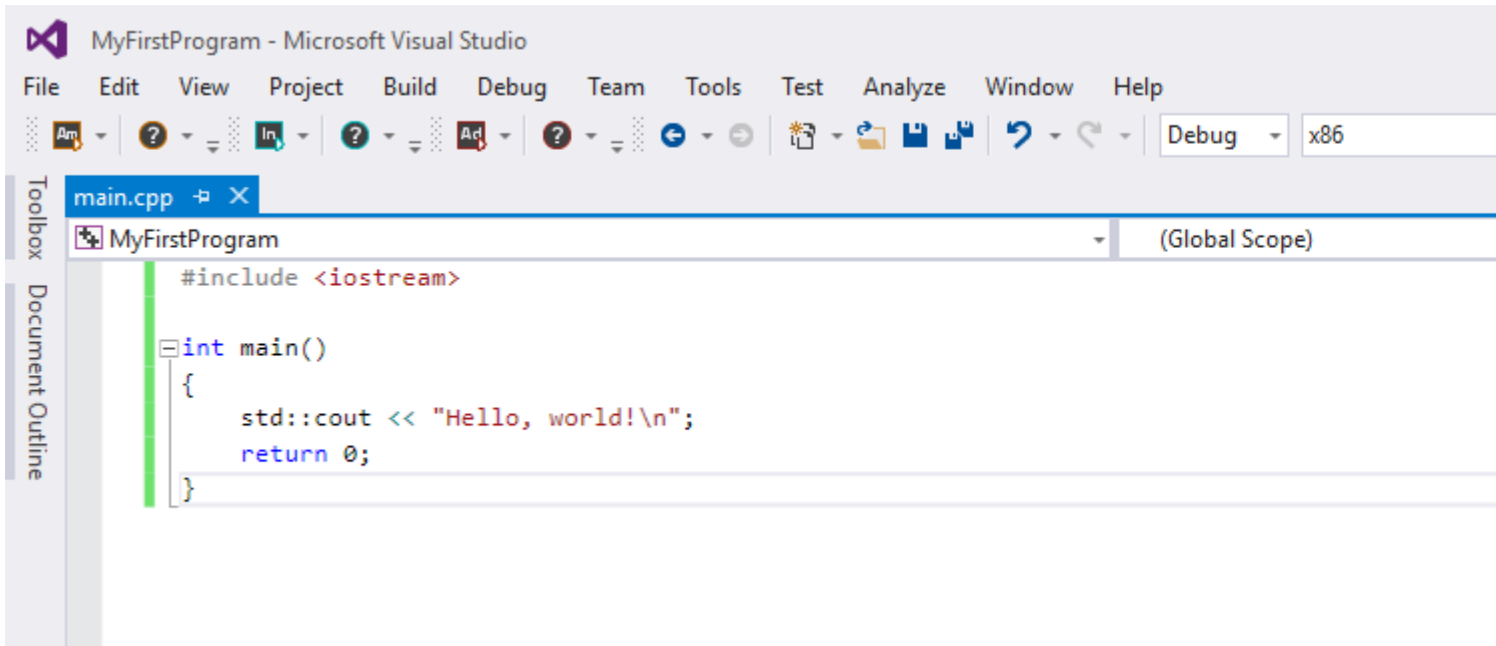




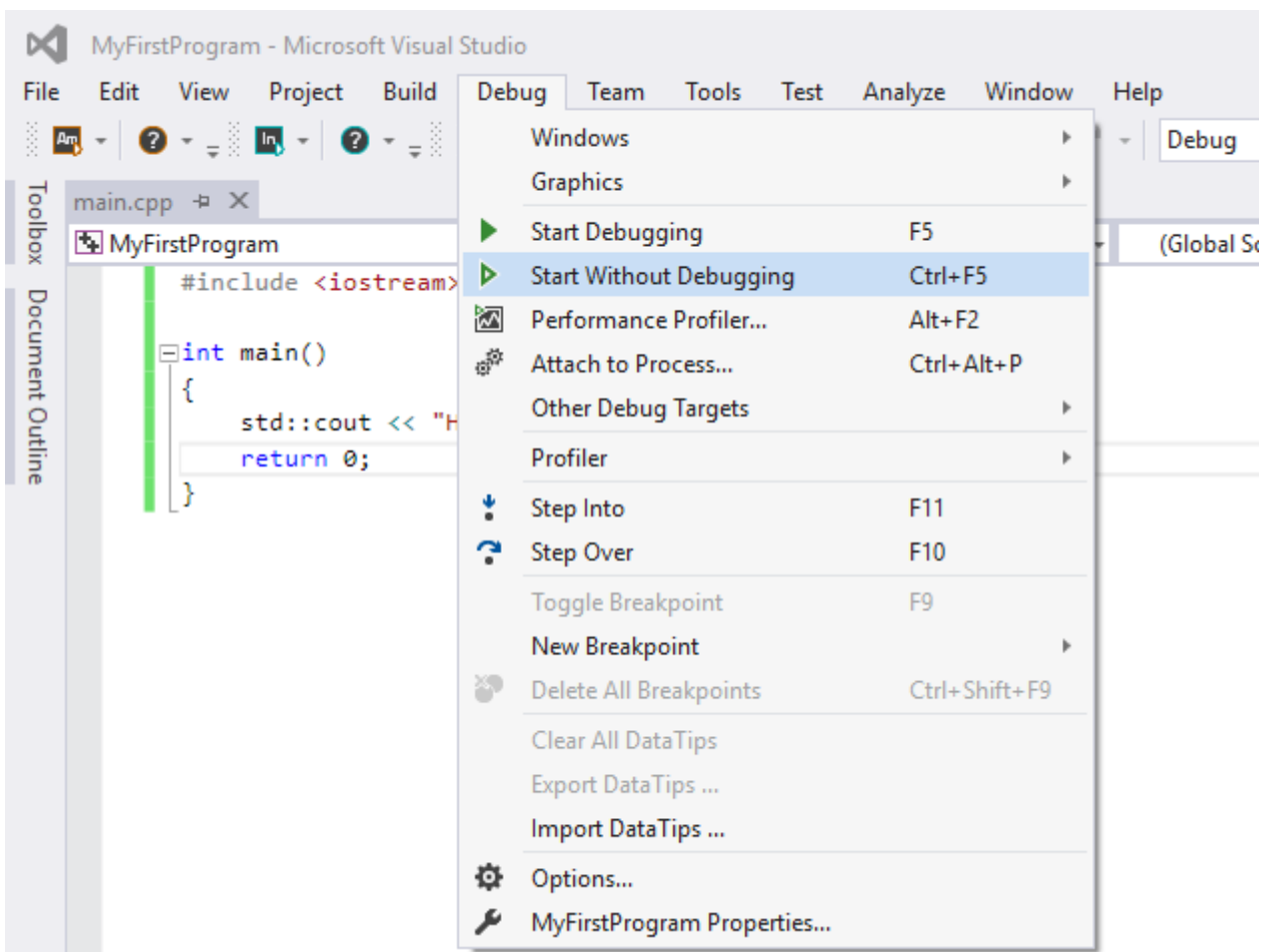
10main.cpp

```
#include <iostream>

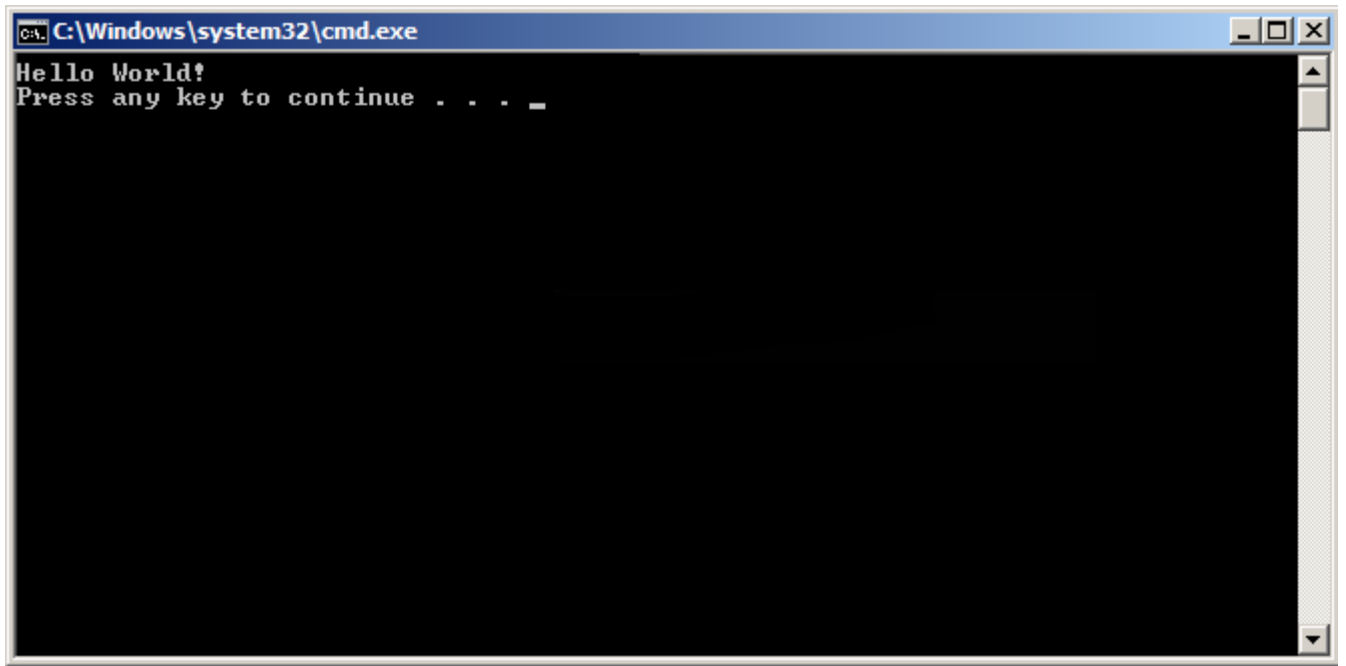
int main()
{
 std::cout << "Hello World!\n";
 return 0;
}
```



11. Debug -> Start **Without Debugging** ctrl + F5



12. ◦



## Clang

Clang GCC clang++ gcc -std=version gnu11

Windows MSVC cl.exe clang-cl.exe clang MSVC

visual studio Platform toolset clang-cl

clang clang-cl MSVC

clang clang-cl Linux IDE Mac XCode

CMake CXX

```
mkdir build
cd build
CC=clang CXX=clang++ cmake ..
cmake --build .
```

Cmake

C++

- Web
- 
- /

◦

- C++
-

- C ++。
- C ++C ++C ++ 14C ++ 17。
- 。
- 。
- C ++。
- 。
- 。
- 。
- <http://codepad.org/>。
- <http://coliru.stacked-crooked.com/>。GCCClang。
- <http://cpp.sh/> - C ++ 14。GUI。
- <https://gcc.godbolt.org/> - 。
- <https://ideone.com/> - 。
- <http://melpon.org/wandbox> - ClangGNU / GCC。
- <http://onlinegdb.com/> - IDEgccgdb。
- <http://rextester.com/> - CC ++ClangGCCVisual StudioBoost。
- [http://tutorialspoint.com/compile\\_cpp11\\_online.php](http://tutorialspoint.com/compile_cpp11_online.php) - GCCUNIX shell。
- <http://webcompiler.cloudapp.net/> - Visual Studio 2015MicrosoftRiSE4fun。

## C ++

C ++。CC ++。prog.cppC ++compile

```
g++ -Wall -ansi -o prog prog.cpp
```

4。

1. C ++C ++#include#define。

2. C ++C ++。

3. 。

4.

。

#include#define。C ++。

C ++#include#define#if #ifdef#ifndef。

。##。

。

#if#error

o

```
g++ -E prog.cpp
```

o C ++. ELFCOFFa.out... o o

o o o

o o o

o

“”

-S

```
g++ -Wall -ansi -S prog.cpp
```

o UNIX.oMSDOS.OBJ o o

-C

```
g++ -Wall -ansi -c prog.cpp
```

o o

o o o

o o o

## Code :: Blocks

1. Code :: Blocks Windowsmingw

2. Code :: Blocks“”

Start here - Code::Blocks 16.01

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plug



Management

Projects Symbols

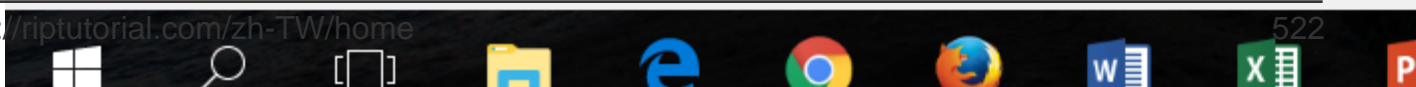
Workspace

Start here

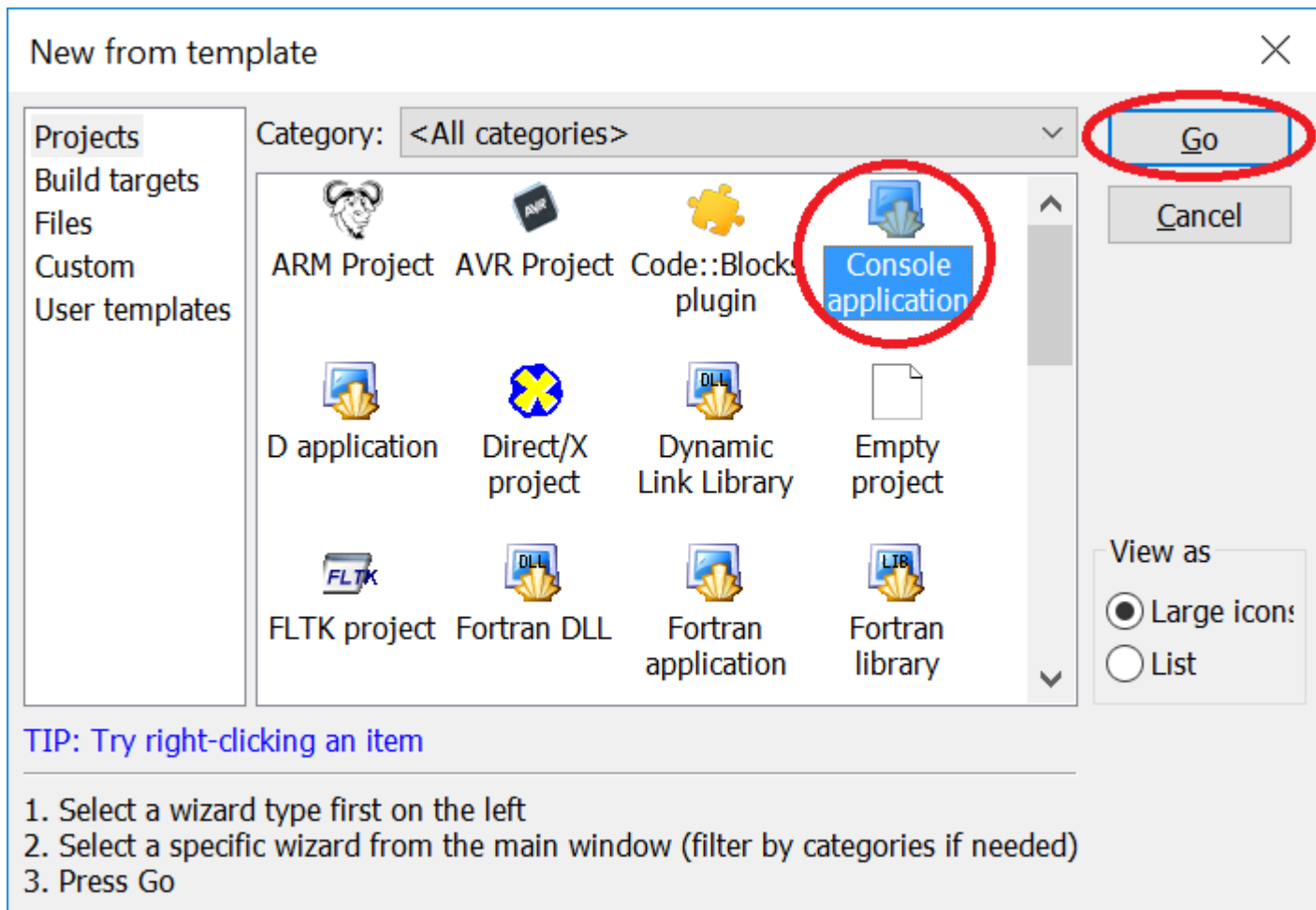
Logs & others

Code::Blocks Search results Cccc Build

Start here



3. ""



**New from template**

Category: <All categories>

Projects  
Build targets  
Files  
Custom  
User templates

ARM Project AVR Project Code::Blocks plugin **Console application**

D application Direct/X project Dynamic Link Library Empty project

FLTK project Fortran DLL Fortran application Fortran library

View as  
 Large icons  
 List

**TIP: Try right-clicking an item**

1. Select a wizard type first on the left
2. Select a specific wizard from the main window (filter by categories if needed)
3. Press Go

4. ""C ++"".

5. "Hello world". //



Management

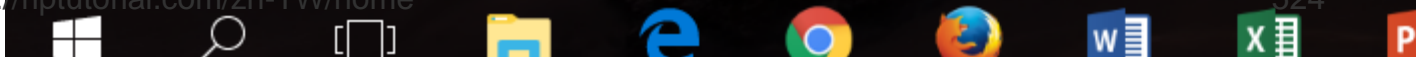
Projects Symbols

- Workspace
  - df
    - Sources
      - main.cpp

```
main.cpp x
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7 cout << "Hello world!" << endl;
8 return 0;
9 }
10
```

Logs & others

- Code::Blocks
- Search results
- Cccc
- Build







“Hello world”

```
Hello world!
Process returned 0 (0x0) execution time : 0.047 s
Press any key to continue.
_
```

<https://riptutorial.com/zh-TW/cplusplus/topic/4708/>

# 118:

C.

- *extern string-literal { declaration-seq opt }*
- *extern string-literal*

```
extern "C" C ++ C extern "C++" . .
```

## Examples

### Unix

CC.

```
volatile sig_atomic_t death_signal = 0;
extern "C" void cleanup(int signum) {
 death_signal = signum;
}
int main() {
 bind(...);
 listen(...);
 signal(SIGTERM, cleanup);
 while (int fd = accept(...)) {
 if (fd == -1 && errno == EINTR && death_signal) {
 printf("Caught signal %d; shutting down\n", death_signal);
 break;
 }
 // ...
 }
}
```

### CC ++

ACC ++ CC ++. foo.h

```
typedef struct Foo {
 int bar;
} Foo;
Foo make_foo(int);
```

make\_foo.

```
C ++ #include <foo.h> make_foo C. make_foo CC ++ C ++ make_foo C ++ make_foo C.
```

extern "C".

```
#ifdef __cplusplus
extern "C" {
```

```
#endif

typedef struct Foo {
 int bar;
} Foo;
Foo make_foo(int);

#ifdef __cplusplus
} /* end of "extern C" block */
#endif
```

foo.h **CC ++**foo.h make\_fooextern "C"unmangledC。

<https://riptutorial.com/zh-TW/cplusplus/topic/9268/>

# 119:

- virtual void f;
- virtual void g= 0;
- // C ++ 11
  - virtual void h;
  - void i;
  - virtual void jfinal;
  - void kfinal;
- virtual ◦
- C ++ 11override ◦
- ◦ [expr.delete]§5.3.5/ 3 ◦

## Examples

### C ++ 11override with virtual

overrideC ++ 11◦

- 
- Basevirtual

run time

◦

override

```
#include <iostream>

struct X {
 virtual void f() { std::cout << "X::f()\n"; }
};

struct Y : X {
 // Y::f() will not override X::f() because it has a different signature,
 // but the compiler will accept the code (and silently ignore Y::f()).
 virtual void f(int a) { std::cout << a << "\n"; }
};
```

override

```
#include <iostream>

struct X {
 virtual void f() { std::cout << "X::f()\n"; }
```

```

};

struct Y : X {
 // The compiler will alert you to the fact that Y::f() does not
 // actually override anything.
 virtual void f(int a) override { std::cout << a << "\n"; }
};

```

override◦ override

```

void foo() {
 int override = 1; // OK.
 int virtual = 2; // Compilation error: keywords can't be used as identifiers.
}

```

```

#include <iostream>

struct X {
 virtual void f() { std::cout << "X::f()\n"; }
};

struct Y : X {
 // Specifying virtual again here is optional
 // because it can be inferred from X::f().
 virtual void f() { std::cout << "Y::f()\n"; }
};

void call(X& a) {
 a.f();
}

int main() {
 X x;
 Y y;
 call(x); // outputs "X::f()"
 call(y); // outputs "Y::f()"
}

```

```

#include <iostream>

struct X {
 void f() { std::cout << "X::f()\n"; }
};

struct Y : X {
 void f() { std::cout << "Y::f()\n"; }
};

void call(X& a) {
 a.f();
}

int main() {
 X x;
 Y y;
 call(x); // outputs "X::f()"
 call(y); // outputs "X::f()"
}

```

```
}
```

## C++ 11 final

```
class Base {
public:
 virtual void foo() {
 std::cout << "Base::Foo\n";
 }
};

class Derived1 : public Base {
public:
 // Overriding Base::foo
 void foo() final {
 std::cout << "Derived1::Foo\n";
 }
};

class Derived2 : public Derived1 {
public:
 // Compilation error: cannot override final method
 virtual void foo() {
 std::cout << "Derived2::Foo\n";
 }
};
```

final`virtual`

final"virtual"

overridefinal

```
class Derived1 : public Base {
public:
 void foo() final override {
 std::cout << "Derived1::Foo\n";
 }
};
```

o

```
#include <iostream>
using namespace std;

class base {
public:
 base() { f("base constructor"); }
 ~base() { f("base destructor"); }

 virtual const char* v() { return "base::v()"; }

 void f(const char* caller) {
 cout << "When called from " << caller << ", " << v() << " gets called.\n";
 }
}
```

```

};

class derived : public base {
public:
 derived() { f("derived constructor"); }
 ~derived() { f("derived destructor"); }

 const char* v() override { return "derived::v()"; }
};

int main() {
 derived d;
}

```

**base :: v.**  
**derived :: v.**  
**derived :: v.**  
**basebase :: v.**

◦ **C++ \*this**◦

```

#include <iostream>
#include <memory>

using namespace std;
class base {
public:
 base()
 {
 std::cout << "foo is " << foo() << std::endl;
 }
 virtual int foo() { return 42; }
};

class derived : public base {
 unique_ptr<int> ptr_;
public:
 derived(int i) : ptr_(new int(i*i)) { }
 // The following cannot be called before derived::derived due to how C++ behaves,
 // if it was possible... Kaboom!
 int foo() override { return *ptr_; }
};

int main() {
 derived d(4);
}

```

= 0virtual ◦ ;◦

```

struct Abstract {
 virtual void f() = 0;
};

struct Concrete {
 void f() override {}
}

```

```
};

Abstract a; // Error.
Concrete c; // Good.
```

◦ ◦ ◦

```
struct DefaultAbstract {
 virtual void f() = 0;
};
void DefaultAbstract::f() {}

struct WhyWouldWeDoThis : DefaultAbstract {
 void f() override { DefaultAbstract::f(); }
};
```

• ◦ ◦ virtual ◦ ◦

```
struct Interface {
 virtual ~Interface() = 0;
};
Interface::~~Interface() = default;

struct Implementation : Interface {};
// ~Implementation() is automatically defined by the compiler if not explicitly
// specified, meeting the "must be defined before instantiation" requirement.
```

• ◦

```
class SharedBase {
 State my_state;
 std::unique_ptr<Helper> my_helper;
 // ...

public:
 virtual void config(const Context& cont) = 0;
 // ...
};
/* virtual */ void SharedBase::config(const Context& cont) {
 my_helper = new Helper(my_state, cont.relevant_field);
 do_this();
 and_that();
}

class OneImplementation : public SharedBase {
 int i;
 // ...

public:
 void config(const Context& cont) override;
 // ...
};
void OneImplementation::config(const Context& cont) /* override */ {
 my_state = { cont.some_field, cont.another_field, i };
 SharedBase::config(cont);
 my_unique_setup();
};
```



```
// And so on, for other classes derived from SharedBase.
```

<https://riptutorial.com/zh-TW/cplusplus/topic/1752/>

# 120:

## Examples

---

---

ET ◦ ◦

◦ ◦ **ETET** ◦

◦

```
Vector vec_1, vec_2, vec_3;

// Initializing vec_1, vec_2 and vec_3.

Vector result = vec_1 + vec_2*vec_3;
```

Vector+\* ◦

**ET** Vectorresult

1. `vec_1 vec_2vec_3` ◦
2. `Vector_tmp_tmp = vec_2*vec_3;_tmp = vec_2*vec_3;` ◦
3. `result = vec_1 + _tmp;result result = vec_1 + _tmp;` ◦

**ET 2** Vector \_tmp Vector ◦

```
Vector result = vec_1 + (vec_2*vec3 + vec_1)*(vec_2 + vec_3*vec_1);
```

Vector vec\_1, vec\_2, vec\_3result ◦ ◦

---

**ET**

---

**ET**

1. **PAE** /◦ /◦ **PAE** ◦ **PAE** ◦
2. ◦ /◦

**ET** ◦

```
Vector vec_1, vec_2, vec_3;

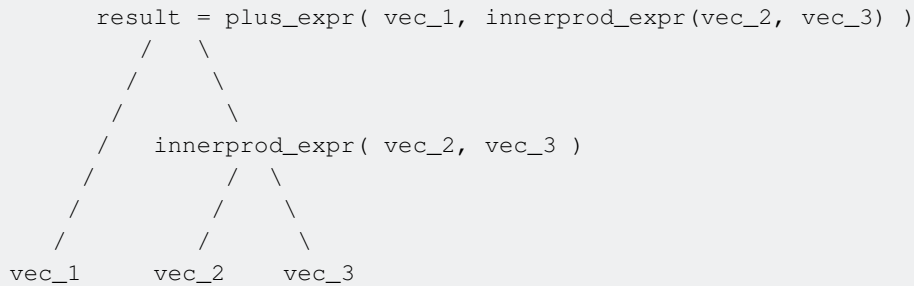
// Initializing vec_1, vec_2 and vec_3.

Vector result = vec_1 + vec_2*vec_3;
```

1. **plus\_expr**
2. **innerprod\_expr** ◦

## ET

- **ET. PAE** ◦ ◦



- `result result = result ◦ resultelem_resvec_1 elem_1 vec_2 elem_2 vec_3 elem_3`

```
elem_res = elem_1 + elem_2*elem_3;
```

Vector ◦

---

◦

---

## vec.hhstd :: vector ◦

```

#ifndef EXPR_VEC
define EXPR_VEC

include <vector>
include <cassert>
include <utility>
include <iostream>
include <algorithm>
include <functional>

///
/// This is a wrapper for std::vector. It's only purpose is to print out a log when a
/// vector constructions in called.
/// It wraps the indexed access operator [] and the size() method, which are
/// important for later ETs implementation.
///
// std::vector wrapper.
template<typename ScalarType> class Vector
{
public:
 explicit Vector() { std::cout << "ctor called.\n"; };
 explicit Vector(int size): _vec(size) { std::cout << "ctor called.\n"; };
 explicit Vector(const std::vector<ScalarType> &vec): _vec(vec)

```

```

{ std::cout << "ctor called.\n"; };

Vector(const Vector<ScalarType> & vec): _vec{vec()}
{ std::cout << "copy ctor called.\n"; };
Vector(Vector<ScalarType> && vec): _vec(std::move(vec()))
{ std::cout << "move ctor called.\n"; };

Vector<ScalarType> & operator=(const Vector<ScalarType> &) = default;
Vector<ScalarType> & operator=(Vector<ScalarType> &&) = default;

decltype(auto) operator[](int indx) { return _vec[indx]; }
decltype(auto) operator[](int indx) const { return _vec[indx]; }

decltype(auto) operator()() & { return (_vec); };
decltype(auto) operator()() const & { return (_vec); };
Vector<ScalarType> && operator()() && { return std::move(*this); }

int size() const { return _vec.size(); }

private:
 std::vector<ScalarType> _vec;
};

///
/// These are conventional overloads of operator + (the vector plus operation)
/// and operator * (the vector inner product operation) without using the expression
/// templates. They are later used for bench-marking purpose.
///

// + (vector plus) operator.
template<typename ScalarType>
auto operator+(const Vector<ScalarType> &lhs, const Vector<ScalarType> &rhs)
{
 assert(lhs().size() == rhs().size() &&
 "error: ops plus -> lhs and rhs size mismatch.");

 std::vector<ScalarType> _vec;
 _vec.resize(lhs().size());
 std::transform(std::cbegin(lhs()), std::cend(lhs()),
 std::cbegin(rhs()), std::begin(_vec),
 std::plus<>());
 return Vector<ScalarType>(std::move(_vec));
}

// * (vector inner product) operator.
template<typename ScalarType>
auto operator*(const Vector<ScalarType> &lhs, const Vector<ScalarType> &rhs)
{
 assert(lhs().size() == rhs().size() &&
 "error: ops multiplies -> lhs and rhs size mismatch.");

 std::vector<ScalarType> _vec;
 _vec.resize(lhs().size());
 std::transform(std::cbegin(lhs()), std::cend(lhs()),
 std::cbegin(rhs()), std::begin(_vec),
 std::multiplies<>());
 return Vector<ScalarType>(std::move(_vec));
}

#endif //!EXPR_VEC

```

# File expr.hhvector plusvector inner product

- 1. 1. **CRTP** .
- 2. **PAE** const .
- 3. **PAE binary\_operation** vector\_plusvector\_innerprod . **PAEPAE** . .
- 4. **vector\_plusvector\_innerprod** . **PAE** soperator +\***PAE** .

```
#ifndef EXPR_EXPR
define EXPR_EXPR

// Fwd declaration.
template<typename> class Vector;

namespace expr
{

// -----
//
// Section 1.
//
// The first section is a base class template for all kinds of expression. It
// employs the Curiously Recurring Template Pattern, which enables its instantiation
// to any kind of expression structure inheriting from it.
//
// -----

// Base class for all expressions.
template<typename Expr> class expr_base
{
public:
 const Expr& self() const { return static_cast<const Expr&>(*this); }
 Expr& self() { return static_cast<Expr&>(*this); }

protected:
 explicit expr_base() {};
 int size() const { return self().size_impl(); }
 auto operator[](int indx) const { return self().at_impl(indx); }
 auto operator()() const { return self()(); };
};

// -----
//
// The following section 2 & 3 are abstractions of pure algebraic expressions (PAE).
// Any PAE can be converted to a real object instance using operator(): it is in
// this conversion process, where the real computations are done.
//
//
// Section 2. Terminal
//
// A terminal is an abstraction wrapping a const reference to the Vector data
```

```

/// structure. It inherits from expr_base, therefore providing a unified interface
/// wrapping a Vector into a PAE.
///
/// It provides the size() method, indexed access through at_impl() and a conversion
/// to referenced object through () operator.
///
/// It might no be necessary for user defined data structures to have a terminal
/// wrapper, since user defined structure can inherit expr_base, therefore eliminates
/// the need to provide such terminal wrapper.
///
/// -----

/// Generic wrapper for underlying data structure.
template<typename DataType> class terminal: expr_base<terminal<DataType>>
{
public:
 using base_type = expr_base<terminal<DataType>>;
 using base_type::size;
 using base_type::operator[];
 friend base_type;

 explicit terminal(const DataType &val): _val(val) {}
 int size_impl() const { return _val.size(); };
 auto at_impl(int indx) const { return _val[indx]; };
 decltype(auto) operator()() const { return (_val); }

private:
 const DataType &_val;
};

/// -----
///
/// Section 3. Binary operation expression.
///
/// This is a PAE abstraction of any binary expression. Similarly it inherits from
/// expr_base.
///
/// It provides the size() method, indexed access through at_impl() and a conversion
/// to referenced object through () operator. Each call to the at_impl() method is
/// a element wise computation.
///
/// -----

/// Generic wrapper for binary operations (that are element-wise).
template<typename Ops, typename lExpr, typename rExpr>
class binary_ops: public expr_base<binary_ops<Ops,lExpr,rExpr>>
{
public:
 using base_type = expr_base<binary_ops<Ops,lExpr,rExpr>>;
 using base_type::size;
 using base_type::operator[];
 friend base_type;

 explicit binary_ops(const Ops &ops, const lExpr &lexpr, const rExpr &rexpr)
 : _ops(ops), _lexpr(lexpr), _rxpr(rxpr) {};
 int size_impl() const { return _lexpr.size(); };

 /// This does the element-wise computation for index indx.

```

```

auto at_impl(int indx) const { return _ops(_lexpr[indx], _rexpr[indx]); };

/// Conversion from arbitrary expr to concrete data type. It evaluates
/// element-wise computations for all indices.
template<typename DataType> operator DataType()
{
 DataType _vec(size());
 for(int _ind = 0; _ind < _vec.size(); ++_ind)
 _vec[_ind] = (*this)[_ind];
 return _vec;
}

private: /// Ops and expr are assumed cheap to copy.
Ops _ops;
lExpr _lexpr;
rExpr _rexpr;
};

/// -----
/// Section 4.
///
/// The following two structs defines algebraic operations on PAEs: here only vector
/// plus and vector inner product are implemented.
///
/// First, some element-wise operations are defined : in other words, vec_plus and
/// vec_prod acts on elements in Vectors, but not whole Vectors.
///
/// Then, operator + & * are overloaded on PAEs, such that: + & * operations on PAEs
/// also return PAEs.
///
/// -----

/// Element-wise plus operation.
struct vec_plus_t
{
 constexpr explicit vec_plus_t() = default;
 template<typename LType, typename RType>
 auto operator()(const LType &lhs, const RType &rhs) const
 { return lhs+rhs; }
};

/// Element-wise inner product operation.
struct vec_prod_t
{
 constexpr explicit vec_prod_t() = default;
 template<typename LType, typename RType>
 auto operator()(const LType &lhs, const RType &rhs) const
 { return lhs*rhs; }
};

/// Constant plus and inner product operator objects.
constexpr vec_plus_t vec_plus{};
constexpr vec_prod_t vec_prod{};

/// Plus operator overload on expressions: return binary expression.
template<typename lExpr, typename rExpr>
auto operator+(const lExpr &lhs, const rExpr &rhs)
{ return binary_ops<vec_plus_t, lExpr, rExpr>(vec_plus, lhs, rhs); }

```

```

 /// Inner prod operator overload on expressions: return binary expression.
 template<typename lExpr, typename rExpr>
 auto operator*(const lExpr &lhs, const rExpr &rhs)
 { return binary_ops<vec_prod_t, lExpr, rExpr>(vec_prod, lhs, rhs); }

} //!expr

#endif //!EXPR_EXPR

```

## main.ccsrc

```

include <chrono>
include <iomanip>
include <iostream>
include "vec.hh"
include "expr.hh"
include "boost/core/demangle.hpp"

int main()
{
 using dtype = float;
 constexpr int size = 5e7;

 std::vector<dtype> _vec1(size);
 std::vector<dtype> _vec2(size);
 std::vector<dtype> _vec3(size);

 // ... Initialize vectors' contents.

 Vector<dtype> vec1(std::move(_vec1));
 Vector<dtype> vec2(std::move(_vec2));
 Vector<dtype> vec3(std::move(_vec3));

 unsigned long start_ms_no_ets =
 std::chrono::duration_cast<std::chrono::milliseconds>
 (std::chrono::system_clock::now().time_since_epoch()).count();
 std::cout << "\nNo-ETs evaluation starts.\n";

 Vector<dtype> result_no_ets = vec1 + (vec2*vec3);

 unsigned long stop_ms_no_ets =
 std::chrono::duration_cast<std::chrono::milliseconds>
 (std::chrono::system_clock::now().time_since_epoch()).count();
 std::cout << std::setprecision(6) << std::fixed
 << "No-ETs. Time elapses: " << (stop_ms_no_ets-start_ms_no_ets)/1000.0
 << " s.\n" << std::endl;

 unsigned long start_ms_ets =
 std::chrono::duration_cast<std::chrono::milliseconds>
 (std::chrono::system_clock::now().time_since_epoch()).count();
 std::cout << "Evaluation using ETs starts.\n";

 expr::terminal<Vector<dtype>> vec4(vec1);
 expr::terminal<Vector<dtype>> vec5(vec2);
 expr::terminal<Vector<dtype>> vec6(vec3);

```



```

Vector<dtype> result_ets = (vec4 + vec5*vec6);

unsigned long stop_ms_ets =
 std::chrono::duration_cast<std::chrono::milliseconds>
 (std::chrono::system_clock::now().time_since_epoch()).count();
std::cout << std::setprecision(6) << std::fixed
 << "With ETs. Time eclapses: " << (stop_ms_ets-start_ms_ets)/1000.0
 << " s.\n" << std::endl;

auto ets_ret_type = (vec4 + vec5*vec6);
std::cout << "\nETs result's type:\n";
std::cout << boost::core::demangle(typeid(decltype(ets_ret_type)).name()) << '\n';

return 0;
}

```

### GCC 5.3-03 -std=c++14

```

ctor called.
ctor called.
ctor called.

No-ETs evaluation starts.
ctor called.
ctor called.
No-ETs. Time eclapses: 0.571000 s.

Evaluation using ETs starts.
ctor called.
With ETs. Time eclapses: 0.164000 s.

ETs result's type:
expr::binary_ops<expr::vec_plus_t, expr::terminal<Vector<float> >,
expr::binary_ops<expr::vec_prod_t, expr::terminal<Vector<float> >,
expr::terminal<Vector<float> > > >

```

- **ET** > 3x.
- **Vector** ◦ **ETsctor** ◦
- **Boost :: demangleET** ◦

- 
- **ET** ◦ ◦
  - **ET**<sub>auto</sub> ◦ **PAE**<sub>auto</sub> ◦

```

auto result = ...; // Some expensive expression:
 // auto returns the expr graph,
 // NOT the computed value.

for(auto i = 0; i < 100; ++i)
 ScalarType value = result* ... // Some other expensive computations using result.

```

**for** for ◦

- [boost :: protoET](#)。
- [EigenET](#)。

◦ ◦ `ArrayMatrix` ◦ [Bjarne Stroustrup“C ++”](#)。

## ◦ Matrix

```
template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
 using value_type = T;

 Matrix() : values(COL * ROW) {}

 static size_t cols() { return COL; }
 static size_t rows() { return ROW; }

 const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
 T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

private:
 std::vector<T> values;
};

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
 Matrix<T, COL, ROW> result;

 for (size_t y = 0; y != lhs.rows(); ++y) {
 for (size_t x = 0; x != lhs.cols(); ++x) {
 result(x, y) = lhs(x, y) + rhs(x, y);
 }
 }
 return result;
}
```

## Matrix

```
const std::size_t cols = 2000;
const std::size_t rows = 1000;

Matrix<double, cols, rows> a, b, c;

// initialize a, b & c
for (std::size_t y = 0; y != rows; ++y) {
 for (std::size_t x = 0; x != cols; ++x) {
 a(x, y) = 1.0;
 b(x, y) = 2.0;
 c(x, y) = 3.0;
 }
}
```

```
Matrix<double, cols, rows> d = a + b + c; // d(x, y) = 6
```

operator+()°

“”°

Matrix d = a + b + c° ((a + b) + c)operator+(operator+(a, b), c)° operator+()° 2° Matrix°

```
template<typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW> add3(const Matrix<T, COL, ROW>& a,
 const Matrix<T, COL, ROW>& b,
 const Matrix<T, COL, ROW>& c)
{
 Matrix<T, COL, ROW> result;
 for (size_t y = 0; y != ROW; ++y) {
 for (size_t x = 0; x != COL; ++x) {
 result(x, y) = a(x, y) + b(x, y) + c(x, y);
 }
 }
 return result;
}
```

**Matrix**Matrix::add2() Matrix::AddMultiply()°

Matrix d = a + b + c“”° a + b + c d°

operator+()Matrix“”°

2

```
template <typename LHS, typename RHS>
class MatrixSum
{
public:
 using value_type = typename LHS::value_type;

 MatrixSum(const LHS& lhs, const RHS& rhs) : rhs(rhs), lhs(lhs) {}

 value_type operator()(int x, int y) const {
 return lhs(x, y) + rhs(x, y);
 }
private:
 const LHS& lhs;
 const RHS& rhs;
};
```

operator+()

```
template <typename LHS, typename RHS>
MatrixSum<LHS, RHS> operator+(const LHS& lhs, const RHS& rhs) {
 return MatrixSum<LHS, RHS>(lhs, rhs);
}
```

operator+()2MatrixMatrix“”° ° °

MatrixSum<>

```
MatrixSum<Matrix<double>, Matrix<double> > SumAB(a, b);
```

d = a + b

```
for (std::size_t y = 0; y != a.rows(); ++y) {
 for (std::size_t x = 0; x != a.cols(); ++x) {
 d(x, y) = SumAB(x, y);
 }
}
```

abd °

° a + b + c

```
MatrixSum<MatrixSum<Matrix<double>, Matrix<double> >, Matrix<double> > SumABC(SumAB, c);
```

```
for (std::size_t y = 0; y != a.rows(); ++y) {
 for (std::size_t x = 0; x != a.cols(); ++x) {
 d(x, y) = SumABC(x, y);
 }
}
```

Matrix ° Matrix::operator=()

```
template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
 using value_type = T;

 Matrix() : values(COL * ROW) {}

 static size_t cols() { return COL; }
 static size_t rows() { return ROW; }

 const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
 T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

 template <typename E>
 Matrix<T, COL, ROW>& operator=(const E& expression) {
 for (std::size_t y = 0; y != rows(); ++y) {
 for (std::size_t x = 0; x != cols(); ++x) {
 values[y * COL + x] = expression(x, y);
 }
 }
 return *this;
 }

private:
 std::vector<T> values;
};
```

<https://riptutorial.com/zh-TW/cplusplus/topic/3404/>

# 121: Elision

## Examples

◦ ◦

```
std::string get_string()
{
 return std::string("I am a string.");
}
```

std::string /◦ ◦

C ++ /◦ ◦ ◦

1. /◦ //◦

2. elision /◦

C ++ 11

```
struct my_type
{
 my_type() = default;
 my_type(const my_type &) {std::cout <<"Copying\n";}
 my_type(my_type &&) {std::cout <<"Moving\n";}
};

my_type func()
{
 return my_type();
}
```

func " my\_type " "

" ;◦

/◦

elision elision ◦ ◦ ◦

C ++ 17

elision ◦ elision ◦ //◦

```
std::mutex a_mutex;
std::lock_guard<std::mutex> get_lock()
{
 return std::lock_guard<std::mutex>(a_mutex);
}
```

a\_mutex

std::lock\_guard C++

C++ 17

C++ 17

pre-C++ 17 C++ 17

C++ 17 prvalue prvalue C++ 17

prvalue C++ 17 get\_lock /

```
std::lock_guard the_lock = get_lock();
```

get\_lock prvalue ;the\_lock /

“C++ 17 ;C++

prvalue

```
std::mutex a_mutex;
std::lock_guard<std::mutex> get_lock()
{
 std::lock_guard<std::mutex> my_lock(a_mutex);
 //Do stuff
 return my_lock;
}
```

C++ 17 / lock\_guard ABI

```
struct trivially_copyable {
 int a;
};

void foo (trivially_copyable a) {}

foo(trivially_copyable{}); //copy elision not mandated
```

prvalue prvalue prvalue

```
std::string func()
{
 return std::string("foo");
}
```

.

prvalue prvalue

```
void func(std::string str) { ... }

func(std::string("foo"));
```

string str ◦ **elision**str+◦

explicit ◦ func("foo") stringconst char\*string ◦ explicit explicit◦ /◦

- return
- 
- 

/

```
std::string func()
{
 std::string str("foo");
 //Do stuff
 return str;
}
```

## elision

```
std::string func()
{
 std::string ret("foo");
 if(some_condition)
 {
 return "bar";
 }
 return ret;
}
```

Elid<sub>ret</sub> ◦

elision ◦

```
std::string func(std::string str)
{
 str.assign("foo");
 //Do stuff
 return str; //No elision possible
}
```

## elision

**prvalue**prvalue◦

```
std::string str = std::string("foo");
```

std::string str("foo"); ◦

```
std::string func()
{
 return std::string("foo");
}

std::string str = func();
```

copy elision `std::string` 2° 1°

Elision <https://riptutorial.com/zh-TW/cplusplus/topic/2489/elision>



# 122:

- 
- MyClass const MyClass other;
- MyClass MyClass other;
- MyClass volatile const MyClass other;
- MyClass volatile MyClass other;
- 
- MyClass operator = const MyClass rhs;
- MyClass operator = MyClass rhs;
- MyClass operator = MyClass rhs;
- const MyClass operator = const MyClass rhs;
- const MyClass operator = MyClass rhs;
- const MyClass operator = MyClass rhs;
- MyClass = const MyClass rhs;
- MyClass = MyClass rhs;
- MyClass = MyClass rhs;

**RHS** ◦ **MyClass operator = MyClass rhs;**

C ++

[GeeksForGeeks](#)

C ++

## Examples

◦

```
// Assignment Operator
#include <iostream>
#include <string>

using std::cout;
using std::endl;

class Foo
{
public:
 Foo(int data)
 {
 this->data = data;
 }
 ~Foo(){};
 Foo& operator=(const Foo& rhs)
 {
 data = rhs.data;
 }
};
```

```

 return *this;
 }

 int data;
};

int main()
{
 Foo foo(2); //Foo(int data) called
 Foo foo2(42);
 foo = foo2; // Assignment Operator Called
 cout << foo.data << endl; //Prints 42
}

```

foo foo2foo operator= function [http //cpp.sh/3qtbm](http://cpp.sh/3qtbm)

## Assignment Constructor . . .

```

// Copy Constructor
#include <iostream>
#include <string>

using std::cout;
using std::endl;

class Foo
{
public:
 Foo(int data)
 {
 this->data = data;
 }
 ~Foo(){};
 Foo(const Foo& rhs)
 {
 data = rhs.data;
 }

 int data;
};

int main()
{
 Foo foo(2); //Foo(int data) called
 Foo foo2 = foo; // Copy Constructor called
 cout << foo2.data << endl;
}

```

Foo foo2 = foo;main foo2intfoo [http //cpp.sh/5iu7](http://cpp.sh/5iu7)

. . .

```

// Copy vs Assignment Constructor
#include <iostream>
#include <string>

using std::cout;

```

```

using std::endl;

class Foo
{
public:
 Foo(int data)
 {
 this->data = data;
 }
 ~Foo(){};
 Foo(const Foo& rhs)
 {
 data = rhs.data;
 }

 Foo& operator=(const Foo& rhs)
 {
 data = rhs.data;
 return *this;
 }

 int data;
};

int main()
{
 Foo foo(2); //Foo(int data) / Normal Constructor called
 Foo foo2 = foo; //Copy Constructor Called
 cout << foo2.data << endl;

 Foo foo3(42);
 foo3=foo; //Assignment Constructor Called
 cout << foo3.data << endl;
}

```

```

2
2

```

Foo foo2 = foo; Foo foo2 = foo; ◦ ◦ **foo3**foo3=foo;

<https://riptutorial.com/zh-TW/cplusplus/topic/7158/>

# 123:

## Examples

`;const` ◦ `const` ◦ `const` ◦

```
const int x = 123;
x = 456; // error
int& r = x; // error

struct S {
 void f();
 void g() const;
};
const S s;
s.f(); // error
s.g(); // OK
```

## decltype

### C++ 11

◦

- `decltype(e)` ◦

```
int x = 42;
std::vector<decltype(x)> v(100, x); // v is a vector<int>
```

- `decltype(e)` ◦

```
struct S {
 int x = 42;
};
const S s;
decltype(s.x) y; // y has type int, even though s.x is const
```

- `decltype(e)`

- `decltype(e) T&` ◦
- `decltype(e) T&&` ◦
- `decltype(e) T`

◦

```
int f() { return 42; }
int& g() { static int x = 42; return x; }
int x = 42;
decltype(f()) a = f(); // a has type int
```

```
decltype(g()) b = g(); // b has type int&
decltype(x) c = x; // c has type int&, since x is an lvalue
```

## C++ 14

`decltype(auto)` returns `decltype auto`.

```
const int x = 123;
auto y = x; // y has type int
decltype(auto) z = x; // z has type const int, the declared type of x
```

◦

- `int` `signed` `signed int` ◦
- `char` `signed` `char` `char` ◦ `signed char` **-127+127** ◦
- `short` `long` `long` `long` ◦
- `signed` `bool` `wchar_t` `char16_t` `char32_t` ◦

```
signed char celsius_temperature;
std::cin >> celsius_temperature;
if (celsius_temperature < -35) {
 std::cout << "cold day, eh?\n";
}
```

◦

- `int` `unsigned` `unsigned int` ◦
- `unsigned` `char` `char` ◦ **255** ◦
- `unsigned` `short` `long` `long` ◦ `bool` `wchar_t` `char16_t` `char32_t` ◦

```
char invert_case_table[256] = { ..., 'a', 'b', 'c', ..., 'A', 'B', 'C', ... };
char invert_case(char c) {
 unsigned char index = c;
 return invert_case_table[index];
 // note: returning invert_case_table[c] directly does the
 // wrong thing on implementations where char is a signed type
}
```

`volatile` ◦ `const volatile` ◦

`memory_mapped_port` `volatile` `sizeof(int)` **1** ◦ `volatile` `sizeof(int)` ◦

```
extern volatile char memory_mapped_port;
void write_to_device(int x) {
 const char* p = reinterpret_cast<const char*>(&x);
 for (int i = 0; i < sizeof(int); i++) {
 memory_mapped_port = p[i];
 }
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7840/>

# 124:

CC ++. C ++. .

## Examples

RAII. . RAII. . std::auto\_ptr

```
#include <memory>
#include <iostream>
using namespace std;

int main() {
 {
 auto_ptr ap(new int(5)); // dynamic memory is the resource
 cout << *ap << endl; // prints 5
 } // auto_ptr is destroyed, its resource is automatically freed
}
```

## C++ 11

std::auto\_ptr

```
#include <memory>
#include <iostream>
using namespace std;

int main() {
 auto_ptr ap1(new int(5));
 cout << *ap1 << endl; // prints 5
 auto_ptr ap2(ap1); // copy ap2 from ap1; ownership now transfers to ap2
 cout << *ap2 << endl; // prints 5
 cout << ap1 == nullptr << endl; // prints 1; ap1 has lost ownership of resource
}
```

std::auto\_ptr. auto\_ptrs. . std::shared\_ptr

```
#include <memory>
#include <iostream>
using namespace std;

int main() {
 shared_ptr sp2;
 {
 shared_ptr sp1(new int(5)); // give ownership to sp1
 cout << *sp1 << endl; // prints 5
 sp2 = sp1; // copy sp2 from sp1; both have ownership of resource
 cout << *sp1 << endl; // prints 5
 cout << *sp2 << endl; // prints 5
 } // sp1 goes out of scope and is destroyed; sp2 has sole ownership of resource
 cout << *sp2 << endl;
} // sp2 goes out of scope; nothing has ownership, so resource is freed
```

。 。 。 1。 3。

|   | 1   | 2   |
|---|-----|-----|
| 1 | 1   |     |
| 2 |     | 1   |
| 3 | 112 |     |
| 4 |     | 112 |
| 5 | 2   |     |
| 6 |     | 2   |

23.21。 。

mutman of **mut** ual **ex** clusion。 “”。“” 。 。 。

C ++ 11

std::mutexC ++ 11。

```
#include <thread>
#include <mutex>
#include <iostream>
using namespace std;

void add_1(int& i, const mutex& m) { // function to be run in thread
 m.lock();
 i += 1;
 m.unlock();
}

int main() {
 int var = 1;
 mutex m;

 cout << var << endl; // prints 1

 thread t1(add_1, var, m); // create thread with arguments
 thread t2(add_1, var, m); // create another thread
 t1.join(); t2.join(); // wait for both threads to finish

 cout << var << endl; // prints 3
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/8336/>



# 125:

## Examples

1. ◦

```
class foo {
 int x;
public:
 int get_x();
 void set_x(int new_x);
};
```

2. ◦ ◦ ◦

```
class foo; // elaborated type specifier -> forward declaration
class bar {
public:
 bar(foo& f);
};
void baz();
class baz; // another elaborated type specifier; another forward declaration
 // note: the class has the same name as the function void baz()
class foo {
 bar b;
 friend class baz; // elaborated type specifier refers to the class,
 // not the function of the same name
public:
 foo();
};
```

3. ◦

```
template <class T>
const T& min(const T& x, const T& y) {
 return b < a ? b : a;
}
```

4. class ◦ class ◦ C++ ◦

```
template <template <class T> class U>
// ^^^^^ "class" used in this sense here;
// U is a template template parameter
void f() {
 U<int>::do_it();
 U<double>::do_it();
}
```

5. 23 ◦

```
template <class T>
```

```
class foo {
};

foo<class bar> x; // <- bar does not have to have previously appeared.
```

## C++ 11

6. ◦

```
enum class Format {
 TEXT,
 PDF,
 OTHER,
};
Format f = F::TEXT;
```

### class

- struct **base** struct public private ◦
- struct; class ◦

1. ◦

```
enum Direction {
 UP,
 LEFT,
 DOWN,
 RIGHT
};
Direction d = UP;
```

## C++ 11

C++ 11 enum class struct ◦ : TT ◦

```
enum class Format : char {
 TEXT,
 PDF,
 OTHER
};
Format f = Format::TEXT;

enum Language : int {
 ENGLISH,
 FRENCH,
 OTHER
};
```

enum ◦

```
Language l1, l2;

l1 = ENGLISH;
```

```
l2 = Language::OTHER;
```

2. . . .

```
enum Foo { FOO };
void Foo() {}
Foo foo = FOO; // ill-formed; Foo refers to the function
enum Foo foo = FOO; // ok; Foo refers to the enum type
```

## C++ 11

3. . .

unscoped . .

.

```
enum class Format; // underlying type is implicitly int
void f(Format f);
enum class Format {
 TEXT,
 PDF,
 OTHER,
};

enum Direction; // ill-formed; must specify underlying type
```

1. .

```
// Example is from POSIX
union sigval {
 int sival_int;
 void *sival_ptr;
};
```

2. . . .

```
union foo; // elaborated type specifier -> forward declaration
class bar {
 public:
 bar(foo& f);
};
void baz();
union baz; // another elaborated type specifier; another forward declaration
 // note: the class has the same name as the function void baz()
union foo {
 long l;
 union baz* b; // elaborated type specifier refers to the class,
 // not the function of the same name
};
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7838/>

# 126:

mTm° Tm ° C ++ - °

C ++C ++**out-arguments** out-argument° ° Barbara LiskovLiskovLSP°

°

virtualvirtual°

## Examples

1.

```
// 1. Base example not using language support for covariance, dynamic type checking.
```

```
class Top
{
public:
 virtual Top* clone() const = 0;
 virtual ~Top() = default; // Necessary for `delete` via Top*.
};

class D : public Top
{
public:
 Top* clone() const override
 { return new D(*this); }
};

class DD : public D
{
private:
 int answer_ = 42;

public:
 int answer() const
 { return answer_;}

 Top* clone() const override
 { return new DD(*this); }
};

#include <assert.h>
#include <iostream>
#include <typeinfo>
using namespace std;

int main()
{
 cout << boolalpha;

 DD* p1 = new DD();
 Top* p2 = p1->clone();
```

```

bool const correct_dynamic_type = (typeid(*p2) == typeid(DD));
cout << correct_dynamic_type << endl; // "true"

assert(correct_dynamic_type); // This is essentially dynamic type checking. :(
auto p2_most_derived = static_cast<DD*>(p2);
cout << p2_most_derived->answer() << endl; // "42"
delete p2;
delete p1;
}

```

## 2.°

```

// 2. Covariant result version of the base example, static type checking.

class Top
{
public:
 virtual Top* clone() const = 0;
 virtual ~Top() = default; // Necessary for `delete` via Top*.
};

class D : public Top
{
public:
 D* /* ← Covariant return */ clone() const override
 { return new D(*this); }
};

class DD : public D
{
private:
 int answer_ = 42;

public:
 int answer() const
 { return answer_; }

 DD* /* ← Covariant return */ clone() const override
 { return new DD(*this); }
};

#include <iostream>
using namespace std;

int main()
{
 DD* p1 = new DD();
 DD* p2 = p1->clone();
 // Correct dynamic type DD for *p2 is guaranteed by the static type checking.

 cout << p2->answer() << endl; // "42"
 delete p2;
 delete p1;
}

```

## 3.°

```

// 3. Covariant smart pointer result (automated cleanup).

#include <memory>
using std::unique_ptr;

template< class Type >
auto up(Type* p) { return unique_ptr<Type>(p); }

class Top
{
private:
 virtual Top* virtual_clone() const = 0;

public:
 unique_ptr<Top> clone() const
 { return up(virtual_clone()); }

 virtual ~Top() = default; // Necessary for `delete` via Top*.
};

class D : public Top
{
private:
 D* /* ← Covariant return */ virtual_clone() const override
 { return new D(*this); }

public:
 unique_ptr<D> /* ← Apparent covariant return */ clone() const
 { return up(virtual_clone()); }
};

class DD : public D
{
private:
 int answer_ = 42;

 DD* /* ← Covariant return */ virtual_clone() const override
 { return new DD(*this); }

public:
 int answer() const
 { return answer_; }

 unique_ptr<DD> /* ← Apparent covariant return */ clone() const
 { return up(virtual_clone()); }
};

#include <iostream>
using namespace std;

int main()
{
 auto p1 = unique_ptr<DD>(new DD());
 auto p2 = p1->clone();
 // Correct dynamic type DD for *p2 is guaranteed by the static type checking.

 cout << p2->answer() << endl; // "42"
 // Cleanup is automated via unique_ptr.
}

```

# 127:

## Examples

switch◦

```
// print the numbers to a file, one per line
for (const int num : num_list) {
 errno = 0;
 fprintf(file, "%d\n", num);
 if (errno == ENOSPC) {
 fprintf(stderr, "no space left on device; output will be truncated\n");
 break;
 }
}
```

◦

```
int sum = 0;
for (int i = 0; i < N; i++) {
 int x;
 std::cin >> x;
 if (x < 0) continue;
 sum += x;
 // equivalent to: if (x >= 0) sum += x;
}
```

do-while ◦

```
// Gets the next non-whitespace character from standard input
char read_char() {
 char c;
 do {
 c = getchar();
 } while (isspace(c));
 return c;
}
```

forC ++ 11for ◦

```
// print 10 asterisks
for (int i = 0; i < 10; i++) {
 putchar('*');
}
```

while ◦

```
int i = 0;
// print 10 asterisks
while (i < 10) {
 putchar('*');
}
```

```
 i++;
}
```

```
std::vector<int> primes = {2, 3, 5, 7, 11, 13};

for(auto prime : primes) {
 std::cout << prime << std::endl;
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7841/>



# 128:

## Examples

### C

```
// This creates an array with 5 values.
const int array[] = { 1, 2, 3, 4, 5 };

#ifdef BEFORE_CPP11

// You can use `sizeof` to determine how many elements are in an array.
const int* first = array;
const int* afterLast = first + sizeof(array) / sizeof(array[0]);

// Then you can iterate over the array by incrementing a pointer until
// it reaches past the end of our array.
for (const int* i = first; i < afterLast; ++i) {
 std::cout << *i << std::endl;
}

#else

// With C++11, you can let the STL compute the start and end iterators:
for (auto i = std::begin(array); i != std::end(array); ++i) {
 std::cout << *i << std::endl;
}

#endif
```

### 15

1  
2  
3  
4

```
const int array[] = { 1, 2, 3, 4, 5 };
```

### 5. C.

```
const int* first = array;
const int* afterLast = first + sizeof(array) / sizeof(array[0]);
```

◦ ◦ **C** sizeof ◦ ◦

```
for (const int* i = first; i < afterLast; ++i) {
```

◦ iafterLast iarray◦

```
std::cout << *i << std::endl;
```

i◦ dereference\*i◦

\_\_\_\_\_

◦ ◦

```
A B C
```

```
+---+---+---+---+
| A | B | C | |
+---+---+---+---+
```

◦ ◦ **A** ◦ erase(A) ◦

\_\_\_\_\_

```
auto my_iterator = my_vector.begin(); // position
auto my_value = *my_iterator; // value
```

◦ end()

```
+---+---+---+---+
| A | B | C | |
+---+---+---+---+
 ↑ ↑
 | +-- An iterator here has no value. Do not dereference it!
+----- An iterator here dereferences to the value A.
```

**C++** begin()end() ◦ firstlast

```
+---+---+---+---+
| A | B | C | |
+---+---+---+---+
 ↑ ↑
 | |
+- first +- last
```

```
+---+
| |
+---+
```

begin()end()

```
+---+
| |
+---+
 ↑
 |
+- empty_sequence.begin()
 |
+- empty_sequence.end()
```

```

+---+---+---+
| A | B | C |
+---+---+---+
^ ^ ^
| | |
+- first +- last

```

◦

- insert
- erase
- 



◦ ◦

```

std::vector<int>::iterator first;
{
 std::vector<int> foo;
 first = foo.begin(); // first is now valid
} // foo falls out of scope and is destroyed
// At this point first is now invalid

```

## C++ ◦ ◦



◦ ◦

```

auto first = my_vector.begin();
++first; // advance the iterator 1 position
std::advance(first, 1); // advance the iterator 1 position
first = std::next(first); // returns iterator to the next element
std::advance(first, -1); // advance the iterator 1 position
backwards
first = std::next(first, 20); // returns iterator to the element 20
position forward
first = std::prev(first, 5); // returns iterator to the element 5
position backward
auto dist = std::distance(my_vector.begin(), first); // returns distance between two
iterators.

```

## std :: distance firstsecond ◦

◦ a = b + 3; b = a; ++b; ++b; ++b; **3**b = a; ++b; ++b; ++b; ◦ ◦



## C++ ◦ ◦

- ◦ ◦
- Forward Iterators◦
- ◦
-

- 
- C++ 17.

- random\_shuffle ◦

---

- 

```
template<class Iter>
Iter find(Iter first, Iter last, typename std::iterator_traits<Iter>::value_type val) {
 while (first != last) {
 if (*first == val)
 return first;
 ++first;
 }
 return last;
}
```

```
template<class BidirIt>
void test(BidirIt a, std::bidirectional_iterator_tag) {
 std::cout << "Bidirectional iterator is used" << std::endl;
}

template<class ForwIt>
void test(ForwIt a, std::forward_iterator_tag) {
 std::cout << "Forward iterator is used" << std::endl;
}

template<class Iter>
void test(Iter a) {
 test(a, typename std::iterator_traits<Iter>::iterator_category());
}
```

- 
- reverse\_iterator ◦ base() ◦

- rbegin() rend() ◦

```
std::vector<int> v{1, 2, 3, 4, 5};
for (std::vector<int>::reverse_iterator it = v.rbegin(); it != v.rend(); ++it)
{
 cout << *it;
} // prints 54321
```

- base() ◦ base()

```
std::vector<int>::reverse_iterator r = v.rbegin();
std::vector<int>::iterator i = r.base();
assert(&*r == &*(i-1)); // always true if r, (i-1) are dereferenceable
 // and are not proxy iterators
```

```
+---+---+---+---+---+---+---+
| | 1 | 2 | 3 | 4 | 5 | |
```

```

+---+---+---+---+---+
 ↑ ↑ ↑ ↑
 | | | |
rend() | rbegin() end()
 | |
 begin() rbegin().base()
 rend().base()

```

```

+---+---+---+---+---+
| 1 | 2 | 3 | 4 | 5 |
+---+---+---+---+---+
 ↑ ↑
 | |
 | end()
 | rbegin()
begin() rbegin().base()
rend()
rend().base()

```

beginiterator◦

enditeratorend◦

**vector** const\_iterator const beginendconst\_iterator ◦ const\_iterator const cbegincend ◦

```

#include <vector>
#include <iostream>

int main() {
 std::vector<int> v = { 1, 2, 3, 4, 5 }; //intialize vector using an initializer_list

 for (std::vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
 std::cout << *it << " ";
 }

 return 0;
}

```

1 2 3 4 5

## Map Iterator

◦

**map** const\_iterator **const** const\_iterator ◦ iterator ◦

```

// Create a map and insert some values
std::map<char,int> mymap;
mymap['b'] = 100;
mymap['a'] = 200;
mymap['c'] = 300;

// Iterate over all tuples
for (std::map<char,int>::iterator it = mymap.begin(); it != mymap.end(); ++it)
 std::cout << it->first << " => " << it->second << '\n';

```

a => 200  
b => 100  
c => 300

```
// Data stream. Any number of various whitespace characters will be OK.
std::istringstream istr("1\t 2 3 4");
std::vector<int> v;

// Constructing stream iterators and copying data from stream into vector.
std::copy(
 // Iterator which will read stream data as integers.
 std::istream_iterator<int>(istr),
 // Default constructor produces end-of-stream iterator.
 std::istream_iterator<int>(),
 std::back_inserter(v));

// Print vector contents.
std::copy(v.begin(), v.end(),
 //Will print values to standard output as integers delimited by " -- ".
 std::ostream_iterator<int>(std::cout, " -- "));
```

1 -- 2 -- 3 -- 4 --。

“”。

## C++

```
template<class T>
struct generator_iterator {
 using difference_type=std::ptrdiff_t;
 using value_type=T;
 using pointer=T*;
 using reference=T;
 using iterator_category=std::input_iterator_tag;
 std::optional<T> state;
 std::function< std::optional<T>() > operation;
 // we store the current element in "state" if we have one:
 T operator*() const {
 return *state;
 }
 // to advance, we invoke our operation. If it returns a nullopt
 // we have reached the end:
 generator_iterator& operator++() {
 state = operation();
 return *this;
 }
 generator_iterator operator++(int) {
 auto r = *this;
 ++(*this);
 return r;
 }
 // generator iterators are only equal if they are both in the "end" state:
 friend bool operator==(generator_iterator const& lhs, generator_iterator const& rhs) {
 if (!lhs.state && !rhs.state) return true;
 return false;
 }
 friend bool operator!=(generator_iterator const& lhs, generator_iterator const& rhs) {
 return !(lhs==rhs);
 }
};
```

```
}
// We implicitly construct from a std::function with the right signature:
generator_iterator(std::function< std::optional<T>() > f):operation(std::move(f))
{
 if (operation)
 state = operation();
}
// default all special member functions:
generator_iterator(generator_iterator &&) =default;
generator_iterator(generator_iterator const&) =default;
generator_iterator& operator=(generator_iterator &&) =default;
generator_iterator& operator=(generator_iterator const&) =default;
generator_iterator() =default;
};
```

◦

◦

std::function◦ ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/473/>

## 129:

- *function\_name* [ *function\_args* ] [ *function\_attributes* ] [ *function\_qualifiers* ] -> *trailing-return-type* [ *requires\_clause* ]

◦

◦ ->◦

## Examples

```
class ClassWithAReallyLongName {
 public:
 class Iterator { /* ... */ };
 Iterator end();
};
```

end

```
auto ClassWithAReallyLongName::end() -> Iterator { return Iterator(); }
```

end

```
ClassWithAReallyLongName::Iterator ClassWithAReallyLongName::end() { return Iterator(); }
```

“”◦

## Lambda

lambda ;lambdas◦ lambda◦

```
struct Base {};
struct Derived1 : Base {};
struct Derived2 : Base {};
auto lambda = [](bool b) -> Base* { if (b) return new Derived1; else return new Derived2; };
// ill-formed: auto lambda = Base* [](bool b) { ... };
```

<https://riptutorial.com/zh-TW/cplusplus/topic/4142/>



# 130:

thisC ++◦ this

```
this.someMember();
```

->

```
this->someMember();
```

this

'this'

<http://www.geeksforgeeks.org/this-pointer-in-c/>

[https://www.tutorialspoint.com/cplusplus/cpp\\_this\\_pointer.htm](https://www.tutorialspoint.com/cplusplus/cpp_this_pointer.htm)

## Examples

this ;◦ this ;◦

```
struct ThisPointer {
 int i;

 ThisPointer(int ii);

 virtual void func();

 int get_i() const;
 void set_i(int ii);
};
ThisPointer::ThisPointer(int ii) : i(ii) {}
// Compiler rewrites as:
ThisPointer::ThisPointer(int ii) : this->i(ii) {}
// Constructor is responsible for turning allocated memory into 'this'.
// As the constructor is responsible for creating the object, 'this' will not be "fully"
// valid until the instance is fully constructed.

/* virtual */ void ThisPointer::func() {
 if (some_external_condition) {
 set_i(182);
 } else {
 i = 218;
 }
}
// Compiler rewrites as:
/* virtual */ void ThisPointer::func(ThisPointer* this) {
 if (some_external_condition) {
 this->set_i(182);
 } else {
 this->i = 218;
 }
}
```

```

int ThisPointer::get_i() const { return i; }
// Compiler rewrites as:
int ThisPointer::get_i(const ThisPointer* this) { return this->i; }

void ThisPointer::set_i(int ii) { i = ii; }
// Compiler rewrites as:
void ThisPointer::set_i(ThisPointer* this, int ii) { this->i = ii; }

```

this;◦ thisV◦

---

◦ ◦ ctd\_goodctd\_badCtorThisBaseCtorThisBase()CtorThisCtorThis()CtorThisDerived ◦ ◦

```

class CtorThisBase {
 short s;

public:
 CtorThisBase() : s(516) {}
};

class CtorThis : public CtorThisBase {
 int i, j, k;

public:
 // Good constructor.
 CtorThis() : i(s + 42), j(this->i), k(j) {}

 // Bad constructor.
 CtorThis(int ii) : i(ii), j(this->k), k(b ? 51 : -51) {
 virt_func();
 }

 virtual void virt_func() { i += 2; }
};

class CtorThisDerived : public CtorThis {
 bool b;

public:
 CtorThisDerived() : b(true) {}
 CtorThisDerived(int ii) : CtorThis(ii), b(false) {}

 void virt_func() override { k += (2 * i); }
};

// ...

CtorThisDerived ctd_good;
CtorThisDerived ctd_bad(3);

```

- ctd\_good
  - CtorThisBaseCtorThis◦ si◦
  - ij(this->i)◦ ij◦
  - jk(j)◦ jk◦
- ctd\_bad
  - kj(this->k)◦ kj◦

- CtorThisDerivedCtorThisCtorThis◦ bk◦
- ctd\_badCtorThis()CtorThis CtorThisDerived()CtorThisDerivedvtable◦ virt\_func()  
CtorThis::virt\_func() CtorThisDerived::virt\_func() ◦

this◦

```
// Example for this pointer
#include <iostream>
#include <string>

using std::cout;
using std::endl;

class Class
{
public:
 Class();
 ~Class();
 int getPrivateNumber () const;
private:
 int private_number = 42;
};

Class::Class(){}
Class::~Class(){}

int Class::getPrivateNumber() const
{
 return this->private_number;
}

int main()
{
 Class class_example;
 cout << class_example.getPrivateNumber() << endl;
}
```

◦

.....

```
// Dog Class Example
#include <iostream>
#include <string>

using std::cout;
using std::endl;

/*
 * @class Dog
 * @member name
 * Dog's name
 * @function bark
 * Dog Barks!
 * @function getName
 * To Get Private
 * Name Variable
 */
```

```

*/
class Dog
{
public:
 Dog(std::string name);
 ~Dog();
 void bark() const;
 std::string getName() const;
private:
 std::string name;
};

Dog::Dog(std::string name)
{
 /*
 * this->name is the
 * name variable from
 * the class dog . and
 * name is from the
 * parameter of the function
 */
 this->name = name;
}

Dog::~Dog() {}

void Dog::bark() const
{
 cout << "BARK" << endl;
}

std::string Dog::getName() const
{
 return this->name;
}

int main()
{
 Dog dog("Max");
 cout << dog.getName() << endl;
 dog.bark();
}

```

```
this->name = name;
```

Dogthis-> name.

[http //cpp.sh/75r7](http://cpp.sh/75r7)

## Pointer CV-Qualifiers

thisCV. this; CV.

```

struct ThisCVQ {
 void no_qualifier() {} // "this" is: ThisCVQ*
 void c_qualifier() const {} // "this" is: const ThisCVQ*
}

```

```

void v_qualifier() volatile {} // "this" is: volatile ThisCVQ*
void cv_qualifier() const volatile {} // "this" is: const volatile ThisCVQ*
};

```

this **CV-** ◦

```

struct CVOverload {
 int func() { return 3; }
 int func() const { return 33; }
 int func() volatile { return 333; }
 int func() const volatile { return 3333; }
};

```

this **const const volatile** ◦ **mutable** ◦ **const** ◦

◦ ◦ ◦

◦ 10 ◦

**C++** **const** ◦

```

class DoSomethingComplexAndOrExpensive {
 mutable ResultType cached_result;
 mutable bool state_changed;

 ResultType calculate_result();
 void modify_somewhat(const Param& p);

 // ...

public:
 DoSomethingComplexAndOrExpensive(Param p) : state_changed(true) {
 modify_somewhat(p);
 }

 void change_state(Param p) {
 modify_somewhat(p);
 state_changed = true;
 }

 // Return some complex and/or expensive-to-calculate result.
 // As this has no reason to modify logical state, it is marked as "const".
 ResultType get_result() const;
};

ResultType DoSomethingComplexAndOrExpensive::get_result() const {
 // cached_result and state_changed can be modified, even with a const "this" pointer.
 // Even though the function doesn't modify logical state, it does modify physical state
 // by caching the result, so it doesn't need to be recalculated every time the function
 // is called. This is indicated by cached_result and state_changed being mutable.

 if (state_changed) {
 cached_result = calculate_result();
 state_changed = false;
 }

 return cached_result;
}

```

```
}
```

const\_cast<this> -CV- mutable ◦ const\_cast constmutable ◦ ◦

const CV; CV const UB ◦

```
class CVAccessor {
 int arr[5];

public:
 const int& get_arr_element(size_t i) const { return arr[i]; }

 int& get_arr_element(size_t i) {
 return const_cast<int&>(const_cast<const CVAccessor*>(this)->get_arr_element(i));
 }
};
```

◦

---

this volatile const volatile ◦ volatile ◦

---

CV this CV

- CV ◦
- const const const volatile ◦
- volatile volatile const volatile ◦
- const volatile const volatile ◦

const ◦

```
struct CVAccess {
 void func() {}
 void func_c() const {}
 void func_v() volatile {}
 void func_cv() const volatile {}
};

CVAccess cva;
cva.func(); // Good.
cva.func_c(); // Good.
cva.func_v(); // Good.
cva.func_cv(); // Good.

const CVAccess c_cva;
c_cva.func(); // Error.
c_cva.func_c(); // Good.
c_cva.func_v(); // Error.
c_cva.func_cv(); // Good.

volatile CVAccess v_cva;
v_cva.func(); // Error.
v_cva.func_c(); // Error.
v_cva.func_v(); // Good.
```

```
v_cva.func_cv(); // Good.

const volatile CVAccess cv_cva;
cv_cva.func(); // Error.
cv_cva.func_c(); // Error.
cv_cva.func_v(); // Error.
cv_cva.func_cv(); // Good.
```

## C++ 11

this **cvref-qualifiers**\*this ◦ Ref-qualifiersnormalrvalue\*this ◦

ref-qualifiers this ◦ ref-qualifiers\*this; ◦

```
struct RefQualifiers {
 std::string s;

 RefQualifiers(const std::string& ss = "The nameless one.") : s(ss) {}

 // Normal version.
 void func() & { std::cout << "Accessed on normal instance " << s << std::endl; }
 // Rvalue version.
 void func() && { std::cout << "Accessed on temporary instance " << s << std::endl; }

 const std::string& still_a_pointer() & { return this->s; }
 const std::string& still_a_pointer() && { this->s = "Bob"; return this->s; }
};

// ...

RefQualifiers rf("Fred");
rf.func(); // Output: Accessed on normal instance Fred
RefQualifiers{}.func(); // Output: Accessed on temporary instance The nameless one
```

ref-qualifiers; ◦ cv-qualifiersref-qualifiersconst ◦

```
struct RefCV {
 void func() & {}
 void func() && {}
 void func() const& {}
 void func() const&& {}
 void func() volatile& {}
 void func() volatile&& {}
 void func() const volatile& {}
 void func() const volatile&& {}
};
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7146/>

# 131:

- ◦
  1. ::
  2. [] () T(...) . -> ++ -- dynamic\_cast static\_cast reinterpret\_cast const\_cast typeid
  3. ++ -- \* & + - ! ~ sizeof new delete delete[] ; C (T) ... ; C ++ 11 sizeof... alignof noexcept
  4. .\*->\*
  5. \* /%
  6. +-
  7. <<>>
  8. < > <= >=
  9. == !=
  10. & AND
  11. ^
  12. |
  13. &&
  14. ||
  15. ?:
  16. = \*= /= %= += -= >>= <<= &= ^= |=
  17. throw
  18. ,

- ◦
  - ?: ◦ ◦ a ? b , c : d ◦
  - ?throwa = b ? c : da = (b ? c : d)throw a ? b : cthrow (a ? b : c) ◦
  - :a ? b : c = da ? b : (c = d) ◦

## Examples

### C ++

- ( ) ◦ ◦

```
// volume of a spherical shell = 4 pi R^3 - 4 pi r^3
double vol = 4.0*pi*R*R*R/3.0 - 4.0*pi*r*r*r/3.0;

//Addition:

int a = 2+4/2; // equal to: 2+(4/2) result: 4
int b = (3+3)/2; // equal to: (3+3)/2 result: 3

//With Multiplication

int c = 3+4/2*6; // equal to: 3+((4/2)*6) result: 15
```



```
int d = 3*(3+6)/9; // equal to: (3*(3+6))/9 result: 3

//Division and Modulo

int g = 3-3%1; // equal to: 3 % 1 = 0 3 - 0 = 3
int h = 3-(3%1); // equal to: 3 % 1 = 0 3 - 0 = 3
int i = 3-3/1%3; // equal to: 3 / 1 = 3 3 % 3 = 0 3 - 0 = 3
int l = 3-(3/1)%3; // equal to: 3 / 1 = 3 3 % 3 = 0 3 - 0 = 3
int m = 3-(3/(1%3)); // equal to: 1 % 3 = 1 3 / 1 = 3 3 - 3 = 0
```

## ANDOR

### C ++ORAND。

```
// You can drive with a foreign license for up to 60 days
bool can_drive = has_domestic_license || has_foreign_license && num_days <= 60;
```

```
// You can drive with a foreign license for up to 60 days
bool can_drive = has_domestic_license || (has_foreign_license && num_days <= 60);
```

◦ ◦

## &&||

### &&||。

### C ++&&||。

### ||true。

```
#include <iostream>
#include <string>

using namespace std;

bool True(string id){
 cout << "True" << id << endl;
 return true;
}

bool False(string id){
 cout << "False" << id << endl;
 return false;
}

int main(){
 bool result;
 //let's evaluate 3 booleans with || and && to illustrate operator precedence
 //precedence does not mean that && will be evaluated first but rather where
 //parentheses would be added
 //example 1
 result =
 False("A") || False("B") && False("C");
 // eq. False("A") || (False("B") && False("C"))
```

```

//FalseA
//FalseB
//"Short-circuit evaluation skip of C"
//A is false so we have to evaluate the right of ||,
//B being false we do not have to evaluate C to know that the result is false

result =
 True("A") || False("B") && False("C");
 // eq. True("A") || (False("B") && False("C"))
cout << result << " :======" << endl;
//TrueA
//"Short-circuit evaluation skip of B"
//"Short-circuit evaluation skip of C"
//A is true so we do not have to evaluate
// the right of || to know that the result is true
//If || had precedence over && the equivalent evaluation would be:
// (True("A") || False("B")) && False("C")
//What would print
//TrueA
//"Short-circuit evaluation skip of B"
//FalseC
//Because the parentheses are placed differently
//the parts that get evaluated are differently
//which makes that the end result in this case would be False because C is false
}

```

o

## postfix

```

int a = 1;
++a; // result: 2
a--; // result: 1
int minusa=-a; // result: -1

bool b = true;
!b; // result: true

a=4;
int c = a++/2; // equal to: (a==4) 4 / 2 result: 2 ('a' incremented postfix)
cout << a << endl; // prints 5!
int d = ++a/2; // equal to: (a+1) == 6 / 2 result: 3

int arr[4] = {1,2,3,4};

int *ptr1 = &arr[0]; // points to arr[0] which is 1
int *ptr2 = ptr1++; // ptr2 points to arr[0] which is still 1; ptr1 incremented
std::cout << *ptr1++ << std::endl; // prints 2

int e = arr[0]++; // receives the value of arr[0] before it is incremented
std::cout << e << std::endl; // prints 1
std::cout << *ptr2 << std::endl; // prints arr[0] which is now 2

```

<https://riptutorial.com/zh-TW/cplusplus/topic/3895/>

# 132:

C++ → `<string>+ operator`

- `""`
- `.*`
- `::`
- `?:`
- `dynamic_cast static_cast reinterpret_cast const_cast typeid sizeof alignof noexcept`
- `###`

99.98

- `&&|| bool`
- `,`
- `&`

`&&|| C++ 17 ,`

## Examples

- `+=`
- `--=`
- `**=`
- `//=`
- `&&=`
- `||=`
- `^^=`
- `>>>>=`
- `<<<<=`

class / struct

```
//operator+ should be implemented in terms of operator+=
T operator+(T lhs, const T& rhs)
{
 lhs += rhs;
 return lhs;
}

T& operator+=(T& lhs, const T& rhs)
{
 //Perform addition
 return lhs;
}
```

class / struct

```
//operator+ should be implemented in terms of operator+=
T operator+(const T& rhs)
{
 *this += rhs;
 return *this;
}

T& operator+=(const T& rhs)
{
 //Perform addition
 return *this;
}
```

---

operator+**const**constconst ◦

1. Object foobar = foo + bar;foo
2. const operator+operator+=

const&◦ ◦

---

operator+=◦

const ◦ const&**const**◦

## 2

- ++foofoo++
- --foofoo--

++-- ◦

class / struct

```
//Prefix operator ++foo
T& operator++(T& lhs)
{
 //Perform addition
 return lhs;
}

//Postfix operator foo++ (int argument is used to separate pre- and postfix)
//Should be implemented in terms of ++foo (prefix operator)
T operator++(T& lhs, int)
{
 T t(lhs);
 ++lhs;
 return t;
}
```

class / struct

```

//Prefix operator ++foo
T& operator++()
{
 //Perform addition
 return *this;
}

//Postfix operator foo++ (int argument is used to separate pre- and postfix)
//Should be implemented in terms of ++foo (prefix operator)
T operator++(int)
{
 T t(*this);
 ++(*this);
 return t;
}

```

---

◦ const◦

---

◦ const ◦

“”constconst ◦

for++postfix ++◦ for++“”◦ for++

```

for (list<string>::const_iterator it = tokens.begin();
 it != tokens.end();
 ++it) { // Don't use it++
 ...
}

```

- ==!=
- ><
- >=<=

2 ==< ◦

class / struct

```

//Only implement those 2
bool operator==(const T& lhs, const T& rhs) { /* Compare */ }
bool operator<(const T& lhs, const T& rhs) { /* Compare */ }

//Now you can define the rest
bool operator!=(const T& lhs, const T& rhs) { return !(lhs == rhs); }
bool operator>(const T& lhs, const T& rhs) { return rhs < lhs; }
bool operator<=(const T& lhs, const T& rhs) { return !(lhs > rhs); }
bool operator>=(const T& lhs, const T& rhs) { return !(lhs < rhs); }

```

class / struct

```

//Note that the functions are const, because if they are not const, you wouldn't be able
//to call them if the object is const

```

```
//Only implement those 2
bool operator==(const T& rhs) const { /* Compare */ }
bool operator<(const T& rhs) const { /* Compare */ }

//Now you can define the rest
bool operator!=(const T& rhs) const { return !(*this == rhs); }
bool operator>(const T& rhs) const { return rhs < *this; }
bool operator<=(const T& rhs) const { return !(*this > rhs); }
bool operator>=(const T& rhs) const { return !(*this < rhs); }
```

bool truefalse ◦

const& ◦ & const **reference** ◦

class / struct const const ◦

◦

class / struct

```
operator T() const { /* return something */ }
```

const const ◦

```
struct Text
{
 std::string text;

 // Now Text can be implicitly converted into a const char*
 /*explicit*/ operator const char*() const { return text.data(); }
 // ^^^^^^^
 // to disable implicit conversion
};

Text t;
t.text = "Hello world!";

//Ok
const char* copyoftext = t;
```

operator [] ◦

99.982 const const const [] ◦

const & **value** const ◦

[]

```
std::vector<int> v{ 1 };
v[0] = 2; //Changes value of 1 to 2
 //wouldn't be possible if not returned by reference
```

class / struct

```
//I is the index type, normally an int
T& operator[](const I& index)
{
 //Do something
 //return something
}

//I is the index type, normally an int
const T& operator[](const I& index) const
{
 //Do something
 //return something
}
```

[][]...°

```
template<class T>
class matrix {
 // class enabling [][] overload to access matrix elements
 template <class C>
 class proxy_row_vector {
 using reference = decltype(std::declval<C>()[0]);
 using const_reference = decltype(std::declval<C const>()[0]);
 public:
 proxy_row_vector(C& _vec, std::size_t _r_ind, std::size_t _cols)
 : vec(_vec), row_index(_r_ind), cols(_cols) {}
 const_reference operator[](std::size_t _col_index) const {
 return vec[row_index*cols + _col_index];
 }
 reference operator[](std::size_t _col_index) {
 return vec[row_index*cols + _col_index];
 }
 private:
 C& vec;
 std::size_t row_index; // row index to access
 std::size_t cols; // number of columns in matrix
 };

 using const_proxy = proxy_row_vector<const std::vector<T>>;
 using proxy = proxy_row_vector<std::vector<T>>;
 public:
 matrix() : mtx(), rows(0), cols(0) {}
 matrix(std::size_t _rows, std::size_t _cols)
 : mtx(_rows*_cols), rows(_rows), cols(_cols) {}

 // call operator[] followed by another [] call to access matrix elements
 const_proxy operator[](std::size_t _row_index) const {
 return const_proxy(mtx, _row_index, cols);
 }

 proxy operator[](std::size_t _row_index) {
 return proxy(mtx, _row_index, cols);
 }
 private:
 std::vector<T> mtx;
 std::size_t rows;
};
```

```
std::size_t cols;
};
```

## operator ()

class / struct

```
//R -> Return type
//Types -> any different type
R operator() (Type name, Type2 name2, ...)
{
 //Do something
 //return something
}

//Use it like this (R is return type, a and b are variables)
R foo = object(a, b, ...);
```

```
struct Sum
{
 int operator() (int a, int b)
 {
 return a + b;
 }
};

//Create instance of struct
Sum sum;
int result = sum(1, 1); //result == 2
```

◦

class / struct<sup>""</sup>◦ class / struct ◦

=◦

- - ;
  - ◦
- ◦ ◦
- classstruct◦
- \*this ◦ int b = (a = 6) + 4; //b == 10 ◦

```
//T is some type
T& operator=(const T& other)
{
 //Do something (like copying values)
 return *this;
}
```

otherconst&operator=const ◦



class / struct = class / struct ◦ ◦

class / struct class / struct ◦

## NOT

NOT ~ ◦

class / struct

```
T operator~(T lhs)
{
 //Do operation
 return lhs;
}
```

class / struct

```
T operator~()
{
 T t(*this);
 //Do operation
 return t;
}
```

---

operator~ ◦ const int a = ~a + 1; ◦

class / struct this ◦

## I/O

<<>> “”

- std::ostream << std::cout
- std::istream >> std::cin

class / struct “”

- std::ostream std::cout << a << b; ◦ std::ostream&
- lhs
- rhsT const& rhs ◦ const Vector& ◦

```
//Overload std::ostream operator<< to allow output from Vector's
std::ostream& operator<<(std::ostream& lhs, const Vector& rhs)
{
 lhs << "x: " << rhs.x << " y: " << rhs.y << " z: " << rhs.z << '\n';
 return lhs;
}

Vector v = { 1, 2, 3};
```

```
//Now you can do
std::cout << v;
```

**+ - \*/complex<T>°**

°

```
#include <type_traits>

namespace not_std{

using std::decay_t;

//-----
// complex< value_t >
//-----

template<typename value_t>
struct complex
{
 value_t x;
 value_t y;

 complex &operator += (const value_t &x)
 {
 this->x += x;
 return *this;
 }
 complex &operator += (const complex &other)
 {
 this->x += other.x;
 this->y += other.y;
 return *this;
 }

 complex &operator -= (const value_t &x)
 {
 this->x -= x;
 return *this;
 }
 complex &operator -= (const complex &other)
 {
 this->x -= other.x;
 this->y -= other.y;
 return *this;
 }

 complex &operator *= (const value_t &s)
 {
 this->x *= s;
 this->y *= s;
 return *this;
 }
 complex &operator *= (const complex &other)
 {
 (*this) = (*this) * other;
 return *this;
 }
}
```

```

 complex &operator /= (const value_t &s)
 {
 this->x /= s;
 this->y /= s;
 return *this;
 }
 complex &operator /= (const complex &other)
 {
 (*this) = (*this) / other;
 return *this;
 }

 complex(const value_t &x, const value_t &y)
 : x{x}
 , y{y}
 {}

 template<typename other_value_t>
 explicit complex(const complex<other_value_t> &other)
 : x{static_cast<const value_t &>(other.x)}
 , y{static_cast<const value_t &>(other.y)}
 {}

 complex &operator = (const complex &) = default;
 complex &operator = (complex &&) = default;
 complex(const complex &) = default;
 complex(complex &&) = default;
 complex() = default;
};

// Absolute value squared
template<typename value_t>
value_t absqr(const complex<value_t> &z)
{ return z.x*z.x + z.y*z.y; }

//-----
// operator - (negation)
//-----

template<typename value_t>
complex<value_t> operator - (const complex<value_t> &z)
{ return {-z.x, -z.y}; }

//-----
// operator +
//-----

template<typename left_t,typename right_t>
auto operator + (const complex<left_t> &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a.x + b.x)>>
{ return{a.x + b.x, a.y + b.y}; }

template<typename left_t,typename right_t>
auto operator + (const left_t &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a + b.x)>>
{ return{a + b.x, b.y}; }

template<typename left_t,typename right_t>
auto operator + (const complex<left_t> &a, const right_t &b)
-> complex<decay_t<decltype(a.x + b)>>
{ return{a.x + b, a.y}; }

```

```

//-----
// operator -
//-----

template<typename left_t,typename right_t>
auto operator - (const complex<left_t> &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a.x - b.x)>>
{ return{a.x - b.x, a.y - b.y}; }

template<typename left_t,typename right_t>
auto operator - (const left_t &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a - b.x)>>
{ return{a - b.x, - b.y}; }

template<typename left_t,typename right_t>
auto operator - (const complex<left_t> &a, const right_t &b)
-> complex<decay_t<decltype(a.x - b)>>
{ return{a.x - b, a.y}; }

//-----
// operator *
//-----

template<typename left_t, typename right_t>
auto operator * (const complex<left_t> &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a.x * b.x)>>
{
 return {
 a.x*b.x - a.y*b.y,
 a.x*b.y + a.y*b.x
 };
}

template<typename left_t, typename right_t>
auto operator * (const left_t &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a * b.x)>>
{ return {a * b.x, a * b.y}; }

template<typename left_t, typename right_t>
auto operator * (const complex<left_t> &a, const right_t &b)
-> complex<decay_t<decltype(a.x * b)>>
{ return {a.x * b, a.y * b}; }

//-----
// operator /
//-----

template<typename left_t, typename right_t>
auto operator / (const complex<left_t> &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a.x / b.x)>>
{
 const auto r = absqr(b);
 return {
 (a.x*b.x + a.y*b.y) / r,
 (-a.x*b.y + a.y*b.x) / r
 };
}

template<typename left_t, typename right_t>
auto operator / (const left_t &a, const complex<right_t> &b)

```

```

-> complex<decay_t<decltype(a / b.x)>>
{
 const auto s = a/absqr(b);
 return {
 b.x * s,
 -b.y * s
 };
}

template<typename left_t, typename right_t>
auto operator / (const complex<left_t> &a, const right_t &b)
-> complex<decay_t<decltype(a.x / b)>>
{ return {a.x / b, a.y / b}; }

} // namespace not_std

int main(int argc, char **argv)
{
 using namespace not_std;

 complex<float> fz{4.0f, 1.0f};

 // makes a complex<double>
 auto dz = fz * 1.0;

 // still a complex<double>
 auto idz = 1.0f/dz;

 // also a complex<double>
 auto one = dz * idz;

 // a complex<double> again
 auto one_again = fz * idz;

 // Operator tests, just to make sure everything compiles.

 complex<float> a{1.0f, -2.0f};
 complex<double> b{3.0, -4.0};

 // All of these are complex<double>
 auto c0 = a + b;
 auto c1 = a - b;
 auto c2 = a * b;
 auto c3 = a / b;

 // All of these are complex<float>
 auto d0 = a + 1;
 auto d1 = 1 + a;
 auto d2 = a - 1;
 auto d3 = 1 - a;
 auto d4 = a * 1;
 auto d5 = 1 * a;
 auto d6 = a / 1;
 auto d7 = 1 / a;

 // All of these are complex<double>
 auto e0 = b + 1;
 auto e1 = 1 + b;
 auto e2 = b - 1;
 auto e3 = 1 - b;

```

```

auto e4 = b * 1;
auto e5 = 1 * b;
auto e6 = b / 1;
auto e7 = 1 / b;

return 0;
}

```

## C++“C++。

```

namespace named_operator {
 template<class D>struct make_operator{constexpr make_operator(){} };

 template<class T, char, class O> struct half_apply { T&& lhs; };

 template<class Lhs, class Op>
 half_apply<Lhs, '*', Op> operator*(Lhs&& lhs, make_operator<Op>) {
 return {std::forward<Lhs>(lhs)};
 }

 template<class Lhs, class Op, class Rhs>
 auto operator*(half_apply<Lhs, '*', Op>&& lhs, Rhs&& rhs)
 -> decltype(named_invoke(std::forward<Lhs>(lhs.lhs), Op{}, std::forward<Rhs>(rhs)))
 {
 return named_invoke(std::forward<Lhs>(lhs.lhs), Op{}, std::forward<Rhs>(rhs));
 }
}

```

。

```

namespace my_ns {
 struct append_t : named_operator::make_operator<append_t> {};
 constexpr append_t append{};

 template<class T, class A0, class A1>
 std::vector<T, A0> named_invoke(std::vector<T, A0> lhs, append_t, std::vector<T, A1> const&
rhs) {
 lhs.insert(lhs.end(), rhs.begin(), rhs.end());
 return std::move(lhs);
 }
}
using my_ns::append;

std::vector<int> a {1,2,3};
std::vector<int> b {4,5,6};

auto c = a *append* b;

```

append\_t:named\_operator::make\_operator<append\_t>append

append\_t:named\_operator::make\_operator<append\_t>。

## named\_invokelhsappend\_trhs。

lhs\*append\_t half\_apply。 half\_apply\*rhsnamed\_invoke( lhs, append\_t, rhs )。

append\_tADLnamed\_invoke。

## std::array

```
template<class=void, std::size_t...Is>
auto indexer(std::index_sequence<Is...>) {
 return [] (auto&& f) {
 return f(std::integral_constant<std::size_t, Is>{}...);
 };
}
template<std::size_t N>
auto indexer() { return indexer(std::make_index_sequence<N>{}); }

namespace my_ns {
 struct e_times_t : named_operator::make_operator<e_times_t> {};
 constexpr e_times_t e_times{};

 template<class L, class R, std::size_t N,
 class Out=std::decay_t<decltype(std::declval<L const&>()*std::declval<R const&>())>
 >
 std::array<Out, N> named_invoke(std::array<L, N> const& lhs, e_times_t, std::array<R, N>
 const& rhs) {
 using result_type = std::array<Out, N>;
 auto index_over_N = indexer<N>();
 return index_over_N([&] (auto...is) -> result_type {
 return {{
 (lhs[is] * rhs[is])...
 }};
 });
 }
}
```

◦

## C◦

lhs \*element\_wise<'+'>\* rhs ◦

\*dot\*\*cross\* product◦

\*+ ◦ C ++extra () s◦

->\*then\*std::function monadic ->\*bind\* ◦ Op

```
named_operator<'+'> append = [] (auto lhs, auto&& rhs) {
 using std::begin; using std::end;
 lhs.insert(end(lhs), begin(rhs), end(rhs));
 return std::move(lhs);
};
```

## C ++17◦

<https://riptutorial.com/zh-TW/cplusplus/topic/562/>

# 133:

- ◦ ◦
- ◦ ◦
- ◦ ◦
- ◦ Class c(value) Class◦
  - Class c = value ◦ Class◦
  - Nonclass c = classObject ◦ ◦
  - R &r = classObject r ◦ ◦
  - Class c{1, 2, 3} ◦ Class◦

## Examples

◦

```
void f(int x);
void f(double x);
f(42); // calls f(int)
```

CV◦

```
void f(int& x);
void f(double x);
int x = 42;
f(x); // argument type is int; exact match with int&

void g(const int& x);
void g(int x);
g(x); // ambiguous; both overloads give exact match
```

“ T ” T T\*◦

```
void f(int* p);
void f(void* p);

void g(int* p);
void g(int (&p)[100]);

int a[100];
f(a); // calls f(int*); exact match with array-to-pointer conversion
g(a); // ambiguous; both overloads give exact match
```

“ ”◦

- void f(int a); f(42);
- void f(std::string s); f("hello");
-



```
void f(...); f(42);
```

- ```
void f(std::vector<int> v); f({1, 2, 3});
```

◦

◦ ◦

◦

```
void f(int x);
struct S {
    void f(double x);
    void g() { f(42); } // calls S::f because global f is not visible here,
                       // even though it would be a better match
};
```

◦ ◦

```
class C {
public:
    static void f(double x);
private:
    static void f(int x);
};
C::f(42); // Error! Calls private C::f(int) even though public C::f(double) is viable.
```

explicit

```
struct X {
    explicit X(int );
    X(char );
};

void foo(X );
foo({4}); // X(int) is better much, but expression is
          // ill-formed because selected constructor is explicit
```

```
struct A {
    A() = default;           // #1
    A(A const& ) = default; // #2

    template <class T>
    A(T&& );                 // #3
};
```

A◦

```
A a; // calls #1
A b(a); // calls #3!
```

```
A(A const& ); // #2
A(A& );      // #3, with T = A&
```

#3#2CV°

SFINAE

```
template <class T,
          class = std::enable_if_t<!std::is_convertible<std::decay_t<T>*, A*>::value>
          >
A(T&& );
```

AA° - °

1. ° °

2. ° °

```
void f(char);           // (1)
void f(int ) = delete; // (2)
void f();              // (3)
void f(int& );        // (4)

f(4); // 1,2 are viable (even though 2 is deleted!)
      // 3 is not viable because the argument lists don't match
      // 4 is not viable because we cannot bind a temporary to
      //      a non-const lvalue reference
```

3. ° F1F2 F1F2...

3.1° F1F2

```
void f(int ); // (1)
void f(char ); // (2)

f(4); // call (1), better conversion sequence
```

3.2° F1F2

```
struct A
{
    operator int();
    operator double();
} a;

int i = a; // a.operator int() is better than a.operator double() and a conversion
float f = a; // ambiguous
```

3.3° F1F2

```
struct A
{
```

```

operator X&(); // #1
operator X&&(); // #2
};
A a;
X& lx = a; // calls #1
X&& rx = a; // calls #2

```

3.4° F1F2

```

template <class T> void f(T ); // #1
void f(int ); // #2

f(42); // calls #2, the non-template

```

3.5° F1F2F1F2°

```

template <class T> void f(T ); // #1
template <class T> void f(T* ); // #2

int* p;
f(p); // calls #2, more specialized

```

° °

```

struct A {
    A(A const& ); // #1

    template <class T>
    A(T&& ); // #2, not constrained
};

A a;
A b(a); // calls #2!
// #1 is not a template but #2 resolves to
// A(A& ), which is a less cv-qualified reference than #1
// which makes it a better implicit conversion sequence

```

```

void f(double ) { }
void f(float ) { }

f(42); // error: ambiguous

```

°

```

void f(int x);
void f(short x);
signed char c = 42;
f(c); // calls f(int); promotion to int is better than conversion to short
short s = 42;
f(s); // calls f(short); exact match is better than promotion to int

```

floatdouble°

```
void f(double x);
void f(long double x);
f(3.14f); // calls f(double); promotion to double is better than conversion to long double
```

◦

```
void f(float x);
void f(long double x);
f(3.14); // ambiguous

void g(long x);
void g(long double x);
g(42); // ambiguous
g(3.14); // ambiguous
```

f◦

```
void f(int x);
void f(unsigned int x);
void f(long x);
void f(unsigned long x);
void f(long long x);
void f(unsigned long long x);
void f(double x);
void f(long double x);
```

```
struct A { int m; };
struct B : A {};
struct C : B {};
```

◦ ◦

```
struct Unrelated {
    Unrelated(B b);
};
void f(A a);
void f(Unrelated u);
B b;
f(b); // calls f(A)
```

void*◦

```
void f(A* p);
void f(void* p);
B b;
f(&b); // calls f(A*)
```

◦ “” ◦

```
void f(const A& a);
void f(const B& b);
C c;
f(c); // calls f(const B&)
```

```
B b;
A& r = b;
f(r); // calls f(const A&); the f(const B&) overload is not viable
```

◦

```
void f(int B::*p);
void f(int C::*p);
int A::*p = &A::m;
f(p); // calls f(int B::*)
```

constnessvolatility

T*const T*◦

```
struct Base {};
struct Derived : Base {};
void f(Base* pb);
void f(const Base* pb);
void f(const Derived* pd);
void f(bool b);

Base b;
f(&b); // f(Base*) is better than f(const Base*)
Derived d;
f(&d); // f(const Derived*) is better than f(Base*) though;
        // constness is only a "tie-breaker" rule
```

T&const T&◦

```
void f(int& r);
void f(const int& r);
int x;
f(x); // both overloads match exactly, but f(int&) is still better
const int y = 42;
f(y); // only f(const int&) is viable
```

constconstconst◦

```
class IntVector {
public:
    // ...
    int* data() { return m_data; }
    const int* data() const { return m_data; }
private:
    // ...
    int* m_data;
};
IntVector v1;
int* data1 = v1.data(); // Vector::data() is better than Vector::data() const;
                        // data1 can be used to modify the vector's data

const IntVector v2;
const int* data2 = v2.data(); // only Vector::data() const is viable;
                              // data2 can't be used to modify the vector's data
```

o

```
class AtomicInt {
public:
    // ...
    int load();
    int load() volatile;
private:
    // ...
};
AtomicInt a1;
a1.load(); // non-volatile overload preferred; no side effect
volatile AtomicInt a2;
a2.load(); // only volatile overload is viable; side effect
static_cast<volatile AtomicInt&>(a1).load(); // force volatile semantics for a1
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2021/>

134:

Examples

std :: recursive_mutex

- °
- ° °

```
std::atomic_int temp{0};
std::recursive_mutex _mutex;

//launch_deferred launches asynchronous tasks on the same thread id

auto future1 = std::async(
    std::launch::deferred,
    [&]()
    {
        std::cout << std::this_thread::get_id() << std::endl;

        std::this_thread::sleep_for(std::chrono::seconds(3));
        std::unique_lock<std::recursive_mutex> lock( _mutex);
        temp=0;

    });

auto future2 = std::async(
    std::launch::deferred,
    [&]()
    {
        std::cout << std::this_thread::get_id() << std::endl;
        while ( true )
        {
            std::this_thread::sleep_for(std::chrono::milliseconds(1));
            std::unique_lock<std::recursive_mutex> lock( _mutex,
std::try_to_lock);

            if ( temp < INT_MAX )
                temp++;

            cout << temp << endl;

        }
    });
future1.get();
future2.get();
```

<https://riptutorial.com/zh-TW/cplusplus/topic/9929/>

135:

◦ ◦

Examples

◦ C++ 111100 [CodeReview](#) ◦

```
#include <iostream>
#include <vector>
#include <cmath>

int main()
{
    int l = 100;
    bool isprime;
    std::vector<int> primes;
    primes.push_back(2);
    for (int no = 3; no < l; no += 2) {
        isprime = true;
        for (int primecount=0; primes[primecount] <= std::sqrt(no); ++primecount) {
            if (no % primes[primecount] == 0) {
                isprime = false;
                break;
            } else if (primes[primecount] * primes[primecount] > no) {
                std::cout << no << "\n";
                break;
            }
        }
        if (isprime) {
            std::cout << no << " ";
            primes.push_back(no);
        }
    }
    std::cout << "\n";
}
```

3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

2◦ - ◦

```
#include <iostream>
#include <vector>
#include <cmath>

std::vector<int> prime_list(int limit)
{
    bool isprime;
    std::vector<int> primes;
    primes.push_back(2);
    for (int no = 3; no < limit; no += 2) {
        isprime = true;
        for (int primecount=0; primes[primecount] <= std::sqrt(no); ++primecount) {
            if (no % primes[primecount] == 0) {
                isprime = false;
            }
        }
    }
}
```



```

        break;
    } else if (primes[primecount] * primes[primecount] > no) {
        break;
    }
}
if (isprime) {
    primes.push_back(no);
}
}
return primes;
}

int main()
{
    std::vector<int> primes = prime_list(100);
    for (std::size_t i = 0; i < primes.size(); ++i) {
        std::cout << primes[i] << ' ';
    }
    std::cout << '\n';
}

```

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

if◦ ◦ ◦

```

std::vector<int> prime_list(int limit)
{
    bool isprime;
    std::vector<int> primes;
    primes.push_back(2);
    for (int no = 3; no < limit; no += 2) {
        isprime = true;
        for (int primecount=0; primes[primecount] <= std::sqrt(no); ++primecount) {
            if (no % primes[primecount] == 0) {
                isprime = false;
                break;
            }
        }
        if (isprime) {
            primes.push_back(no);
        }
    }
    return primes;
}

```

◦ primecount◦ ◦ no^{“”}no◦ break◦

```

std::vector<int> prime_list(int limit)
{
    std::vector<int> primes{2};
    for (int n = 3; n < limit; n += 2) {
        bool isprime = true;
        for (int i=0; isprime && primes[i] <= std::sqrt(n); ++i) {
            isprime &= (n % primes[i] != 0);
        }
        if (isprime) {
            primes.push_back(n);
        }
    }
}

```

```

    }
    return primes;
}

```

main “range-for”

```

int main()
{
    std::vector<int> primes = prime_list(100);
    for (auto p : primes) {
        std::cout << p << ' ';
    }
    std::cout << '\n';
}

```

◦ ◦

CC ++ goto cleanup ◦ goto ◦ **return** ◦ goto cleanup ◦

```

short calculate(VectorStr **data) {
    short result = FALSE;
    VectorStr *vec = NULL;
    if (!data)
        goto cleanup; //< Could become return false

    // ... Calculation which 'new's VectorStr

    result = TRUE;
cleanup:
    delete [] vec;
    return result;
}

```

C ++RAII

```

struct VectorRAII final {
    VectorStr *data{nullptr};
    VectorRAII() = default;
    ~VectorRAII() {
        delete [] data;
    }
    VectorRAII(const VectorRAII &) = delete;
};

short calculate(VectorStr **data) {
    VectorRAII vec{};
    if (!data)
        return FALSE; //< Could become return false

    // ... Calculation which 'new's VectorStr and stores it in vec.data

    return TRUE;
}

```

◦ std::unique_ptr ◦ std::vector ◦ VectorRAII ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/7600/>

136:

C ++ . . .

- `asm string-literal ;`
- `noexcept //1`
- `noexcept constant-expression //2`
- `noexcept //2`
- `sizeof unary-expression`
- `sizeof type-id`
- `sizeof ... identifier //C ++ 11`
- `typename nested-name-specifier identifier //1`
- `typename nested-name-specifier template opt simple-template-id //1`
- `typename identifier opt //2`
- `typename ... identifier opt //2;C ++ 11`
- `typename identifier opt = type-id //2`
- `template < template-parameter-list > typename ... opt identifier opt //3`
- `template < template-parameter-list > typename identifier opt = id-expression //3`

- `alignas C ++ 11`
- `alignof C ++ 11`
- `asm`
- `auto C ++ 11 C ++ 11`
- `bool`
- `break`
- `case`
- `catch`
- `char`
- `char16_t C ++ 11`
- `char32_t C ++ 11`
- `class`
- `const`
- `constexpr C ++ 11`
- `const_cast`
- `continue`
- `decltype C ++ 11`
- `default`
- `delete C ++ 11`
- `do`
- `double`
- `dynamic_cast`
- `else`
- `enum`
- `explicit`
- `export`
- `extern`
- `false`
- `float`
- `for`

- friend
- goto
- if
- inline C++ 11 C++ 17
- int
- long
- mutable
- namespace
- new
- noexcept C++ 11
- nullptr C++ 11
- operator
- private
- protected
- public
- register
- reinterpret_cast
- return
- short
- signed
- sizeof
- static
- static_assert C++ 11
- static_cast
- struct
- switch
- template
- this
- thread_local C++ 11
- throw
- true
- try
- typedef
- typeid
- typename
- union
- unsigned
- using
- virtual
- void
- volatile
- wchar_t
- while

finaloverride ° °

and and_eq bitand bitor compl not not_eq or or_eq xorxor_eq&& &= & | ° ~ ! != || |= ^^= ° °

C++ °

-
-
-

-
-
-
- /
-

Examples

ASM

asm ◦

asm ◦ **C++**asmconstexpr◦

```
[[noreturn]] void halt_system() {
    asm("hlt");
}
```

1. ◦

```
class MyVector {
public:
    explicit MyVector(uint64_t size);
};
MyVector v1(100); // ok
uint64_t len1 = 100;
MyVector v2{len1}; // ok, len1 is uint64_t
int len2 = 100;
MyVector v3{len2}; // ill-formed, implicit conversion from int to uint64_t
```

C++ 11 C++ 11 explicit◦

```
struct S {
    explicit S(int x, int y);
};
S f() {
    return {12, 34}; // ill-formed
    return S{12, 34}; // ok
}
```

C++ 11

2. ◦

```
class C {
    const int x;
public:
    C(int x) : x(x) {}
}
```

```

    explicit operator int() { return x; }
};
C c(42);
int x = c; // ill-formed
int y = static_cast<int>(c); // ok; explicit conversion

```

noexcept

C++ 11

1. noexcept

```

#include <iostream>
#include <stdexcept>
void foo() { throw std::runtime_error("oops"); }
void bar() {}
struct S {};
int main() {
    std::cout << noexcept(foo()) << '\n'; // prints 0
    std::cout << noexcept(bar()) << '\n'; // prints 0
    std::cout << noexcept(1 + 1) << '\n'; // prints 1
    std::cout << noexcept(S()) << '\n'; // prints 1
}

```

bar() noexcept(bar()) **false** bar()

2. . . .

```

void f1() { throw std::runtime_error("oops"); }
void f2() noexcept(false) { throw std::runtime_error("oops"); }
void f3() {}
void f4() noexcept {}
void f5() noexcept(true) {}
void f6() noexcept {
    try {
        f1();
    } catch (const std::runtime_error&) {}
}

```

f4 f5f6 f6 f2 noexceptnoexcept(false) f1f3f3

C++ 17

noexcept f1 f2f3f4 f5f6 noexcept

```

void g1() {}
void g2() noexcept {}
void (*p1)() noexcept = &g1; // ill-formed, since g1 is not noexcept
void (*p2)() noexcept = &g2; // ok; types match
void (*p3)() = &g1; // ok; types match
void (*p4)() = &g2; // ok; implicit conversion

```

1. typename std::decay<T>T type typename **“““typename”””**

```
template <class T>
auto decay_copy(T&& r) -> typename std::decay<T>::type;
```

2. class

```
template <typename T>
const T& min(const T& x, const T& y) {
    return b < a ? b : a;
}
```

C++ 17

3. typename class

```
template <template <class T> typename U>
void f() {
    U<int>::do_it();
    U<double>::do_it();
}
```

sizeof

- ◦ **size**std::size_t◦
-
- sizeof◦
- sizeofvoid◦
- **sizeof**T&T&& sizeof(T) ◦
- sizeof◦ sizeof0. ◦
- char signed charunsigned char**1**.char◦ **88**char◦

*expr*sizeof(*expr*) sizeof(T) T*expr*◦

```
int a[100];
std::cout << "The number of bytes in `a` is: " << sizeof a;
memset(a, 0, sizeof a); // zeroes out the array
```

C++ 11

sizeof...◦

```
template <class... T>
void f(T&&...) {
    std::cout << "f was called with " << sizeof...(T) << " arguments\n";
}
```

void C++

1. void◦ void◦ void""◦

2. void *constvolatile. void. void.

3. voidC ++.

```
void vobject; // C2182
void *pv; // okay
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
    pint = (int *)pv;
```

C ++

1. .

```
volatile declarator ;
```

C ++

1. virtual.

```
virtual [type-specifiers] member-function-declarator
virtual [access-specifier] base-class-name
```

1. **type-specifiers**.

2. **member-function-declarator**.

3. **access-specifier**publicprotectedprivate. virtual.

4. **base-class-name**

1. this. . this.

```
this->member-identifier
```

this;sizeof. .

```
myDate.setMonth( 3 );
```

can be interpreted this way:

```
setMonth( &myDate, 3 );
```

The object's address is available from within the member function as the this pointer. Most uses of this are implicit. It is legal, though unnecessary, to explicitly use this when referring to members of the class. For example:

```
void Date::setMonth( int mn )
```

```

{
    month = mn;           // These three statements
    this->month = mn;     // are equivalent
    (*this).month = mn;
}

```

The expression `*this` is commonly used to return the current object from a member function:

```
return *this;
```

The `this` pointer is also used to guard against self-reference:

```

if (&Object != this) {
// do not execute in cases of self-reference

```

C ++

1. C ++trythrowcatch。
2. try。
3. throw - try。 throw。 。 std :: exception。 std :: exception。
4. trycatch。 catch。

```

    MyData md;
try {
    // Code that could throw an exception
    md = GetNetworkResource();
}
catch (const networkIOException& e) {
    // Code that executes when an exception of type
    // networkIOException is thrown in the try block
    // ...
    // Log error message in the exception object
    cerr << e.what();
}
catch (const myDataFormatException& e) {
    // Code that handles another exception type
    // ...
    cerr << e.what();
}

// The following syntax shows a throw expression
MyData GetNetworkResource()
{
    // ...
    if (IOSuccess == false)
        throw networkIOException("Unable to connect");
    // ...
    if (readError)
        throw myDataFormatException("Format error");
    // ...
}

```

try。 throw - 。 catch。 throwcatch。

```
try {
```

```

    throw CSomeOtherException();
}
catch(...) {
    // Catch all exceptions - dangerous!!!
    // Respond (perhaps only partially) to the exception, then
    // re-throw to pass the exception to some other handler
    // ...
    throw;
}

```

C++

1. . . . friend . .

2. .

```

class friend F
friend F;
class ForwardDeclared; // Class name is known.
class HasFriends
{
    friend int ForwardDeclared::IsAFriend(); // C2039 error expected
};

```

1. . ; .

2. . -> .

3. . . .

```

#include <iostream>

using namespace std;
class Point
{
    friend void ChangePrivate( Point & );
public:
    Point( void ) : m_i(0) {}
    void PrintPrivate( void ){cout << m_i << endl; }

private:
    int m_i;
};

void ChangePrivate ( Point &i ) { i.m_i++; }

int main()
{
    Point sPoint;
    sPoint.PrintPrivate();
    ChangePrivate(sPoint);
    sPoint.PrintPrivate();
    // Output: 0
    1
}

```

```
class B;

class A {
public:
    int Func1( B& b );

private:
    int Func2( B& b );
};

class B {
private:
    int _b;

    // A::Func1 is a friend function to class B
    // so A::Func1 has access to all members of B
    friend int A::Func1( B& );
};

int A::Func1( B& b ) { return b._b; } // OK
int A::Func2( B& b ) { return b._b; } // C2248
```

<https://riptutorial.com/zh-TW/cplusplus/topic/4891/>

137:

C++<random>◦◦◦

◦◦

◦◦◦

◦◦

Examples

std::random_device◦

```
#include <iostream>
#include <random>

int main()
{
    std::random_device crypto_random_generator;
    std::uniform_int_distribution<int> int_distribution(0,9);

    int actual_distribution[10] = {0,0,0,0,0,0,0,0,0,0};

    for(int i = 0; i < 10000; i++) {
        int result = int_distribution(crypto_random_generator);
        actual_distribution[result]++;
    }

    for(int i = 0; i < 10; i++) {
        std::cout << actual_distribution[i] << " ";
    }

    return 0;
}
```

std::random_device◦

std::random_device◦

[entropy](#) GCClibstdc++LLVMlibc++◦

◦◦◦

```
#include <iostream>
#include <random>

int main()
{
    std::default_random_engine pseudo_random_generator;
    std::uniform_int_distribution<int> int_distribution(0, 9);
```

```

int actual_distribution[10] = {0,0,0,0,0,0,0,0,0,0};

for(int i = 0; i < 10000; i++) {
    int result = int_distribution(pseudo_random_generator);
    actual_distribution[result]++;
}

for(int i = 0; i <= 9; i++) {
    std::cout << actual_distribution[i] << " ";
}

return 0;
}

```

[0,9]。

`std::uniform_int_distribution<T>`。 `std::uniform_real_distribution<T>`。

。

```

#include <iostream>
#include <random>

int main()
{
    std::default_random_engine pseudo_random_generator;
    std::uniform_int_distribution<int> int_distribution(0, 9);
    std::uniform_real_distribution<float> float_distribution(0.0, 1.0);
    std::discrete_distribution<int> rigged_dice({1,1,1,1,1,100});

    std::cout << int_distribution(pseudo_random_generator) << std::endl;
    std::cout << float_distribution(pseudo_random_generator) << std::endl;
    std::cout << (rigged_dice(pseudo_random_generator) + 1) << std::endl;

    return 0;
}

```

。 。 rigged_dice **05** 5100 / 105 。

<https://riptutorial.com/zh-TW/cplusplus/topic/1541/>

138:

- //
 - variable.member_function;
 - variable_pointer-> member_function;
- //
 - ret_type class_name :: member_functioncv-qualifiers {
 - ;
 - }
- //
 - class class_name {
 - virt-specifier ret_type member_functioncv-qualifiers virt-specifier-seq;
 - // virt-specifier“virtual”。
 - // cv-qualifiers“const”/“volatile”。
 - // virt-specifier-seq“”/“”。
 - }

staticclass / struct / union ◦ static ◦

◦

Examples

classstruct ◦ ; :: ◦

```
class CL {
public:
    void definedInside() {}
    void definedOutside();
};
void CL::definedOutside() {}
```

. -> ;this ◦

```
struct ST {
    ST(const std::string& ss = "Wolf", int ii = 359) : s(ss), i(ii) { }

    int get_i() const { return i; }
    bool compare_i(const ST& other) const { return (i == other.i); }

private:
    std::string s;
    int i;
};
```

```

ST st1;
ST st2("Species", 8472);

int i = st1.get_i(); // Can access st1.i, but not st2.i.
bool b = st1.compare_i(st2); // Can access st1 & st2.

```

/o /o

```

class Access {
public:
    Access(int i_ = 8088, int j_ = 8086, int k_ = 6502) : i(i_), j(j_), k(k_) {}

    int i;
    int get_k() const { return k; }
    bool private_no_more() const { return i_be_private(); }
protected:
    int j;
    int get_i() const { return i; }
private:
    int k;
    int get_j() const { return j; }
    bool i_be_private() const { return ((i > j) && (k < j)); }
};

```

getter mutator setter

```

class Encapsulator {
    int encapsulated;

public:
    int get_encapsulated() const { return encapsulated; }
    void set_encapsulated(int e) { encapsulated = e; }

    void some_func() {
        do_something_with(encapsulated);
    }
};

```

encapsulated;get_encapsulated() set_encapsulated() [getter setter]

o

```

struct HiddenBase {
    void f(int) { std::cout << "int" << std::endl; }
    void f(bool) { std::cout << "bool" << std::endl; }
    void f(std::string) { std::cout << "std::string" << std::endl; }
};

struct HidingDerived : HiddenBase {
    void f(float) { std::cout << "float" << std::endl; }
};

// ...

HiddenBase hb;
HidingDerived hd;

```



```

std::string s;

hb.f(1);    // Output: int
hb.f(true); // Output: bool
hb.f(s);    // Output: std::string;

hd.f(1.f);  // Output: float
hd.f(3);    // Output: float
hd.f(true); // Output: float
hd.f(s);    // Error: Can't convert from std::string to float.

```

hd.f(s) ;◦ **using-declaration**“”◦

```

struct HidingDerived : HiddenBase {
    // All members named HiddenBase::f shall be considered members of HidingDerived for
    lookup.
    using HiddenBase::f;

    void f(float) { std::cout << "float" << std::endl; }
};

// ...

HidingDerived hd;

hd.f(1.f); // Output: float
hd.f(3);   // Output: int
hd.f(true); // Output: bool
hd.f(s);   // Output: std::string

```

using◦

```

struct NamesHidden {
    virtual void hide_me()      {}
    virtual void hide_me(float) {}
    void hide_me(int)          {}
    void hide_me(bool)         {}
};

struct NameHider : NamesHidden {
    using NamesHidden::hide_me;

    void hide_me()      {} // Overrides NamesHidden::hide_me().
    void hide_me(int) {} // Hides NamesHidden::hide_me(int).
};

```

publicprotected **using**◦

```

struct ProMem {
    protected:
    void func() {}
};

struct BecomesPub : ProMem {
    using ProMem::func;
};

```

```
// ...

ProMem pm;
BecomesPub bp;

pm.func(); // Error: protected.
bp.func(); // Good.
```

◦

```
struct One {
    virtual void f() { std::cout << "One." << std::endl; }
};

struct Two : One {
    void f() override {
        One::f(); // this->One::f();
        std::cout << "Two." << std::endl;
    }
};

struct Three : Two {
    void f() override {
        Two::f(); // this->Two::f();
        std::cout << "Three." << std::endl;
    }
};

// ...

Three t;

t.f(); // Normal syntax.
t.Two::f(); // Calls version of f() defined in Two.
t.One::f(); // Calls version of f() defined in One.
```

`virtual` ◦ ;vtablevftable ◦ ◦

```
struct Base {
    virtual void func() { std::cout << "In Base." << std::endl; }
};

struct Derived : Base {
    void func() override { std::cout << "In Derived." << std::endl; }
};

void slicer(Base x) { x.func(); }

// ...

Base b;
Derived d;

Base *pb = &b, *pd = &d; // Pointers.
Base &rb = b, &rd = d; // References.

b.func(); // Output: In Base.
d.func(); // Output: In Derived.
```

```

pb->func(); // Output: In Base.
pd->func(); // Output: In Derived.

rb.func(); // Output: In Base.
rd.func(); // Output: In Derived.

slicer(b); // Output: In Base.
slicer(d); // Output: In Base.

```

```

pdBase* rdBase& func()Derived::func()Base::func() ;DerivedvtableBase::func()Derived::func()
slicer()Base::func() Derived DerivedBaseDerivedBase

```

virtualvirtual virtual

```

struct B {
    virtual void f() {}
};

struct D : B {
    void f() {} // Implicitly virtual, overrides B::f.
                // You'd have to check B to know that, though.
};

```

virtual

```

struct BadB {
    virtual void f() {}
};

struct BadD : BadB {
    virtual void f(int i) {} // Does NOT override BadB::f.
};

```

C++ 11

C++ 11override

```

struct CPP11B {
    virtual void f() {}
};

struct CPP11D : CPP11B {
    void f() override {}
    void f(int i) override {} // Error: Doesn't actually override anything.
};

```

.

virtual virtual

C++ 11

override

```

struct VB {
    virtual void f(); // "virtual" goes here.
    void g();
};
/* virtual */ void VB::f() {} // Not here.
virtual void VB::g() {} // Error.

```

virtualvirtual◦

```

struct BOverload {
    virtual void func() {}
    void func(int) {}
};

struct DOverload : BOverload {
    void func() override {}
    void func(int) {}
};

// ...

BOverload* bo = new DOverload;
bo->func(); // Calls DOverload::func().
bo->func(1); // Calls BOverload::func(int).

```

◦

Const

this **CV**const◦◦◦ constconstconst◦

constconst const◦ const constconstconst -ness◦ getterconst◦

```

class ConstIncorrect {
    Field fld;

public:
    ConstIncorrect(const Field& f) : fld(f) {} // Modifies.

    const Field& get_field() { return fld; } // Doesn't modify; should be const.
    void set_field(const Field& f) { fld = f; } // Modifies.

    void do_something(int i) { // Modifies.
        fld.insert_value(i);
    }
    void do_nothing() { } // Doesn't modify; should be const.
};

class ConstCorrect {
    Field fld;

public:
    ConstCorrect(const Field& f) : fld(f) {} // Not const: Modifies.

    const Field& get_field() const { return fld; } // const: Doesn't modify.
    void set_field(const Field& f) { fld = f; } // Not const: Modifies.
}

```

```

void do_something(int i) {                               // Not const: Modifies.
    fld.insert_value(i);
}
void do_nothing() const { }                             // const: Doesn't modify.
};

// ...

const ConstIncorrect i_cant_do_anything(make_me_a_field());
// Now, let's read it...
Field f = i_cant_do_anything.get_field();
// Error: Loses cv-qualifiers, get_field() isn't const.
i_cant_do_anything.do_nothing();
// Error: Same as above.
// Oops.

const ConstCorrect but_i_can(make_me_a_field());
// Now, let's read it...
Field f = but_i_can.get_field(); // Good.
but_i_can.do_nothing();         // Good.

```

ConstIncorrectConstCorrectConstCorrect **cv-qualifying**◦ constconstconst◦

<https://riptutorial.com/zh-TW/cplusplus/topic/5661/>

139:

C/ C++

◦ #include

#define

C ◦ #if #ifdef

d ◦ /

◦ ◦ ◦

Examples

◦ ◦ 2003 C++ §3.2

“” ◦ #ifndef #endif #define #ifndef #endif

```
// Foo.h
#ifndef FOO_H_INCLUDED
#define FOO_H_INCLUDED

class Foo // a class definition
{
};

#endif
```

◦

◦ FOO_H_INCLUDED ◦ ◦

◦

C++ #pragma once ◦ ISO C++

```
// Foo.h
#pragma once

class Foo
{
};
```

#pragma once #pragma - - ◦ #pragma once

C++ #include

-
- ◦
- - ◦
- **BasicPremiumPro** - ◦

a

```

#ifdef _WIN32
#include <windows.h> // and other windows system files
#endif
#include <cstdio>

bool remove_file(const std::string &path)
{
#ifdef _WIN32
    return DeleteFile(path.c_str());
#elif defined(_POSIX_VERSION) || defined(__unix__)
    return (0 == remove(path.c_str()));
#elif defined(__APPLE__)
    //TODO: check if NSAPI has a more specific function with permission dialog
    return (0 == remove(path.c_str()));
#else
#error "This platform is not supported"
#endif
}

```

`_WIN32 __APPLE__ unix__`◦

b

```

void s_PrintAppStateOnUserPrompt()
{
    std::cout << "-----BEGIN-DUMP-----\n"
              << AppState::Instance()->Settings().ToString() << "\n"
    #if ( 1 == TESTING_MODE ) //privacy: we want user details only when testing
        << ListToString(AppState::UndoStack()->GetActionNames())
        << AppState::Instance()->CrntDocument().Name()
        << AppState::Instance()->CrntDocument().SignatureSHA() << "\n"
    #endif
        << "-----END-DUMP-----\n"
}

```

c◦

```

void MainWindow::OnProcessButtonClick()
{
#ifdef _PREMIUM
    CreatePurchaseDialog("Buy App Premium", "This feature is available for our App Premium
users. Click the Buy button to purchase the Premium version at our website");
    return;
#endif
    //...actual feature logic here
}

```

◦ gcc -E

```
gcc -E -DOPTIMISE_FOR_OS_X -DTESTING_MODE=1 Sample.cpp
```

Sample.cpp#define OPTIMISE_FOR_OS_X#define TESTING_MODE 1Sample.cpp◦

```
◦ ◦ ◦ 0ENABLE_EXTRA_DEBUGGING = 0-DENABLE_EXTRA_DEBUGGING = 1◦ ◦ #ifndef
#error
```

```
#ifndef (ENABLE_EXTRA_DEBUGGING)
// please include DefaultDefines.h if not already included.
#   error "ENABLE_EXTRA_DEBUGGING is not defined"
#else
#   if ( 1 == ENABLE_EXTRA_DEBUGGING )
//code
#   endif
#endif
```

◦ ◦ ◦

```
// This is an object-like macro
#define    PI          3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define    AREA(r)    (PI*(r)*(r))

// They can be used like this:
double pi_macro    = PI;
double area_macro = AREA(4.6);
```

QtQObjectQ_OBJECT.◦

◦ ◦

◦ ◦

```
double pi_squared = PI * PI;
// Compiler sees:
double pi_squared = 3.14159265358979 * 3.14159265358979;

double area = AREA(5);
// Compiler sees:
double area = (3.14159265358979*(5)*(5))
```

AREA()◦ ◦

```
#define BAD_AREA(r) PI * r * r
```



```
double bad_area = BAD_AREA(5 + 1.6);
// Compiler sees:
double bad_area = 3.14159265358979 * 5 + 1.6 * 5 + 1.6;

double good_area = AREA(5 + 1.6);
// Compiler sees:
double good_area = (3.14159265358979*(5 + 1.6)*(5 + 1.6));
```

◦ ◦ ◦

```
int oops = 5;
double incremental_damage = AREA(oops++);
// Compiler sees:
double incremental_damage = (3.14159265358979*(oops++)*(oops++));
```

◦

“”
;◦

```
#define IF_BREAKER(Func) Func();

if (some_condition)
    // Oops.
    IF_BREAKER(some_func);
else
    std::cout << "I am accidentally an orphan." << std::endl;
```

if...elseelseif◦ “”◦

```
#define IF_FIXER(Func) Func()

if (some_condition)
    IF_FIXER(some_func);
else
    std::cout << "Hooray! I work again!" << std::endl;
```

◦

```
#define DO_SOMETHING(Func, Param) Func(Param, 2)

// ...

some_function(DO_SOMETHING(some_func, 3), DO_SOMETHING(some_func, 42));
```

◦ ◦ ◦ ◦

```
#define TEXT "I \
am \
many \
lines."
```

```
// ...

std::cout << TEXT << std::endl; // Output: I am many lines.
```

◦

```
#define CREATE_OUTPUT_AND_DELETE(Str) \
    std::string* tmp = new std::string(Str); \
    std::cout << *tmp << std::endl; \
    delete tmp;

// ...

CREATE_OUTPUT_AND_DELETE("There's no real need for this to use 'new'.")
```

◦ *0 do-while* ◦

```
#define DO_STUFF(Type, Param, ReturnVar) do { \
    Type temp(some_setup_values); \
    ReturnVar = temp.process(Param); \
} while (0)

int x;
DO_STUFF(MyClass, 41153.7, x);

// Compiler sees:

int x;
do {
    MyClass temp(some_setup_values);
    x = temp.process(41153.7);
} while (0);
```

◦ **“Varargs”** `__VA_ARGS__` ◦

```
#define VARIADIC(Param, ...) Param(__VA_ARGS__)

VARIADIC(sprintf, "%d", 8);
// Compiler sees:
sprintf("%d", 8);
```

`__VA_ARGS__` ◦

```
#define VARIADIC2(POne, PTwo, PThree, ...) POne(PThree, __VA_ARGS__, PTwo)

VARIADIC2(some_func, 3, 8, 6, 9);
// Compiler sees:
some_func(8, 6, 9, 3);
```

◦ **Visual Studio** ◦ `GCC__VA_ARGS__##` ◦ ◦

```
// In this example, COMPILER is a user-defined macro specifying the compiler being used.
```

```

#if COMPILER == "VS"
#define VARIADIC3(Name, Param, ...) Name(Param, __VA_ARGS__)
#elif COMPILER == "GCC"
#define VARIADIC3(Name, Param, ...) Name(Param, ##__VA_ARGS__)
#endif /* COMPILER */

```

◦ ◦

gcc3.0.0◦

```

#if __GNUC__ < 3
#error "This code requires gcc > 3.0.0"
#endif

```

Apple◦

```

#ifdef __APPLE__
#error "Apple products are not supported in this release"
#endif

```

◦ ◦

C ++

- `__LINE__` #line◦
- `__FILE__` #line◦
- `__DATE__` "Mmm dd yyyy"*Mmm*std::asctime()◦
- `__TIME__` "hh:mm:ss"◦
- `__cplusplus` C ++C ++◦ 199711LC ++ 98C ++ 03 201103LC ++ 11201402LC ++ 14◦

C ++ 11

- `__STDC_HOSTED__` 1 0◦

C ++ 17

- `__STDCPP_DEFAULT_NEW_ALIGNMENT__` size_t - operator new◦
- `__STDC__` CC◦ ◦

C ++ 11

- `__STDC_VERSION__` C __cplusplusC ++◦ ◦
- `__STDC_MB_MIGHT_NEQ_WC__` 1 if (uintmax_t)'x' != (uintmax_t)L'x'
- `wchar_t` Unicode __STDC_ISO_10646__ yyyymmLUnicode◦
- `__STDCPP_STRICT_POINTER_SAFETY__` 1
- `__STDCPP_THREADS__` 1 -

`__func__`◦ ◦

◦ ◦

- GCC
- Microsoft Visual C ++
-
- C ++

```
#ifndef __cplusplus // if compiled by C++ compiler
extern "C"{ // C code has to be decorated
    // C library header declarations here
}
#endif
```

C ++ 11

```
bool success = doSomething( /*some arguments*/ );
if( !success ){
    std::cerr << "ERROR: doSomething() failed on line " << __LINE__ - 2
                << " in function " << __func__ << "()"
                << " in file " << __FILE__
                << std::endl;
}
```

```
int main( int argc, char *argv[] ){
    if( argc == 2 && std::string( argv[1] ) == "-v" ){
        std::cout << "Hello World program\n"
                  << "v 1.1\n" // I have to remember to update this manually
                  << "compiled: " << __DATE__ << ' ' << __TIME__ // this updates automagically
                  << std::endl;
    }
    else{
        std::cout << "Hello World!\n";
    }
}
```

X-

◦

X-macro◦

```
#define LIST \
    X(dog) \
    X(cat) \
    X(racoon)

// class Animal {
//     public:
//     void say();
// };

#define X(name) Animal name;
LIST
#undef X
```

```
int main() {
#define X(name) name.say();
    LIST
#undef X

    return 0;
}
```

```
Animal dog;
Animal cat;
Animal racoon;

int main() {
    dog.say();
    cat.say();
    racoon.say();

    return 0;
}
```

100°

https://en.wikipedia.org/wiki/X_Macro

X-macros

LISTseaminglyx

```
#define LIST(MACRO) \
    MACRO(dog) \
    MACRO(cat) \
    MACRO(racoon)
```

```
#define FORWARD_DECLARE_ANIMAL(name) Animal name;
LIST(FORWARD_DECLARE_ANIMAL)
```

MACRO -

```
//a walkaround for Visual studio
#define EXPAND(x) x

#define LIST(MACRO, ...) \
    EXPAND(MACRO(dog, __VA_ARGS__)) \
    EXPAND(MACRO(cat, __VA_ARGS__)) \
    EXPAND(MACRO(racoon, __VA_ARGS__))
```

LIST LIST°

```
#define FORWARD_DECLARE(name, type, prefix) type prefix##name;
LIST(FORWARD_DECLARE, Animal, anim_)
LIST(FORWARD_DECLARE, Object, obj_)
```

```
Animal anim_dog;
Animal anim_cat;
Animal anim_racoon;
Object obj_dog;
Object obj_cat;
Object obj_racoon;
```

#pragma

C++ #pragma once ◦ ISO C++ ◦

```
// Foo.h
#pragma once

class Foo
{
};
```

#pragma once #pragma - - ◦ #pragma once ◦

- - #pragma once ◦ ◦ #pragma once include guard ◦

WindowsMFC #pragma once include guards Visual Studio add class add dialog add windows ◦ C++ Windows ◦

◦ ◦

```
// preprocessor will convert the parameter x to the string literal x
#define PRINT(x) printf(#x "\n")

PRINT(This line will be converted to string by preprocessor);
// Compiler sees
printf("This line will be converted to string by preprocessor""\n");
```

printf() ◦

◦ print ◦

```
PRINT( This line will be converted to string by preprocessor );
```

◦

```
PRINT(This "line" will be converted to "string" by preprocessor);
// Compiler sees
printf("This \"line\" will be converted to \"string\" by preprocessor""\n");
```

operator Token pasting ◦

```
// preprocessor will combine the variable and the x
#define PRINT(x) printf("variable" #x " = %d", variable##x)
```

```
int variableY = 15;
PRINT(Y);
//compiler sees
printf("variable""Y"" = %d", variableY);
```

```
variableY = 15
```

<https://riptutorial.com/zh-TW/cplusplus/topic/1098/>

140:

Examples

{ ... }° °

```
{
    int x = 100;
    //   ^
    //   Scope of `x` begins here
    //
} // <- Scope of `x` ends here
```

°

```
{
    int x = 100;

    {
        int x = 200;

        std::cout << x; // <- Output is 200
    }

    std::cout << x; // <- Output is 100
}
```

extern° /°

```
// File my_globals.h:

#ifndef __MY_GLOBALS_H__
#define __MY_GLOBALS_H__

extern int circle_radius; // Promise to the compiler that circle_radius
                          // will be defined somewhere

#endif
```

```
// File fool.cpp:

#include "my_globals.h"

int circle_radius = 123; // Defining the extern variable
```

```
// File main.cpp:

#include "my_globals.h"
#include <iostream>

int main()
{
```



```
std::cout << "The radius is: " << circle_radius << "\n";  
return 0;  
}
```

The radius is: 123

<https://riptutorial.com/zh-TW/cplusplus/topic/3453/>

141:

C ++CC ++C。 GNU C

C。 C“#include”。

。

- 。
- 。

C。

/。

C ++/。

C ++C ++ 20。

Examples

// filename。

```
// my_function.h

/* Note how this header contains only a declaration of a function.
 * Header functions usually do not define implementations for declarations
 * unless code must be further processed at compile time, as in templates.
 */

/* Also, usually header files include preprocessor guards so that every header
 * is never included twice.
 *
 * The guard is implemented by checking if a header-file unique preprocessor
 * token is defined, and only including the header if it hasn't been included
 * once before.
 */
#ifndef MY_FUNCTION_H
#define MY_FUNCTION_H

// global_value and my_function() will be
// recognized as the same constructs if this header is included by different files.
const int global_value = 42;
int my_function();

#endif // MY_FUNCTION_H
```

```
// my_function.cpp
```

```

/* Note how the corresponding source file for the header includes the interface
 * defined in the header so that the compiler is aware of what the source file is
 * implementing.
 *
 * In this case, the source file requires knowledge of the global constant
 * global_value only defined in my_function.h. Without inclusion of the header
 * file, this source file would not compile.
 */
#include "my_function.h" // or #include "my_function.hpp"
int my_function() {
    return global_value; // return 42;
}

```

◦ my_function.h

```

// main.cpp

#include <iostream> // A C++ Standard Library header.
#include "my_function.h" // A personal header

int main(int argc, char** argv) {
    std::cout << my_function() << std::endl;
    return 0;
}

```

/◦

```

g++ -c my_function.cpp # Compiles the source file my_function.cpp
                        # --> object file my_function.o

g++ main.cpp my_function.o # Links the object file containing the
                           # implementation of int my_function()
                           # to the compiled, object version of main.cpp
                           # and then produces the final executable a.out

```

main.cpp

```

g++ -c my_function.cpp
g++ -c main.cpp

g++ main.o my_function.o

```

◦

◦

◦

```

// templated_function.h

template <typename T>

```

```
T* null_T_pointer() {  
    T* type_point = NULL; // or, alternatively, nullptr instead of NULL  
                          // for C++11 or later  
    return type_point;  
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7211/>

142: /

- variable.member_var = constant;
- variable.member_function;
- variable_pointer-> member_var = constant;
- variable_pointer-> member_function;

struct class struct public class private ◦ C++ ◦ C++ ◦

Examples

◦ class struct union class ◦ "" ◦

- ""
- ""
- typedef ""
-

class struct class class "" struct union class "public" ◦

```
struct Vector
{
    int x;
    int y;
    int z;
};
// are equivalent to
class Vector
{
public:
    int x;
    int y;
    int z;
};
```

```
Vector my_vector;
```

◦

```
my_vector.x = 10;
my_vector.y = my_vector.x + 1; // my_vector.y = 11;
my_vector.z = my_vector.y - 4; // my_vector.z = 7;
```

◦

public	
protected	

```
private
```

```
classprivate structpublic
```

```
struct MyStruct { int x; };
class MyClass { int x; };

MyStruct s;
s.x = 9; // well formed, because x is public

MyClass c;
c.x = 9; // ill-formed, because x is private
```

gettersetter

```
class MyClass {

public: /* Methods: */

    int x() const noexcept { return m_x; }
    void setX(int const x) noexcept { m_x = x; }

private: /* Fields: */

    int m_x;

};
```

```
protectedcalculateValue()Plus2BasePlus2Base FortyTwo
```

```
struct Plus2Base {
    int value() noexcept { return calculateValue() + 2; }
protected: /* Methods: */
    virtual int calculateValue() noexcept = 0;
};
struct FortyTwo: Plus2Base {
protected: /* Methods: */
    int calculateValue() noexcept final override { return 40; }
};
```

```
friend
```

```
public protectedprivate
```

```
/
```

```
/B/A BA B/A A/
```

```
struct A
{
public:
    int p1;
protected:
    int p2;
```

```
private:
    int p3;
};

//Make B inherit publicly (default) from A
struct B : A
{
};
```

/3

- public
- private
- protected

publicstructprivateclass°

classstruct ° structclassclassstruct°

public

```
struct B : public A // or just `struct B : A`
{
    void foo()
    {
        p1 = 0; //well formed, p1 is public in B
        p2 = 0; //well formed, p2 is protected in B
        p3 = 0; //ill formed, p3 is private in A
    }
};

B b;
b.p1 = 1; //well formed, p1 is public
b.p2 = 1; //ill formed, p2 is protected
b.p3 = 1; //ill formed, p3 is inaccessible
```

private

```
struct B : private A
{
    void foo()
    {
        p1 = 0; //well formed, p1 is private in B
        p2 = 0; //well formed, p2 is private in B
        p3 = 0; //ill formed, p3 is private in A
    }
};

B b;
b.p1 = 1; //ill formed, p1 is private
b.p2 = 1; //ill formed, p2 is private
b.p3 = 1; //ill formed, p3 is inaccessible
```

protected

```
struct B : protected A
{
```

```

void foo()
{
    p1 = 0; //well formed, p1 is protected in B
    p2 = 0; //well formed, p2 is protected in B
    p3 = 0; //ill formed, p3 is private in A
}
};

B b;
b.p1 = 1; //ill formed, p1 is protected
b.p2 = 1; //ill formed, p2 is protected
b.p3 = 1; //ill formed, p3 is inaccessible

```

protected° protected“”°

OOP“IS-A”° °

[Liskov](#) /°

“”“HAS-A”° StackVector° °

°

virtual

```

struct A{};
struct B: public virtual A{};

```

BA A

```

struct A
{
    int member;
    A(int param)
    {
        member = param;
    }
};

struct B: virtual A
{
    B(): A(5){}
};

struct C: B
{
    C(): /*A(88)*/ {}
};

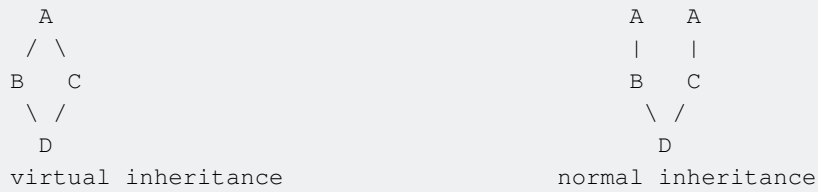
void f()
{
    C object; //error since C is not initializing it's indirect virtual base `A`
}

```

/*A(88)*/CA

object C C A A::member88 5 B°

°



BCA DBC **D2A**AD BC °

DA°

```
struct A
{
    void foo() {}
};

struct B : public /*virtual*/ A {};
struct C : public /*virtual*/ A {};

struct D : public B, public C
{
    void bar()
    {
        foo(); //Error, which foo? B::foo() or C::foo()? - Ambiguous
    }
};
```

°

```
class A {};
class B : public A {};
```

```
class A {};
class B {};
class C : public A, public B {};
```

CAB°

classstruct°

°

° °

```
class base1
{
public:
    void funtion( )
    { //code for base1 function }
```

```

};
class base2
{
    void function( )
        { // code for base2 function }
};

class derived : public base1, public base2
{
};

int main()
{
    derived obj;

    // Error because compiler can't figure out which function to call
    //either function( ) of base1 or base2 .
    obj.function( )
}

```

base1base2

```

int main()
{
    obj.base1::function( ); // Function of class base1 is called.
    obj.base2::function( ); // Function of class base2 is called.
}

```

```

struct SomeStruct {
    int a;
    int b;
    void foo() {}
};

SomeStruct var;
// Accessing member variable a in var.
std::cout << var.a << std::endl;
// Assigning member variable b in var.
var.b = 1;
// Calling a member function.
var.foo();

```

-> .

```

struct SomeStruct {
    int a;
    int b;
    void foo() {}
};

SomeStruct var;
SomeStruct *p = &var;
// Accessing member variable a in var via pointer.
std::cout << p->a << std::endl;

```

```

std::cout << (*p).a << std::endl;
// Assigning member variable b in var via pointer.
p->b = 1;
(*p).b = 1;
// Calling a member function via a pointer.
p->foo();
(*p).foo();

```

::° .->°

```

struct SomeStruct {
    int a;
    int b;
    void foo() {}

    static int c;
    static void bar() {}
};
int SomeStruct::c;

SomeStruct var;
SomeStruct* p = &var;
// Assigning static member variable c in struct SomeStruct.
SomeStruct::c = 5;
// Accessing static member variable c in struct SomeStruct, through var and p.
var.a = var.c;
var.b = p->c;
// Calling a static member function.
SomeStruct::bar();
var.bar();
p->bar();

```

->.*°

*pap pa° pa° *pa*(pa)° pa° a°

(*p).a

->° ° (*p).ap->a°

::° ° .->°;

```

class A {
public:
    int move();
    int turn();
};

class B : private A {
public:
    using A::turn;
};

B b;
b.move(); // compile error
b.turn(); // OK

```

AA

```
B b;  
A& a = static_cast<A&>(b); // compile error
```

AB

```
class B : public A {  
private:  
    int move();  
};
```

```
class B : public A {  
private:  
    using A::move;  
};
```

```
B b;  
A& a = static_cast<A&>(b); // OK for public inheritance  
a.move(); // OK
```

C++ 11

final ◦

```
class A final {  
};
```

```
// Compilation error: cannot derive from final class:  
class B : public A {  
};
```

```
class A {  
};  
  
// OK.  
class B final : public A {  
};  
  
// Compilation error: cannot derive from final class B.  
class C : public B {  
};
```

friend ◦

```
class Animal{  
private:  
    double weight;  
    double height;  
public:  
    friend void printWeight (Animal animal);  
    friend class AnimalPrinter;
```

```

// A common use for a friend function is to overload the operator<< for streaming.
friend std::ostream& operator<<(std::ostream& os, Animal animal);
};

void printWeight(Animal animal)
{
    std::cout << animal.weight << "\n";
}

class AnimalPrinter
{
public:
    void print(const Animal& animal)
    {
        // Because of the `friend class AnimalPrinter;" declaration, we are
        // allowed to access private members here.
        std::cout << animal.weight << ", " << animal.height << std::endl;
    }
}

std::ostream& operator<<(std::ostream& os, Animal animal)
{
    os << "Animal height: " << animal.height << "\n";
    return os;
}

int main() {
    Animal animal = {10, 5};
    printWeight(animal);

    AnimalPrinter aPrinter;
    aPrinter.print(animal);

    std::cout << animal;
}

```

```

10
10, 5
Animal height: 5

```

/

classstructclass / struct“”“”。

```

struct Outer {
    struct Inner { };
};

```

。

```

struct Outer {
    struct Inner { };

    Inner in;
};

```

```
// ...  
  
Outer o;  
Outer::Inner i = o.in;
```

class / struct ° °

```
// Bad.  
struct Outer {  
    struct Inner {  
        void do_something();  
    };  
  
    void Inner::do_something() {}  
};  
  
// Good.  
struct Outer {  
    struct Inner {  
        void do_something();  
    };  
  
};  
  
void Outer::Inner::do_something() {}
```

°

```
class Outer {  
    class Inner1;  
    class Inner2;  
  
    class Inner1 {};  
  
    Inner1 in1;  
    Inner2* in2p;  
  
public:  
    Outer();  
    ~Outer();  
};  
  
class Outer::Inner2 {};  
  
Outer::Outer() : in1(Inner1()), in2p(new Inner2) {}  
Outer::~~Outer() {  
    if (in2p) { delete in2p; }  
}
```

C++ 11

C++ 11 static;°

C++ 11

C++11 friend;

```
class Outer {
    struct Inner {
        int get_sizeof_x() {
            return sizeof(x); // Legal (C++11): x is unevaluated, so no instance is required.
        }

        int get_x() {
            return x; // Illegal: Can't access non-static member without an instance.
        }

        int get_x(Outer& o) {
            return o.x; // Legal (C++11): As a member of Outer, Inner can access private
members.
        }
    };

    int x;
};
```

o

```
class Outer {
    class Inner {
        // friend class Outer;

        int x;
    };

    Inner in;

public:
    int get_x() {
        return in.x; // Error: int Outer::Inner::x is private.
        // Uncomment "friend" line above to fix.
    }
};
```

;o o

```
class Outer {
    friend void barge_out(Outer& out, Inner& in);

    class Inner {
        friend void barge_in(Outer& out, Inner& in);

        int i;
    };

    int o;
};

void barge_in(Outer& out, Outer::Inner& in) {
    int i = in.i; // Good.
    int o = out.o; // Error: int Outer::o is private.
}
```

```
void barge_out(Outer& out, Outer::Inner& in) {
    int i = in.i; // Error: int Outer::Inner::i is private.
    int o = out.o; // Good.
}
```

◦ ◦

```
class Outer {
    struct Inner {
        void func() { std::cout << "I have no private taboo.\n"; }
    };

    public:
        static Inner make_Inner() { return Inner(); }
};

// ...

Outer::Inner oi; // Error: Outer::Inner is private.

auto oi = Outer::make_Inner(); // Good.
oi.func(); // Good.
Outer::make_Inner().func(); // Good.
```

◦ ◦ typedef◦

```
class Outer {
    class Inner_ {};

    public:
        typedef Inner_ Inner;
};

typedef Outer::Inner ImOut; // Good.
typedef Outer::Inner_ ImBad; // Error.

// ...

Outer::Inner oi; // Good.
Outer::Inner_ oi; // Error.
ImOut oi; // Good.
```

◦

```
struct Base {};

struct Outer {
    struct Inner : Base {};
};

struct Derived : Outer::Inner {};
```

◦ typedef


```

class BaseOuter {
    struct BaseInner_ {
        virtual void do_something() {}
        virtual void do_something_else();
    } b_in;

public:
    typedef BaseInner_ Inner;

    virtual ~BaseOuter() = default;

    virtual Inner& getInner() { return b_in; }
};

void BaseOuter::BaseInner_::do_something_else() {}

// ---

class DerivedOuter : public BaseOuter {
    // Note the use of the qualified typedef; BaseOuter::BaseInner_ is private.
    struct DerivedInner_ : BaseOuter::Inner {
        void do_something() override {}
        void do_something_else() override;
    } d_in;

public:
    typedef DerivedInner_ Inner;

    BaseOuter::Inner& getInner() override { return d_in; }
};

void DerivedOuter::DerivedInner_::do_something_else() {}

// ...

// Calls BaseOuter::BaseInner_::do_something();
BaseOuter* b = new BaseOuter;
BaseOuter::Inner& bin = b->getInner();
bin.do_something();
b->getInner().do_something();

// Calls DerivedOuter::DerivedInner_::do_something();
BaseOuter* d = new DerivedOuter;
BaseOuter::Inner& din = d->getInner();
din.do_something();
d->getInner().do_something();

```

BaseOuterDerivedOuterInner BaseInner_DerivedInner_ ◦ ◦

classstruct◦

```

struct IHaveATypedef {
    typedef int MyTypedef;
};

struct IHaveATemplateTypedef {
    template<typename T>
    using MyTemplateTypedef = std::vector<T>;
};

```

::typedef

```
IAHaveATypedef::MyTypedef i = 5; // i is an int.

IAHaveATemplateTypedef::MyTemplateTypedef<int> v; // v is a std::vector<int>.
```

◦ typedeftypedef

```
template<typename T>
struct Helper {
    T get() const { return static_cast<T>(42); }
};

struct IHaveTypedefs {
    // typedef MyTypedef NonLinearTypedef; // Error if uncommented.
    typedef int MyTypedef;
    typedef Helper<MyTypedef> MyTypedefHelper;
};

IHaveTypedefs::MyTypedef i; // x_i is an int.
IHaveTypedefs::MyTypedefHelper hi; // x_hi is a Helper<int>.

typedef IHaveTypedefs::MyTypedef TypedefBeFree;
TypedefBeFree ii; // ii is an int.
```

◦

```
class TypedefAccessLevels {
    typedef int PrvInt;

protected:
    typedef int ProInt;

public:
    typedef int PubInt;
};

TypedefAccessLevels::PrvInt prv_i; // Error: TypedefAccessLevels::PrvInt is private.
TypedefAccessLevels::ProInt pro_i; // Error: TypedefAccessLevels::ProInt is protected.
TypedefAccessLevels::PubInt pub_i; // Good.

class Derived : public TypedefAccessLevels {
    PrvInt prv_i; // Error: TypedefAccessLevels::PrvInt is private.
    ProInt pro_i; // Good.
    PubInt pub_i; // Good.
};
```

◦

```
class Something {
    friend class SomeComplexType;

    short s;
    // ...
};
```

```

public:
    typedef SomeComplexType MyHelper;

    MyHelper get_helper() const { return MyHelper(8, s, 19.5, "shoe", false); }

    // ...
};

// ...

Something s;
Something::MyHelper hlp = s.get_helper();

```

SomeComplexType typedef friend; Something::MyHelper ° °

decltype °

```

class SomethingElse {
    AnotherComplexType<bool, int, SomeThirdClass> helper;

public:
    typedef decltype(helper) MyHelper;

private:
    InternalVariable<MyHelper> ivh;

    // ...

public:
    MyHelper& get_helper() const { return helper; }

    // ...
};

```

decltype SomethingElse::helper typedef ° helper °

° typename ° typedef ° °

°

```

template<typename T>
class SomeClass {
    // ...

public:
    typedef T MyParam;
    MyParam getParam() { return static_cast<T>(42); }
};

template<typename T>
typename T::MyParam some_func(T& t) {
    return t.getParam();
}

SomeClass<int> si;
int i = some_func(si);

```

◦ C++12

```
template<typename T>
class SomeContainer {
    // ...

public:
    // Let's provide the same helper types as most standard containers.
    typedef T value_type;
    typedef std::allocator<value_type> allocator_type;
    typedef value_type& reference;
    typedef const value_type& const_reference;
    typedef value_type* pointer;
    typedef const value_type* const_pointer;
    typedef MyIterator<value_type> iterator;
    typedef MyConstIterator<value_type> const_iterator;
    typedef std::reverse_iterator<iterator> reverse_iterator;
    typedef std::reverse_iterator<const_iterator> const_reverse_iterator;
    typedef size_t size_type;
    typedef ptrdiff_t difference_type;
};
```

C++ 11 typedef ""; ◦ type ◦

```
template<typename T>
struct TemplateTypedef {
    typedef T type;
}

TemplateTypedef<int>::type i; // i is an int.
```

◦

```
template<typename T, size_t SZ, size_t D>
class Array { /* ... */ };

template<typename T, size_t SZ>
struct OneDArray {
    typedef Array<T, SZ, 1> type;
};

template<typename T, size_t SZ>
struct TwoDArray {
    typedef Array<T, SZ, 2> type;
};

template<typename T>
struct MonoDisplayLine {
    typedef Array<T, 80, 1> type;
};

OneDArray<int, 3>::type arr1i; // arr1i is an Array<int, 3, 1>.
TwoDArray<short, 5>::type arr2s; // arr2s is an Array<short, 5, 2>.
MonoDisplayLine<char>::type arr3c; // arr3c is an Array<char, 80, 1>.
```

static ◦ ; ◦

```

class Example {
    static int num_instances;    // Static data member (static member variable).
    int i;                      // Non-static member variable.

public:
    static std::string static_str; // Static data member (static member variable).
    static int static_func();      // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int         Example::num_instances;
std::string Example::static_str = "Hello.";

// ...

Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)

```

;
◦ static ◦

```

class Example {
    static int num_instances;    // Declaration.

public:
    static std::string static_str; // Declaration.

    // ...
};

int         Example::num_instances;    // Definition. Zero-initialised.
std::string Example::static_str = "Hello."; // Definition.

```

void ◦

```

struct ForwardDeclared;

class ExIncomplete {
    static ForwardDeclared fd;
    static ExIncomplete   i_contain_myself;
    static int            an_array[];
};

struct ForwardDeclared {};

ForwardDeclared ExIncomplete::fd;
ExIncomplete   ExIncomplete::i_contain_myself;
int            ExIncomplete::an_array[5];

```

◦ `static` ◦

```
// For Example above, either...
class Example {
    // ...

public:
    static int static_func() { return num_instances; }

    // ...

    void set_str(const std::string& str) { static_str = str; }
};

// Or...

class Example { /* ... */ };

int Example::static_func() { return num_instances; }
void Example::set_str(const std::string& str) { static_str = str; }
```

`constvolatile` ◦

```
enum E { VAL = 5 };

struct ExConst {
    const static int ci = 5;           // Good.
    static const E ce = VAL;          // Good.
    const static double cd = 5;       // Error.
    static const volatile int cvi = 5; // Error.

    const static double good_cd;
    static const volatile int good_cvi;
};

const double ExConst::good_cd = 5;    // Good.
const volatile int ExConst::good_cvi = 5; // Good.
```

C++ 11

C++ 11 `LiteralTypeconstexprconstexpr`;◦

```
struct ExConstexpr {
    constexpr static int ci = 5;           // Good.
    static constexpr double cd = 5;       // Good.
    constexpr static int carr[] = { 1, 1, 2 }; // Good.
    static constexpr ConstexprConstructibleClass c{}; // Good.
    constexpr static int bad_ci;          // Error.
};

constexpr int ExConstexpr::bad_ci = 5;    // Still an error.
```

`constconstexpr` *odr* ◦ ◦

```
struct ExODR {
```

```
    static const int odr_used = 5;
};

// const int ExODR::odr_used;

const int* odr_user = & ExODR::odr_used; // Error; uncomment above line to resolve.
```

::°

```
std::string str = Example::static_str;
```

° °

```
Example ex;
std::string rts = ex.static_str;
```

°

```
class ExTwo {
    static int num_instances;
    int my_num;

public:
    ExTwo() : my_num(num_instances++) {}

    static int get_total_instances() { return num_instances; }
    int get_instance_number() const { return my_num; }
};

int ExTwo::num_instances;
```

mutable ;const°

```
struct ExDontNeedMutable {
    int immuta;
    mutable int muta;

    static int i;

    ExDontNeedMutable() : immuta(-5), muta(-5) {}
};
int ExDontNeedMutable::i;

// ...

const ExDontNeedMutable dnm;
dnm.immuta = 5; // Error: Can't modify read-only object.
dnm.muta = 5; // Good. Mutable fields of const objects can be written.
dnm.i = 5; // Good. Static members can be written regardless of an instance's const-
ness.
```

°

```

class ExAccess {
    static int prv_int;

protected:
    static int pro_int;

public:
    static int pub_int;
};

int ExAccess::prv_int;
int ExAccess::pro_int;
int ExAccess::pub_int;

// ...

int x1 = ExAccess::prv_int; // Error: int ExAccess::prv_int is private.
int x2 = ExAccess::pro_int; // Error: int ExAccess::pro_int is protected.
int x3 = ExAccess::pub_int; // Good.

```

this;°

```

class ExInstanceRequired {
    int i;

public:
    ExInstanceRequired() : i(0) {}

    static void bad_mutate() { ++i *= 5; } // Error.
    static void good_mutate(ExInstanceRequired& e) { ++e.i *= 5; } // Good.
};

```

this°

```

struct ExPointer {
    void nsfunc() {}
    static void sfunc() {}
};

typedef void (ExPointer::* mem_f_ptr)();
typedef void (*f_ptr)();

mem_f_ptr p_sf = &ExPointer::sfunc; // Error.
f_ptr p_sf = &ExPointer::sfunc; // Good.

```

thisconstvolatile **ref-qualifiers**° °

```

struct ExCVQualifiersAndVirtual {
    static void func() {} // Good.
    static void cfunc() const {} // Error.
    static void vfunc() volatile {} // Error.
    static void cvfunc() const volatile {} // Error.
    static void rfunc() & {} // Error.
    static void rvfunc() && {} // Error.

    virtual static void vsfunc() {} // Error.
};

```



```
static virtual void svfunc()      {} // Error.
};
```

◦ thread_local C++ 11

◦ ◦

◦

```
class CL {
public:
    void member_function() {}
};
```

```
CL instance;
instance.member_function();
```

◦

```
struct ST {
    void defined_inside() {}
    void defined_outside();
};
void ST::defined_outside() {}
```

CV/ref-qualified ;cv-qualifier ◦ cv ◦ cvcv ◦

```
struct CVQualifiers {
    void func()                {} // 1: Instance is non-cv-qualified.
    void func() const          {} // 2: Instance is const.

    void cv_only() const volatile {}
};

CVQualifiers      non_cv_instance;
const CVQualifiers c_instance;

non_cv_instance.func(); // Calls #1.
c_instance.func();      // Calls #2.

non_cv_instance.cv_only(); // Calls const volatile version.
c_instance.cv_only();      // Calls const volatile version.
```

C++ 11

ref-qualifiers rvalue cv-qualifiers ◦

```
struct RefQualifiers {
    void func() & {} // 1: Called on normal instances.
    void func() && {} // 2: Called on rvalue (temporary) instances.
};
```

```

RefQualifiers rf;
rf.func();           // Calls #1.
RefQualifiers{}.func(); // Calls #2.

```

CVref

```

struct BothCVAndRef {
    void func() const& {} // Called on normal instances. Sees instance as const.
    void func() &&      {} // Called on temporary instances.
};

```

;

```

struct Base {
    virtual void func() {}
};
struct Derived {
    virtual void func() {}
};

Base* bp = new Base;
Base* dp = new Derived;
bp.func(); // Calls Base::func().
dp.func(); // Calls Derived::func().

```

o

/

struct

```

void foo()
{
    struct /* No name */ {
        float x;
        float y;
    } point;

    point.x = 42;
}

```

```

struct Circle
{
    struct /* No name */ {
        float x;
        float y;
    } center; // but a member name
    float radius;
};

```

```

Circle circle;
circle.center.x = 42.f;

```

struct

```
struct InvalidCircle
{
    struct /* No name */ {
        float centerX;
        float centerY;
    }; // No member either.
    float radius;
};
```

struct °

C++ 11

- *lambda* *struct* °
- *decltype* *struct*

```
decltype(circle.point) otherPoint;
```

- *struct*

```
void print_square_coordinates()
{
    const struct {float x; float y;} points[] = {
        {-1, -1}, {-1, 1}, {1, -1}, {1, 1}
    };

    // for range relies on `template <class T, std::size_t N> std::begin(T (&)[N])`
    for (const auto& point : points) {
        std::cout << "{" << point.x << ", " << point.y << "}\n";
    }

    decltype(points[0]) topRightCorner{1, 1};
    auto it = std::find(points, points + 4, topRightCorner);
    std::cout << "top right corner is the "
        << 1 + std::distance(points, it) << "th\n";
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/508/>

143:

201411C ++N3922。 C ++。

Examples

C ++ 17。 ◦ **lambdas** ◦ `make_pair()` ◦ `make_tuple()` ◦ `back_inserter()` ◦

C ++ 17

```
std::pair p(2, 4.5); // std::pair<int, double>
std::tuple t(4, 3, 2.5); // std::tuple<int, int, double>
std::copy_n(vil.begin(), 3,
            std::back_inserter(vi2)); // constructs a back_inserter_iterator<std::vector<int>>
std::lock_guard lk(mtx); // std::lock_guard<decltype(mtx)>
```

```
template <class Iter>
vector(Iter, Iter) -> vector<typename iterator_traits<Iter>::value_type>

int array[] = {1, 2, 3};
std::vector v(std::begin(array), std::end(array)); // deduces std::vector<int>
```

```
template<typename T>
void f(ParamType param);

f(expr);
```

1 ParamType ◦ ◦ expr ◦ exprParamTypeT

```
template<typename T>
void f(T& param); //param is a reference

int x = 27; // x is an int
const int cx = x; // cx is a const int
const int& rx = x; // rx is a reference to x as a const int

f(x); // T is int, param's type is int&
f(cx); // T is const int, param's type is const int&
f(rx); // T is const int, param's type is const int&
```

2 ParamType ◦ `expr rvalue1` ◦ `expr TParamType` ◦

```
template<typename T>
void f(T&& param); // param is a universal reference

int x = 27; // x is an int
const int cx = x; // cx is a const int
const int& rx = x; // rx is a reference to x as a const int

f(x); // x is lvalue, so T is int&, param's type is also int&
f(cx); // cx is lvalue, so T is const int&, param's type is also const int&
```

```
f(rx);           // rx is lvalue, so T is const int&, param's type is also const int&
f(27);          // 27 is rvalue, so T is int, param's type is therefore int&&
```

3 ParamType◦ expr◦ exprconst◦ volatileT◦

```
template<typename T>
void f(T param);           // param is now passed by value

int x = 27;                // x is an int
const int cx = x;         // cx is a const int
const int& rx = x;        // rx is a reference to x as a const int

f(x);                     // T's and param's types are both int
f(cx);                    // T's and param's types are again both int
f(rx);                    // T's and param's types are still both int
```

C++ 11

auto◦

```
auto x = 27;              // (x is neither a pointer nor a reference), x's type is int
const auto cx = x;       // (cx is neither a pointer nor a reference), cx's type is const int
const auto& rx = x;      // (rx is a non-universal reference), rx's type is a reference to a
                        // const int

auto&& uref1 = x;         // x is int and lvalue, so uref1's type is int&
auto&& uref2 = cx;       // cx is const int and lvalue, so uref2's type is const int &
auto&& uref3 = 27;       // 27 is an int and rvalue, so uref3's type is int&&
```

```
auto x1 = 27;            // type is int, value is 27
auto x2(27);             // type is int, value is 27
auto x3 = { 27 };       // type is std::initializer_list<int>, value is { 27 }
auto x4{ 27 };          // type is std::initializer_list<int>, value is { 27 }
                        // in some compilers type may be deduced as an int with a
                        // value of 27. See remarks for more information.
auto x5 = { 1, 2.0 }    // error! can't deduce T for std::initializer_list<t>
```

auto std::initializer_list<T>◦ T◦

auto◦

```
auto f() -> int {
    return 42;
}
```

C++ 14

C++ 11 C++ 14

1. return◦

```
// f returns int:
auto f() { return 42; }
```

```
// g returns void:
auto g() { std::cout << "hello, world!\n"; }
```

2. lambda lambda lambda

```
auto triple = [](auto x) { return 3*x; };
const auto x = triple(42); // x is a const int with value 126
```

decltype(auto) decltype(auto)

```
int* p = new int(42);
auto x = *p; // x has type int
decltype(auto) y = *p; // y is a reference to *p
```

C++ 03 auto C

<https://riptutorial.com/zh-TW/cplusplus/topic/7863/>

144:

C++ 11 auto

auto const &constexpr • auto STL

Examples

•

```
auto a = 1;           // a = int
auto b = 2u;         // b = unsigned int
auto c = &a;         // c = int*
const auto d = c;    // d = const int*
const auto& e = b;   // e = const unsigned int&

auto x = a + b       // x = int, #compiler warning unsigned and signed

auto v = std::vector<int>; // v = std::vector<int>
```

&constconstexpr auto

```
// y = unsigned int,
// note that y does not infer as const unsigned int&
// The compiler would have generated a copy instead of a reference value to e or b
auto y = e;
```

Lambda

auto lambda •

```
auto DoThis = [](int a, int b) { return a + b; };
// Do this is of type (int)(*DoThis)(int, int)
// else we would have to write this long
int(*pDoThis)(int, int) = [](int a, int b) { return a + b; };

auto c = Dothis(1, 2); // c = int
auto d = pDothis(1, 2); // d = int

// using 'auto' shortens the definition for lambda functions
```

lambda •

```
[](int a, int b) -> int { return a + b; };
[](int a, int b) -> auto { return a + b; };
[](int a, int b) { return a + b; };
```

auto for

```
std::map<int, std::string> Map;
for (auto pair : Map)           // pair = std::pair<int, std::string>
for (const auto pair : Map)     // pair = const std::pair<int, std::string>
for (const auto& pair : Map)    // pair = const std::pair<int, std::string>&
for (auto i = 0; i < 1000; ++i) // i = int
for (auto i = 0; i < Map.size(); ++i) // Note that i = int and not size_t
for (auto i = Map.size(); i > 0; --i) // i = size_t
```

<https://riptutorial.com/zh-TW/cplusplus/topic/8233/>

145:

- `std::function<R(A...)>` C++

Examples

- ;;

◦

```
#include <ostream>

class Printable
{
public:
    template <typename T>
    Printable(T value) : pValue(new Value<T>(value)) {}
    ~Printable() { delete pValue; }
    void print(std::ostream &os) const { pValue->print(os); }

private:
    Printable(Printable const &) /* in C++1x: =delete */; // not implemented
    void operator = (Printable const &) /* in C++1x: =delete */; // not implemented
    struct ValueBase
    {
        virtual ~ValueBase() = default;
        virtual void print(std::ostream &) const = 0;
    };
    template <typename T>
    struct Value : ValueBase
    {
        Value(T const &t) : v(t) {}
        virtual void print(std::ostream &os) const { os << v; }
        T v;
    };
    ValueBase *pValue;
};
```

◦

```
#include <iostream>

void print_value(Printable const &p)
{
    p.print(std::cout);
}
```

◦

Printable

```
struct MyType { int i; };
ostream& operator << (ostream &os, MyType const &mc)
```

```
{
    return os << "MyType {" << mc.i << "}";
}
```

```
MyType foo = { 42 };
print_value(foo);
```

vtable

C++RegularPseudo-Regular。

- - ° ° std°

C++ 11std::hash°

vtable°

```
using dtor_unique_ptr = std::unique_ptr<void, void*(void*)>;
template<class T, class...Args>
dtor_unique_ptr make_dtor_unique_ptr( Args&&... args ) {
    return {new T(std::forward<Args>(args)...), [](void* self){ delete static_cast<T*>(self);
}};
}
struct regular_vtable {
    void(*copy_assign)(void* dest, void const* src); // T&=(T const&)
    void(*move_assign)(void* dest, void* src); // T&=(T&&)
    bool(*equals)(void const* lhs, void const* rhs); // T const&==T const&
    bool(*order)(void const* lhs, void const* rhs); // std::less<T>{}(T const&, T const&)
    std::size_t(*hash)(void const* self); // std::hash<T>{}(T const&)
    std::type_info const&(*type)(); // typeid(T)
    dtor_unique_ptr(*clone)(void const* self); // T(T const&)
};
template<class T>
regular_vtable make_regular_vtable() noexcept {
    return {
        [](void* dest, void const* src){ *static_cast<T*>(dest) = *static_cast<T const*>(src); },
        [](void* dest, void* src){ *static_cast<T*>(dest) = std::move(*static_cast<T*>(src)); },
        [](void const* lhs, void const* rhs){ return *static_cast<T const*>(lhs) == *static_cast<T const*>(rhs); },
        [](void const* lhs, void const* rhs) { return std::less<T>{}(*static_cast<T const*>(lhs), *static_cast<T const*>(rhs)); },
        [](void const* self){ return std::hash<T>{}(*static_cast<T const*>(self)); },
        []()->decltype(auto){ return typeid(T); },
        [](void const* self){ return make_dtor_unique_ptr<T>(*static_cast<T const*>(self)); }
    };
}
template<class T>
regular_vtable const* get_regular_vtable() noexcept {
    static const regular_vtable vtable=make_regular_vtable<T>();
    return &vtable;
}
struct regular_type {
    using self=regular_type;
    regular_vtable const* vtable = 0;
    dtor_unique_ptr ptr{nullptr, [](void*){}};
```

```

bool empty() const { return !vtable; }

template<class T, class...Args>
void emplace( Args&&... args ) {
    ptr = make_dtorg_unique_ptr<T>(std::forward<Args>(args)...);
    if (ptr)
        vtable = get_regular_vtable<T>();
    else
        vtable = nullptr;
}
friend bool operator==(regular_type const& lhs, regular_type const& rhs) {
    if (lhs.vtable != rhs.vtable) return false;
    return lhs.vtable->equals( lhs.ptr.get(), rhs.ptr.get() );
}
bool before(regular_type const& rhs) const {
    auto const& lhs = *this;
    if (!lhs.vtable || !rhs.vtable)
        return std::less<regular_vtable const*>{}(lhs.vtable, rhs.vtable);
    if (lhs.vtable != rhs.vtable)
        return lhs.vtable->type().before(rhs.vtable->type());
    return lhs.vtable->order( lhs.ptr.get(), rhs.ptr.get() );
}
// technically friend bool operator< that calls before is also required

std::type_info const* type() const {
    if (!vtable) return nullptr;
    return &vtable->type();
}
regular_type(regular_type&& o):
    vtable(o.vtable),
    ptr(std::move(o.ptr))
{
    o.vtable = nullptr;
}
friend void swap(regular_type& lhs, regular_type& rhs){
    std::swap(lhs.ptr, rhs.ptr);
    std::swap(lhs.vtable, rhs.vtable);
}
regular_type& operator=(regular_type&& o) {
    if (o.vtable == vtable) {
        vtable->move_assign(ptr.get(), o.ptr.get());
        return *this;
    }
    auto tmp = std::move(o);
    swap(*this, tmp);
    return *this;
}
regular_type(regular_type const& o):
    vtable(o.vtable),
    ptr(o.vtable?o.vtable->clone(o.ptr.get()):dtorg_unique_ptr{nullptr, [] (void*){}})
{
    if (!ptr && vtable) vtable = nullptr;
}
regular_type& operator=(regular_type const& o) {
    if (o.vtable == vtable) {
        vtable->copy_assign(ptr.get(), o.ptr.get());
        return *this;
    }
    auto tmp = o;
    swap(*this, tmp);
}

```

```

    return *this;
}
std::size_t hash() const {
    if (!vtable) return 0;
    return vtable->hash(ptr.get());
}
template<class T,
        std::enable_if_t< !std::is_same<std::decay_t<T>, regular_type>{}, int>* =nullptr
>
regular_type(T&& t) {
    emplace<std::decay_t<T>>(std::forward<T>(t));
}
};
namespace std {
    template<>
    struct hash<regular_type> {
        std::size_t operator()( regular_type const& r )const {
            return r.hash();
        }
    };
    template<>
    struct less<regular_type> {
        bool operator()( regular_type const& lhs, regular_type const& rhs ) const {
            return lhs.before(rhs);
        }
    };
}
}

```

◦

std::mapstd::unordered_map

```
std::map<regular_type, std::any>
```

◦

anyregular_type◦ ◦

regular_typeptr◦

make_dtor_unique_ptr◦ ◦

`std :: function`

std::function◦ ◦

ptrslambda◦ std::function◦

std::packaged_task<Sig>◦ std::packaged_task<void(Args...)>std::packaged_task<R(Args...)>◦

task◦ std::function◦ details::task_pimpl<...>clone◦

```
template<class Sig>
struct task;
```

```

// putting it in a namespace allows us to specialize it nicely for void return value:
namespace details {
    template<class R, class...Args>
    struct task_pimpl {
        virtual R invoke(Args&&...args) const = 0;
        virtual ~task_pimpl() {};
        virtual const std::type_info& target_type() const = 0;
    };

    // store an F.  invoke(Args&&...) calls the f
    template<class F, class R, class...Args>
    struct task_pimpl_impl:task_pimpl<R,Args...> {
        F f;
        template<class Fin>
        task_pimpl_impl( Fin&& fin ):f(std::forward<Fin>(fin)) {}
        virtual R invoke(Args&&...args) const final override {
            return f(std::forward<Args>(args)...);
        }
        virtual const std::type_info& target_type() const final override {
            return typeid(F);
        }
    };

    // the void version discards the return value of f:
    template<class F, class...Args>
    struct task_pimpl_impl<F,void,Args...>:task_pimpl<void,Args...> {
        F f;
        template<class Fin>
        task_pimpl_impl( Fin&& fin ):f(std::forward<Fin>(fin)) {}
        virtual void invoke(Args&&...args) const final override {
            f(std::forward<Args>(args)...);
        }
        virtual const std::type_info& target_type() const final override {
            return typeid(F);
        }
    };
};

template<class R, class...Args>
struct task<R(Args...)> {
    // semi-regular:
    task()=default;
    task(task&&)=default;
    // no copy

private:
    // aliases to make some SFINAE code below less ugly:
    template<class F>
    using call_r = std::result_of_t<F const&(Args...)>;
    template<class F>
    using is_task = std::is_same<std::decay_t<F>, task>;
public:
    // can be constructed from a callable F
    template<class F,
        // that can be invoked with Args... and converted-to-R:
        class= decltype( (R) (std::declval<call_r<F>>()) ),
        // and is not this same type:
        std::enable_if_t<!is_task<F>{}, int>* = nullptr
    >
    task(F&& f):

```

```

    m_pImpl( make_pimpl(std::forward<F>(f)) )
{}

// the meat: the call operator
R operator()(Args... args) const {
    return m_pImpl->invoke( std::forward<Args>(args)... );
}
explicit operator bool() const {
    return (bool)m_pImpl;
}
void swap( task& o ) {
    std::swap( m_pImpl, o.m_pImpl );
}
template<class F>
void assign( F&& f ) {
    m_pImpl = make_pimpl(std::forward<F>(f));
}
// Part of the std::function interface:
const std::type_info& target_type() const {
    if (!*this) return typeid(void);
    return m_pImpl->target_type();
}
template< class T >
T* target() {
    return target_impl<T>();
}
template< class T >
const T* target() const {
    return target_impl<T>();
}
// compare with nullptr
friend bool operator==( std::nullptr_t, task const& self ) { return !self; }
friend bool operator==( task const& self, std::nullptr_t ) { return !self; }
friend bool operator!=( std::nullptr_t, task const& self ) { return !!self; }
friend bool operator!=( task const& self, std::nullptr_t ) { return !!self; }
private:
template<class T>
using pimpl_t = details::task_pimpl_impl<T, R, Args...>;

template<class F>
static auto make_pimpl( F&& f ) {
    using dF=std::decay_t<F>;
    using pImpl_t = pimpl_t<dF>;
    return std::make_unique<pImpl_t>(std::forward<F>(f));
}
std::unique_ptr<details::task_pimpl<R,Args...>> m_pImpl;

template< class T >
T* target_impl() const {
    return dynamic_cast<pimpl_t<T>*>(m_pImpl.get());
}
};

```

◦

SBO task(task&&) std::aligned_storage_t m_pImpl unique_ptr emplace_move_to(void*) = 0 task_pimpl
emplace_move_to(void*) = 0 ◦

SBO ◦

T

◦

◦ `array_view`.

`array_view`

```
// helper traits for SFINAE:
template<class T>
using data_t = decltype( std::declval<T>().data() );

template<class Src, class T>
using compatible_data = std::integral_constant<bool, std::is_same< data_t<Src>, T* >{} ||
std::is_same< data_t<Src>, std::remove_const_t<T>* >{}>;

template<class T>
struct array_view {
    // the core of the class:
    T* b=nullptr;
    T* e=nullptr;
    T* begin() const { return b; }
    T* end() const { return e; }

    // provide the expected methods of a good contiguous range:
    T* data() const { return begin(); }
    bool empty() const { return begin()==end(); }
    std::size_t size() const { return end()-begin(); }

    T& operator[](std::size_t i)const{ return begin()[i]; }
    T& front()const{ return *begin(); }
    T& back()const{ return *(end()-1); }

    // useful helpers that let you generate other ranges from this one
    // quickly and safely:
    array_view without_front( std::size_t i=1 ) const {
        i = (std::min)(i, size());
        return {begin()+i, end()};
    }
    array_view without_back( std::size_t i=1 ) const {
        i = (std::min)(i, size());
        return {begin(), end()-i};
    }
}

// array_view is plain old data, so default copy:
array_view(array_view const&)=default;
// generates a null, empty range:
array_view()=default;

// final constructor:
array_view(T* s, T* f):b(s),e(f) {}
// start and length is useful in my experience:
array_view(T* s, std::size_t length):array_view(s, s+length) {}

// SFINAE constructor that takes any .data() supporting container
// or other range in one fell swoop:
template<class Src,
    std::enable_if_t< compatible_data<std::remove_reference_t<Src>&, T >{}, int>* =nullptr,
    std::enable_if_t< !std::is_same<std::decay_t<Src>, array_view >{}, int>* =nullptr
```

```

>
array_view( Src&& src ):
    array_view( src.data(), src.size() )
{}

// array constructor:
template<std::size_t N>
array_view( T(&arr)[N] ):array_view(arr, N) {}

// initializer list, allowing {} based:
template<class U,
        std::enable_if_t< std::is_same<const U, T>{}, int>* =nullptr
>
array_view( std::initializer_list<U> il ):array_view(il.begin(), il.end()) {}
};

```

array_view.data()T.size()T S°

std::vector<T> std::string<T> std::array<T, N> a T[37] {} T* x.data()size_t x.size() °

“”。

°

ADLdatasize °

std :: any

C ++ 14boost::any ° C ++ 17std::any °

```

const auto print =
    make_any_method<void(std::ostream&)>([](auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print)(std::cout);

```

°

[@dyp@cplearner](#)°

```

template<class T>struct tag_t{constexpr tag_t(){};};
template<class T>constexpr tag_t<T> tag{};

```

traitany_method

any_method any_method

```

template<class any_method>
using any_sig_from_method = typename any_method::signature;

template<class any_method, class Sig=any_sig_from_method<any_method>>

```



```

struct any_method_function;

template<class any_method, class R, class...Args>
struct any_method_function<any_method, R(Args...)>
{
    template<class T>
    using decorate = std::conditional_t< any_method::is_const, T const, T >;

    using any = decorate<boost::any>;

    using type = R(*) (any&, any_method const*, Args&&...);
    template<class T>
    type operator()( tag_t<T> ) const {
        return +[] (any& self, any_method const* method, Args&&...args) {
            return (*method)( boost::any_cast<decorate<T>&>(self), decltype(args)(args)... );
        };
    }
};

```

any_method_function::type any_method_function::operator() tag_t<T> any_method_function::type any&T

。。

```

template<class...any_methods>
using any_method_tuple = std::tuple< typename any_method_function<any_methods>::type... >;

template<class...any_methods, class T>
any_method_tuple<any_methods...> make_vtable( tag_t<T> ) {
    return std::make_tuple(
        any_method_function<any_methods>{}(tag<T>)...
    );
}

template<class...methods>
struct any_methods {
private:
    any_method_tuple<methods...> const* vtable = 0;
    template<class T>
    static any_method_tuple<methods...> const* get_vtable( tag_t<T> ) {
        static const auto table = make_vtable<methods...>(tag<T>);
        return &table;
    }
public:
    any_methods() = default;
    template<class T>
    any_methods( tag_t<T> ): vtable( get_vtable(tag<T>) ) {}
    any_methods& operator=(any_methods const&)=default;
    template<class T>
    void change_type( tag_t<T> ={} ) { vtable = get_vtable(tag<T>); }

    template<class any_method>
    auto get_invoker( tag_t<any_method> ={} ) const {
        return std::get<typename any_method_function<any_method>::type>( *vtable );
    }
};

```

vtable1。

super_any

◦ `super_any_t` ◦ `super_any` ◦

```
template<class...methods>
struct super_any_t;
```

super any SFINAE

```
template<class super_any, class method>
struct super_method_applies_helper : std::false_type {};

template<class M0, class...Methods, class method>
struct super_method_applies_helper<super_any_t<M0, Methods...>, method> :
    std::integral_constant<bool, std::is_same<M0, method>{} ||
    super_method_applies_helper<super_any_t<Methods...>, method>{}>
{};

template<class...methods, class method>
auto super_method_test( super_any_t<methods...> const&, tag_t<method> )
{
    return std::integral_constant<bool, super_method_applies_helper< super_any_t<methods...>,
    method >{} && method::is_const >{};
}

template<class...methods, class method>
auto super_method_test( super_any_t<methods...>&, tag_t<method> )
{
    return std::integral_constant<bool, super_method_applies_helper< super_any_t<methods...>,
    method >{} >{};
}

template<class super_any, class method>
struct super_method_applies:
    decltype( super_method_test( std::declval<super_any>(), tag<method> ) )
{};
```

`any_method` ◦ `any_method` ◦ `const`

```
const auto print=make_any_method( [](auto&&self, auto&&os){ os << self; } );
```

C++ 17

```
const any_method print=[](auto&&self, auto&&os){ os << self; };
```

`lambda` ◦ ◦ `lambda` ◦ `lambda` ◦

```
template<class Sig, bool const_method, class F>
struct any_method {
    using signature=Sig;
    enum{is_const=const_method};
private:
    F f;
public:

    template<class Any,
        // SFINAE testing that one of the Anys's matches this type:
        std::enable_if_t< super_method_applies< Any&&, any_method >{}, int>* =nullptr
```

```

>
friend auto operator->*( Any&& self, any_method const& m ) {
    // we don't use the value of the any_method, because each any_method has
    // a unique type (!) and we check that one of the auto*'s in the super_any
    // already has a pointer to us. We then dispatch to the corresponding
    // any_method_data...

    return [&self, invoke = self.get_invoker(tag<any_method>), m](auto&&...args)-
>decltype(auto)
    {
        return invoke( decltype(self)(self), &m, decltype(args)(args)... );
    };
}
any_method( F fin ):f(std::move(fin)) {}

template<class...Args>
decltype(auto) operator() (Args&&...args) const {
    return f(std::forward<Args>(args)...);
}
};

```

C++17

```

template<class Sig, bool is_const=false, class F>
any_method<Sig, is_const, std::decay_t<F>>
make_any_method( F&& f ) {
    return {std::forward<F>(f)};
}

```

any ◦ any any

```

template<class... methods>
struct super_any_t:boost::any, any_methods<methods...> {
    using vtable=any_methods<methods...>;
public:
    template<class T,
        std::enable_if_t< !std::is_base_of<super_any_t, std::decay_t<T>>{}, int> =0
    >
    super_any_t( T&& t ):
        boost::any( std::forward<T>(t) )
    {
        using dT=std::decay_t<T>;
        this->change_type( tag<dT> );
    }

    boost::any& as_any()&{return *this;}
    boost::any&& as_any()&&{return std::move(*this);}
    boost::any const& as_any()const&{return *this;}
    super_any_t()=default;
    super_any_t(super_any_t&& o):
        boost::any( std::move( o.as_any() ) ),
        vtable(o)
    {}
    super_any_t(super_any_t const& o):
        boost::any( o.as_any() ),
        vtable(o)
    {}
    template<class S,
        std::enable_if_t< std::is_same<std::decay_t<S>, super_any_t>{}, int> =0

```

```

>
super_any_t( S&& o ):
    boost::any( std::forward<S>(o).as_any() ),
    vtable(o)
{}
super_any_t& operator=(super_any_t&&)=default;
super_any_t& operator=(super_any_t const&)=default;

template<class T,
    std::enable_if_t< !std::is_same<std::decay_t<T>, super_any_t>{}>, int>* =nullptr
>
super_any_t& operator=( T&& t ) {
    ((boost::any&)*this) = std::forward<T>(t);
    using dT=std::decay_t<T>;
    this->change_type( tag<dT> );
    return *this;
}
};

```

any_methodconstsuper_any

```

template<class...Ts>
using super_any = super_any_t< std::remove_cv_t<Ts>... >;

```

```

const auto print = make_any_method<void(std::ostream&)>([](auto&& p, std::ostream& t){ t << p
<< "\n"; });
const auto wprint = make_any_method<void(std::wostream&)>([](auto&& p, std::wostream& os ){ os
<< p << L"\n"; });

int main()
{
    super_any<decltype(print), decltype(wprint)> a = 7;
    super_any<decltype(print), decltype(wprint)> a2 = 7;

    (a->*print)(std::cout);
    (a->*wprint)(std::wcout);
}

```

o

SOo

<https://riptutorial.com/zh-TW/cplusplus/topic/2872/>

146:

◦ ◦ c++11 ◦ ◦

Examples

C++ 11

type_traits◦

◦

◦

```
template <class T> struct is_foo;
```

```
foois_foo<T>std::integral_constant<bool,true> std::true_type std::integral_constant<bool,false>
std::false_type ◦
```

```
static constexpr bool value
```

```
Tfoo true false
```

```
operator bool
```

```
value
```

C++ 14

```
bool operator()
```

```
value
```

value_type	bool
type	std::integral_constant<bool,value>

```
static_assertstd::enable_if ◦ std::is_pointer
```

```
template <typename T>
void i_require_a_pointer (T t) {
    static_assert(std::is_pointer<T>::value, "T must be a pointer type");
}
```

```

//Overload for when T is not a pointer type
template <typename T>
typename std::enable_if<!std::is_pointer<T>::value>::type
does_something_special_with_pointer (T t) {
    //Do something boring
}

//Overload for when T is a pointer type
template <typename T>
typename std::enable_if<std::is_pointer<T>::value>::type
does_something_special_with_pointer (T t) {
    //Do something special
}

```

`std::add_pointerstd::underlying_type ◦ type◦ std::add_pointer<int>::typeint* ◦`

std :: is_same

C ++ 11

`std::is_same<T, T>◦ truefalse◦`

```

// Prints true on most x86 and x86_64 compilers.
std::cout << std::is_same<int, int32_t>::value << "\n";
// Prints false on all compilers.
std::cout << std::is_same<float, int>::value << "\n";
// Prints false on all compilers.
std::cout << std::is_same<unsigned int, int>::value << "\n";

```

`typedef std::is_same◦ int == int32_t◦`

```

// Prints true on all compilers.
typedef int MyType
std::cout << std::is_same<int, MyType>::value << "\n";

```

`std::is_same◦`

`std::is_same◦`

`int◦`

```

#include <type_traits>
struct foo {
    int member;
    // Other variables
};

struct bar {
    char member;
};

template<typename T>
int AddStructMember(T var1, int var2) {

```

```

// If type T != foo || T != bar then show error message.
static_assert(std::is_same<T, foo>::value ||
    std::is_same<T, bar>::value,
    "This function does not support the specified type.");
return var1.member + var2;
}

```

C++ 11

◦

int char long unsigned int **true**◦

```

std::cout << std::is_integral<int>::value << "\n"; // Prints true.
std::cout << std::is_integral<char>::value << "\n"; // Prints true.
std::cout << std::is_integral<float>::value << "\n"; // Prints false.

```

true◦ float double long double

```

std::cout << std::is_floating_point<float>::value << "\n"; // Prints true.
std::cout << std::is_floating_point<double>::value << "\n"; // Prints true.
std::cout << std::is_floating_point<char>::value << "\n"; // Prints false.

```

Enum

enum class **true**◦

```

enum fruit {apple, pair, banana};
enum class vegetable {carrot, spinach, leek};
std::cout << std::is_enum<fruit>::value << "\n"; // Prints true.
std::cout << std::is_enum<vegetable>::value << "\n"; // Prints true.
std::cout << std::is_enum<int>::value << "\n"; // Prints false.

```

◦

```

std::cout << std::is_pointer<int *>::value << "\n"; // Prints true.
typedef int* MyPTR;
std::cout << std::is_pointer<MyPTR>::value << "\n"; // Prints true.
std::cout << std::is_pointer<int>::value << "\n"; // Prints false.

```

trueenum class◦

```

struct FOO {int x, y;};
class BAR {
public:
    int x, y;
};
enum class fruit {apple, pair, banana};
std::cout << std::is_class<FOO>::value << "\n"; // Prints true.
std::cout << std::is_class<BAR>::value << "\n"; // Prints true.
std::cout << std::is_class<fruit>::value << "\n"; // Prints false.
std::cout << std::is_class<int>::value << "\n"; // Prints false.

```

C++ 11

- ◦
- ◦
-

```
template<typename T>
inline T FastDivideByFour(const T &var) {
    // Will give an error if the inputted type is not an unsigned integral type.
    static_assert(std::is_unsigned<T>::value && std::is_integral<T>::value,
        "This function is only designed for unsigned integral types.");
    return (var >> 2);
}
```

-

```
std::cout << std::is_const<const int>::value << "\n"; // Prints true.
std::cout << std::is_const<int>::value << "\n"; // Prints false.
```

volatile true

```
std::cout << std::is_volatile<static volatile int>::value << "\n"; // Prints true.
std::cout << std::is_const<const int>::value << "\n"; // Prints false.
```

true

```
std::cout << std::is_signed<int>::value << "\n"; // Prints true.
std::cout << std::is_signed<float>::value << "\n"; // Prints true.
std::cout << std::is_signed<unsigned int>::value << "\n"; // Prints false.
std::cout << std::is_signed<uint8_t>::value << "\n"; // Prints false.
```

true

```
std::cout << std::is_unsigned<unsigned int>::value << "\n"; // Prints true.
std::cout << std::is_signed<uint8_t>::value << "\n"; // Prints true.
std::cout << std::is_unsigned<int>::value << "\n"; // Prints false.
std::cout << std::is_signed<float>::value << "\n"; // Prints false.
```

<https://riptutorial.com/zh-TW/cplusplus/topic/4750/>

147:

`Tdynamic_cast<T> static_cast<T> reinterpret_cast<T>const_cast<T>◦`

C ++ `T(expr) C(T) expr ◦`

- *simple-type-specifier* ()
- *simple-type-specifier* ()
- *simple-type-specifier braced-init-list*
- *typename-specifier* ()
- *typename-specifier* ()
- *typename-specifier braced-init-list*
- `dynamic_cast < type-id > ()`
- `static_cast < type-id > ()`
- `reinterpret_cast < type-id > ()`
- `const_cast < type-id > ()`
- `(type-id) cast-expression`

- `dynamic_cast<Derived&>(base) ◦ ◦`
- `static_cast<string&&>(s) ◦`
- `(int)x prvalue ◦`

`reinterpret_cast` “”

- “” ◦
- ◦

`static_cast`

-
- ◦ ◦
- `void` ◦

```
// on some compilers, suppresses warning about x being unused
static_cast<void>(x);
```

- ◦
- ◦ ◦
- `void*T*` ◦

C ++ 11

- `Xstd::move` ◦ ◦

Examples

static_cast^o static_cast^o

```
struct Base {};  
struct Derived : Base {};  
Derived d;  
Base* p1 = &d;  
Derived* p2 = p1; // error; cast required  
Derived* p3 = static_cast<Derived*>(p1); // OK; p2 now points to Derived object  
Base b;  
Base* p4 = &b;  
Derived* p5 = static_cast<Derived*>(p4); // undefined behaviour since p4 does not  
// point to a Derived object
```

static_cast^o

```
struct Base {};  
struct Derived : Base {};  
Derived d;  
Base& r1 = d;  
Derived& r2 = r1; // error; cast required  
Derived& r3 = static_cast<Derived&>(r1); // OK; r3 now refers to Derived object
```

dynamic_cast^o ^o ^o std::bad_cast std::bad_cast^o

```
struct Base { virtual ~Base(); }; // Base is polymorphic  
struct Derived : Base {};  
Base* b1 = new Derived;  
Derived* d1 = dynamic_cast<Derived*>(b1); // OK; d1 points to Derived object  
Base* b2 = new Base;  
Derived* d2 = dynamic_cast<Derived*>(b2); // d2 is a null pointer
```

const_cast constconst^o const_castconst-correct^o const char*

```
void bad_strlen(char*);  
const char* s = "hello, world!";  
bad_strlen(s); // compile error  
bad_strlen(const_cast<char*>(s)); // OK, but it's better to make bad_strlen accept const char*
```

const_castconstconst^o

const_castC ++const^o ^o

```
const int x = 123;  
int& mutable_x = const_cast<int&>(x);  
mutable_x = 456; // may compile, but produces *undefined behavior*
```

reinterpret_castresp^o resp^o ^o ^o

```
int x = 42;
```

```
char* p = static_cast<char*>(&x); // error: static_cast cannot perform this conversion
char* p = reinterpret_cast<char*>(&x); // OK
*p = 'z'; // maybe this modifies x (see below)
```

C++ 11

reinterpret_cast

```
int x = 42;
char& r = reinterpret_cast<char&>(x);
const void* px = &x;
const void* pr = &r;
assert(px == pr); // should never fire
```

C++ 11

reinterpret_cast

```
int x = 123;
unsigned int& r1 = reinterpret_cast<unsigned int&>(x);
int& r2 = reinterpret_cast<int&>(r1);
r2 = 456; // sets x to 456
```

reinterpret_cast

reinterpret_cast

reinterpret_cast

reinterpret_cast

long unsigned long

C++ 11

std::intptr_t

reinterpret_cast

```
void register_callback(void (*fp)(void*), void* arg); // probably a C API
void my_callback(void* x) {
    std::cout << "the value is: " << reinterpret_cast<long>(x); // will probably compile
}
long x;
std::cin >> x;
register_callback(my_callback,
                reinterpret_cast<void*>(x)); // hopefully this doesn't lose information...
```

static_cast

```
class C {
    std::unique_ptr<int> p;
```

```

public:
    explicit C(int* p) : p(p) {}
};
void f(C c);
void g(int* p) {
    f(p); // error: C::C(int*) is explicit
    f(static_cast<C>(p)); // ok
    f(C(p)); // equivalent to previous line
    C c(p); f(c); // error: C is not copyable
}

```

static_cast° static_cast

- “”。

```

const double x = 3.14;
printf("%d\n", static_cast<int>(x)); // prints 3
// printf("%d\n", x); // undefined behaviour; printf is expecting an int here
// alternative:
// const int y = x; printf("%d\n", y);

```

double°

- ```

struct Base { /* ... */ };
struct Derived : Base {
 Derived& operator=(const Derived& other) {
 static_cast<Base&>(*this) = other;
 // alternative:
 // Base& this_base_ref = *this; this_base_ref = other;
 }
};

```

static\_cast ° °

- ;static\_cast°

## C++ 11

- - ◦
  - ◦
  - ◦

```

enum class Format {
 TEXT = 0,
 PDF = 1000,
 OTHER = 2000,
};
Format f = Format::PDF;
int a = f; // error
int b = static_cast<int>(f); // ok; b is 1000
char c = static_cast<char>(f); // unspecified, if 1000 doesn't fit into char
double d = static_cast<double>(f); // d is 1000.0... probably

```

- -

- ◦
- `<= C++ 14> = C++ 17.`

```
enum Scale {
 SINGLE = 1,
 DOUBLE = 2,
 QUAD = 4
};
Scale s1 = 1; // error
Scale s2 = static_cast<Scale>(2); // s2 is DOUBLE
Scale s3 = static_cast<Scale>(3); // s3 has value 3, and is not equal to any enumerator
Scale s9 = static_cast<Scale>(9); // unspecified value in C++14; UB in C++17
```

## C++ 11

- ◦

```
enum Direction {
 UP = 0,
 LEFT = 1,
 DOWN = 2,
 RIGHT = 3,
};
Direction d = static_cast<Direction>(3.14); // d is RIGHT
```

`static_cast` ◦ ◦

◦

◦ `static_cast` ◦ ◦

```
struct A {};
struct B { int x; };
struct C : A, B { int y; double z; };
int B::*p1 = &B::x;
int C::*p2 = p1; // ok; implicit conversion
int B::*p3 = p2; // error
int B::*p4 = static_cast<int B::*>(p2); // ok; p4 is equal to p1
int A::*p5 = static_cast<int A::*>(p2); // undefined; p2 points to x, which is a member
// of the unrelated class B
double C::*p6 = &C::z;
double A::*p7 = static_cast<double A::*>(p6); // ok, even though A doesn't contain z
int A::*p8 = static_cast<int A::*>(p6); // error: types don't match
```

**\*T\***

**C++** `void*T* T◦ static_cast◦ T◦ ◦`

## C++ 11

TT◦

```
// allocating an array of 100 ints, the hard way
```

```

int* a = malloc(100*sizeof(*a)); // error; malloc returns void*
int* a = static_cast<int*>(malloc(100*sizeof(*a))); // ok
// int* a = new int[100]; // no cast needed
// std::vector<int> a(100); // better

const char c = '!';
const void* p1 = &c;
const char* p2 = p1; // error
const char* p3 = static_cast<const char*>(p1); // ok; p3 points to c
const int* p4 = static_cast<const int*>(p1); // unspecified in C++03;
// possibly unspecified in C++11 if
// alignof(int) > alignof(char)
char* p5 = static_cast<char*>(p1); // error: casting away constness

```

## C

**C** (NewType) variable ◦

### C++

- `const_cast<NewType>(variable)`
- `static_cast<NewType>(variable)`
- `const_cast<NewType>(static_cast<const NewType>(variable))`
- `reinterpret_cast<const NewType>(variable)`
- `const_cast<NewType>(reinterpret_cast<const NewType>(variable))`

NewType(expression) ◦ ◦

### C++C++

`reinterpret_cast` ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/3090/>

| S. No |                            | Contributors                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1     | C ++                       | <a href="#">Adhokshaj Mishra</a> , <a href="#">ankit dassor</a> , <a href="#">aquirdturtle</a> , <a href="#">ArchbishopOfBanterbury</a> , <a href="#">Bakhtiar Hasan</a> , <a href="#">Bart van Nierop</a> , <a href="#">Ben H</a> , <a href="#">Bo Persson</a> , <a href="#">Brandon</a> , <a href="#">Brian</a> , <a href="#">BullshitPingu</a> , <a href="#">cb4</a> , <a href="#">celtschk</a> , <a href="#">Cheers and hth. - Alf</a> , <a href="#">chrisb2244</a> , <a href="#">Cody Gray</a> , <a href="#">Community</a> , <a href="#">cpatricio</a> , <a href="#">Curious</a> , <a href="#">Daemon</a> , <a href="#">Daksh Gupta</a> , <a href="#">Danh</a> , <a href="#">darkpsychic</a> , <a href="#">David Bippes</a> , <a href="#">David G.</a> , <a href="#">DeepCoder</a> , <a href="#">Dim_ov</a> , <a href="#">dlemstra</a> , <a href="#">Donald Duck</a> , <a href="#">Dr t</a> , <a href="#">Dylan Little</a> , <a href="#">Edward</a> , <a href="#">emlai</a> , <a href="#">Erick Q.</a> , <a href="#">ethanwu10</a> , <a href="#">Fantastic Mr Fox</a> , <a href="#">Florian</a> , <a href="#">GIRISH kuniyal</a> , <a href="#">greatwolf</a> , <a href="#">honk</a> , <a href="#">Humam Helfawi</a> , <a href="#">Hurkyl</a> , <a href="#">Ilyas Mimouni</a> , <a href="#">Isak Combrinck</a> , <a href="#">itzmukeshy7</a> , <a href="#">Jason Watkins</a> , <a href="#">JedaiCoder</a> , <a href="#">Jerry Coffin</a> , <a href="#">Jim Clark</a> , <a href="#">Johan Lundberg</a> , <a href="#">Jon Harper</a> , <a href="#">jotik</a> , <a href="#">Justin</a> , <a href="#">Justin Time</a> , <a href="#">JVApn</a> , <a href="#">K48</a> , <a href="#">Ken Y-N</a> , <a href="#">Keshav Sharma</a> , <a href="#">kiner_shah</a> , <a href="#">krOoze</a> , <a href="#">Leandros</a> , <a href="#">maccard</a> , <a href="#">Malcolm</a> , <a href="#">Malick</a> , <a href="#">Manan Sharma</a> , <a href="#">manetsus</a> , <a href="#">manlio</a> , <a href="#">Marco A.</a> , <a href="#">Mark Gardner</a> , <a href="#">MasterHD</a> , <a href="#">Matt</a> , <a href="#">Matt Lord</a> , <a href="#">mnoronha</a> , <a href="#">Muhammad Aladdin</a> , <a href="#">Mustaghees</a> , <a href="#">muXXmit2X</a> , <a href="#">mynameisausten</a> , <a href="#">Nathan Osman</a> , <a href="#">Neil A.</a> , <a href="#">Nemanja Boric</a> , <a href="#">neuro</a> , <a href="#">Nicol Bolas</a> , <a href="#">ουφλγλζεηθ</a> , <a href="#">Optimus Prime</a> , <a href="#">Pavel Strakhov</a> , <a href="#">Peter</a> , <a href="#">Praetorian</a> , <a href="#">Qchmqz</a> , <a href="#">Quirk</a> , <a href="#">RamenChef</a> , <a href="#">Rushikesh Deshpande</a> , <a href="#">SajithP</a> , <a href="#">Sam Cristall</a> , <a href="#">Serikov</a> , <a href="#">Shoe</a> , <a href="#">SirGuy</a> , <a href="#">Soapy</a> , <a href="#">Soul_man</a> , <a href="#">theo2003</a> , <a href="#">τολεεζ ευλ qoq</a> , <a href="#">Tom K</a> , <a href="#">TriskaJM</a> , <a href="#">Trizzle</a> , <a href="#">UncleZeiv</a> , <a href="#">VermillionAzure</a> , <a href="#">Walter</a> , <a href="#">Wen Qin</a> , <a href="#">Wexiwa</a> , <a href="#">πάντα ρεῖ</a> , <a href="#">パスカル</a> |
| 2     | Arithmitic Metaprogramming | <a href="#">Meena Alfons</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 3     | C ++ 11                    | <a href="#">NonNumeric</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 4     | C ++ Streams               | <a href="#">Ami Tavory</a> , <a href="#">didiz</a> , <a href="#">JVApn</a> , <a href="#">mpromonet</a> , <a href="#">Sergey</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 5     | C ++                       | <a href="#">didiz</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 6     | C ++                       | <a href="#">4444</a> , <a href="#">JVApn</a> , <a href="#">lorro</a> , <a href="#">mindriot</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 7     | C ++                       | <a href="#">elvis.dukaj</a> , <a href="#">VermillionAzure</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 8     | C ++                       | <a href="#">John Bargman</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 9     | c ++/                      | <a href="#">Daemon</a> , <a href="#">Nicol Bolas</a> , <a href="#">Владимир Стрелец</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 10    | C ++                       | <a href="#">Gaurav Sehgal</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 11    | C ++                       | <a href="#">celtschk</a> , <a href="#">R_Kapp</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

|    |                           |                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 12 | C ++ <sup>“”“”</sup>      | Error - Syntactical Remorse, Henkersmann                                                                                                                                                                                                                                                                                                                                                                                          |
| 13 | C ++                      | John DiFini                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 14 | C ++                      | Adam Trhon, JVApen, King's jester, Misgevolution                                                                                                                                                                                                                                                                                                                                                                                  |
| 15 | constexpr                 | Ajay, Brian, diegodfrf, mtb, Null                                                                                                                                                                                                                                                                                                                                                                                                 |
| 16 | Const                     | amanuel2, Justin Time                                                                                                                                                                                                                                                                                                                                                                                                             |
| 17 | const                     | Barry, Jarod42, Jatin, Justin, Podgorskiy, tenpercent, ThyReaper                                                                                                                                                                                                                                                                                                                                                                  |
| 18 | C                         | パスカル                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 19 | decltype                  | Ajay                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 20 | ISO C ++                  | Bakhtiar Hasan, Barry, C.W.Holeman II, ComicSansMS, didiz, diegodfrf, Guillaume Pascal, Ivan Kush, Johan Lundberg, Justin Time, JVApen, manlio, Marco A., MSalters, Nicol Bolas, sth, vishal                                                                                                                                                                                                                                      |
| 21 | Lambda                    | Adi Lester, Aganju, Ajay, alain, anderas, Andrea Corbelli, Barry, bcmpinc, Brian, Christopher Oezbek, Community, cpplerner, derekerdmann, Edd, Falias, Firas Moalla, honk, Jean-Baptiste Yunès, Johan Lundberg, Johannes Schaub - litb, John Slegers, JVApen, Loki Astari, Loufylouf, M. Viaz, Mike Dvorkin, Nicol Bolas, Patryk, Praetorian, Rakete1111, RamenChef, Ryan Haining, Sergio, Serikov, Snowhawk, teivaz, Yakk, ygram |
| 22 | Pimpl                     | Danh, Daniele Pallastrelli, emlai, Jordan Chapman, JVApen, manlio, Stephen Cross, Yakk                                                                                                                                                                                                                                                                                                                                            |
| 23 | RAII                      | Barry, defube, Jarod42, JVApen, Loki Astari, Niall, Nicol Bolas, RamenChef, Sumurai8, Tannin                                                                                                                                                                                                                                                                                                                                      |
| 24 | RTTI                      | Brian, deepmax, Pankaj Kumar Boora, Roland, Savas Mikail KAPLAN                                                                                                                                                                                                                                                                                                                                                                   |
| 25 | SFINAE                    | Barry, Fox, Jarod42, Jason R, Jonathan Lee, Luc Danton, sp2danny, SU3, w1th0utnam3, Xosdy, Yakk                                                                                                                                                                                                                                                                                                                                   |
| 26 | static_assert             | Jarod42, JVApen, lorro, Marco A., Richard Dally, T.C.                                                                                                                                                                                                                                                                                                                                                                             |
| 27 | std :: function           | elimad, Evgeniy, Nicol Bolas, Tarod                                                                                                                                                                                                                                                                                                                                                                                               |
| 28 | std :: setstd :: multiset | G-Man, JVApen, Mikitori                                                                                                                                                                                                                                                                                                                                                                                                           |
| 29 | Typedef                   | Brian                                                                                                                                                                                                                                                                                                                                                                                                                             |



|    |                                   |                                                                                                                                                                                                              |
|----|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 30 | ODR                               | Brian, Jarod42                                                                                                                                                                                               |
| 31 |                                   | Adrien Descamps, Barry, ChrisN, hello, honk, Johan Lundberg, Justin Time, JVApen, Loki Astari, mpromonet, Nicol Bolas, Nirmal4G, NonNumeric, Null, Peter, relgukxilef, Scott Weldon, T.C., TriskalJM, Venemo |
| 32 | C ++C ++ 11C ++ 14<br>C ++ 17C ++ | wasthishelpful                                                                                                                                                                                               |
| 33 |                                   | didiz, hyoslee, JVApen                                                                                                                                                                                       |
| 34 |                                   | Ajay, Perette Barella                                                                                                                                                                                        |
| 35 |                                   | A. Sarid, Barry, Cody Gray, CroCo, FedeWar, Jarod42, JVApen, manlio, tambre, Tarod, Trevor Hickey, Алексей Неудачин                                                                                          |
| 36 |                                   | Loki Astari, Mads Marquart, manlio, txtechhelp, Алексей Неудачин                                                                                                                                             |
| 37 | OpenMP                            | Andrea Chua, JVApen, Nicol Bolas, Sumurai8                                                                                                                                                                   |
| 38 | std ::<br>unordered_map           | tulak.hord                                                                                                                                                                                                   |
| 39 |                                   | Edward, marcinj, Naor Hadar, RamenChef                                                                                                                                                                       |
| 40 |                                   | Brian                                                                                                                                                                                                        |
| 41 |                                   | Alexey Guseynov, Brian, callyalater, Dr t, Jahid, Jarod42, Johan Lundberg, jotik, Martin Ba, Nemanja Boric, Null, Peter, Rakete1111, Ronen Ness                                                              |
| 42 |                                   | didiz                                                                                                                                                                                                        |
| 43 |                                   | JVApen, Nicol Bolas                                                                                                                                                                                          |
| 44 |                                   | Barry, ChemiCalChems, Curious, fefe, Johannes Schaub - litb, mnoronha, Nicol Bolas, Praetorian, SirGuy                                                                                                       |
| 45 |                                   | chema989, ralismark                                                                                                                                                                                          |
| 46 |                                   | anderas, Barry, Brian, celtschk, Colin Basnett, DawidPi, deepmax, dmi_, Holt, Jarod42, Justin, manlio, Matthieu M., Nicol Bolas, Oz., rhynodegreat, rtmh, sth, TartanLlama, Venki, W.F., ysdx, πάντα ρῆϊ     |
| 47 |                                   | Andrei, Brian, callyalater, Daksh Gupta, Galik, JVApen, madduci, nrrales, RamenChef, ThyReaper                                                                                                               |

|    |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 48 |      | <a href="#">amanuel2</a> , <a href="#">Aravind .KEN</a> , <a href="#">Bim</a> , <a href="#">Brian</a> , <a href="#">legends2k</a>                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 49 |      | <a href="#">Brian</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 50 |      | <a href="#">Denkkar</a> , <a href="#">Fantastic Mr Fox</a> , <a href="#">Jarod42</a> , <a href="#">Nicol Bolas</a> , <a href="#">SajithP</a> , <a href="#">stackptr</a> , <a href="#">T.C.</a>                                                                                                                                                                                                                                                                                                                                                                |
| 51 |      | <a href="#">Ami Tavory</a> , <a href="#">paul-g</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 52 |      | <a href="#">Johannes Schaub - litb</a> , <a href="#">Kunal Tyagi</a> , <a href="#">RamenChef</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 53 |      | <a href="#">Bakhtiar Hasan</a> , <a href="#">Barry</a> , <a href="#">Cody Gray</a> , <a href="#">CoffeandCode</a> , <a href="#">didiz</a> , <a href="#">Galik</a> , <a href="#">Jatin</a> , <a href="#">Johannes Schaub - litb</a> , <a href="#">Justin Time</a> , <a href="#">JVApen</a> , <a href="#">Rakete1111</a> , <a href="#">Sean</a> , <a href="#">Sumurai8</a> , <a href="#">Tim Straubinger</a>                                                                                                                                                    |
| 54 |      | <a href="#">JVApen</a> , <a href="#">Stephen</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 55 |      | <a href="#">Fanael</a> , <a href="#">Johannes Schaub - litb</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 56 |      | <a href="#">Marco A.</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 57 |      | <a href="#">Andrea Corbelli</a> , <a href="#">Asu</a> , <a href="#">Daksh Gupta</a> , <a href="#">darkpsychic</a> , <a href="#">rockoder</a>                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 58 |      | <a href="#">JVApen</a> , <a href="#">turon</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 59 |      | <a href="#">Barry</a> , <a href="#">Community</a> , <a href="#">Dean Seo</a> , <a href="#">start2learn</a> , <a href="#">T.C.</a> , <a href="#">tenpercent</a>                                                                                                                                                                                                                                                                                                                                                                                                |
| 60 |      | <a href="#">anderas</a> , <a href="#">Andrea Chua</a> , <a href="#">Barry</a> , <a href="#">Brian</a> , <a href="#">DeepCoder</a> , <a href="#">emlai</a> , <a href="#">Isak Combrinck</a> , <a href="#">Jarod42</a> , <a href="#">J r my Roy</a> , <a href="#">Johannes Schaub - litb</a> , <a href="#">Julien-L</a> , <a href="#">JVApen</a> , <a href="#">Nicol Bolas</a> , <a href="#">Null</a> , <a href="#">Rakete1111</a> , <a href="#">randag</a> , <a href="#">Roland</a> , <a href="#">T.C.</a> , <a href="#">tenpercent</a> , <a href="#">Yakk</a> |
| 61 |      | <a href="#">deepmax</a> , <a href="#">Galik</a> , <a href="#">Jarod42</a> , <a href="#">Johan Lundberg</a> , <a href="#">JVApen</a> , <a href="#">Stradigos</a>                                                                                                                                                                                                                                                                                                                                                                                               |
| 62 |      | <a href="#">amanuel2</a> , <a href="#">Brian</a> , <a href="#">Kerrek SB</a> , <a href="#">RamenChef</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 63 |      | <a href="#">A. Sarid</a> , <a href="#">Christophe</a> , <a href="#">Jarod42</a> , <a href="#">Jeremi Podlasek</a> , <a href="#">Justin Time</a> , <a href="#">manlio</a>                                                                                                                                                                                                                                                                                                                                                                                      |
| 64 | CRTP | <a href="#">Barry</a> , <a href="#">Brian</a> , <a href="#">Gabriel</a> , <a href="#">honk</a> , <a href="#">Nicol Bolas</a> , <a href="#">Ryan Haining</a>                                                                                                                                                                                                                                                                                                                                                                                                   |
| 65 |      | <a href="#">Brian</a> , <a href="#">Nikola Vasilev</a> , <a href="#">RamenChef</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 66 |      | <a href="#">Brian</a> , <a href="#">start2learn</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 67 |      | <a href="#">In silico</a> , <a href="#">Johannes Schaub - litb</a> , <a href="#">Nicol Bolas</a> , <a href="#">Roland</a>                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 68 |      | <a href="#">Abhinav Gaunial</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 69 |      | <a href="#">2501</a> , <a href="#">Bo Persson</a> , <a href="#">Brian</a> , <a href="#">Dutow</a> , <a href="#">Jahid</a> , <a href="#">Jarod42</a> , <a href="#">jotik</a> , <a href="#">Justin Time</a> , <a href="#">Iz96</a> , <a href="#">manlio</a> , <a href="#">Nicol Bolas</a> , <a href="#">Peter</a>                                                                                                                                                                                                                                               |

|    |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 70 |                        | <a href="#">Brian</a> , <a href="#">Justin Time</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 71 |                        | <a href="#">Brian</a> , <a href="#">Marco A.</a> , <a href="#">Nicol Bolas</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 72 |                        | <a href="#">ibrahim5253</a> , <a href="#">JVApn</a> , <a href="#">Kerrek SB</a> , <a href="#">MathSquared</a> , <a href="#">SingerOfTheFall</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 73 |                        | <a href="#">manlio</a> , <a href="#">ThyReaper</a> , <a href="#">txtechhelp</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 74 |                        | <a href="#">honk</a> , <a href="#">Jonathan Mee</a> , <a href="#">Justin</a> , <a href="#">JVApn</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 75 | <a href="#">/GCC</a>   | <a href="#">Asu</a> , <a href="#">immerhart</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 76 |                        | <a href="#">Vijayabhaskarreddy CH</a> , <a href="#">Yakk</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 77 |                        | <a href="#">aaron斯诺well</a> , <a href="#">Bakhtiar Hasan</a> , <a href="#">Barry</a> , <a href="#">bitek</a> , <a href="#">celtschk</a> , <a href="#">Christopher Oezbek</a> , <a href="#">Community</a> , <a href="#">DeepCoder</a> , <a href="#">Dr t</a> , <a href="#">Ela782</a> , <a href="#">Fantastic Mr Fox</a> , <a href="#">Galik</a> , <a href="#">honk</a> , <a href="#">J_T</a> , <a href="#">Jarod42</a> , <a href="#">Johan Lundberg</a> , <a href="#">Johannes Schaub - litb</a> , <a href="#">John Slegers</a> , <a href="#">Jon Chesterfield</a> , <a href="#">Kevin Katzke</a> , <a href="#">Let_Me_Be</a> , <a href="#">Loki Astari</a> , <a href="#">M. Sadeq H. E.</a> , <a href="#">manetsus</a> , <a href="#">Menasheh</a> , <a href="#">Michael Gaskill</a> , <a href="#">mnoronha</a> , <a href="#">Niall</a> , <a href="#">Nicol Bolas</a> , <a href="#">Null</a> , <a href="#">Peter</a> , <a href="#">Rakete1111</a> , <a href="#">Richard Forrest</a> , <a href="#">Ryan Hilbert</a> , <a href="#">Stephen</a> , <a href="#">T.C.</a> , <a href="#">templatetypedef</a> , <a href="#">tenpercent</a> , <a href="#">user3384414</a> , <a href="#">Yakk</a> , <a href="#">Ze Rubeus</a> , <a href="#">パスカル</a> |
| 78 |                        | <a href="#">ankit dassor</a> , <a href="#">anotherGatsby</a> , <a href="#">Barry</a> , <a href="#">ChemiCalChems</a> , <a href="#">Chris</a> , <a href="#">ChrisN</a> , <a href="#">Christian Rau</a> , <a href="#">ColleenV</a> , <a href="#">Debanjan Dhar</a> , <a href="#">DrZoo</a> , <a href="#">Edward</a> , <a href="#">emlai</a> , <a href="#">holmicz</a> , <a href="#">honk</a> , <a href="#">Johannes Schaub - litb</a> , <a href="#">Justin Time</a> , <a href="#">L.V.Rao</a> , <a href="#">manlio</a> , <a href="#">Nicholas</a> , <a href="#">Nicol Bolas</a> , <a href="#">Null</a> , <a href="#">Ped7g</a> , <a href="#">pmelanson</a> , <a href="#">Pyves</a> , <a href="#">Rakete1111</a> , <a href="#">Sergey</a> , <a href="#">sp2danny</a> , <a href="#">user1336087</a> , <a href="#">VladimirS</a> , <a href="#">Yakk</a>                                                                                                                                                                                                                                                                                                                                                                                         |
| 79 |                        | <a href="#">AndyG</a> , <a href="#">Barry</a> , <a href="#">cplearner</a> , <a href="#">Firas Moalla</a> , <a href="#">Marco A.</a> , <a href="#">Rakete1111</a> , <a href="#">T.C.</a> , <a href="#">Yakk</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 80 |                        | <a href="#">John Burger</a> , <a href="#">start2learn</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81 |                        | <a href="#">Baron</a> , <a href="#">daB0bby</a> , <a href="#">FedeWar</a> , <a href="#">Hindrik Stegenga</a> , <a href="#">Nicol Bolas</a> , <a href="#">Nitinkumar Ambekar</a> , <a href="#">Pietro Saccardi</a> , <a href="#">Reverie Wisp</a> , <a href="#">West</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 82 |                        | <a href="#">anatolyg</a> , <a href="#">Barry</a> , <a href="#">Daniel</a> , <a href="#">Ivan Kush</a> , <a href="#">maccard</a> , <a href="#">manetsus</a> , <a href="#">manlio</a> , <a href="#">MikeMB</a> , <a href="#">MKAROL</a> , <a href="#">Nicol Bolas</a> , <a href="#">Patrick</a> , <a href="#">Ravi Chandra</a> , <a href="#">SajithP</a> , <a href="#">timrau</a> , <a href="#">Trevor Hickey</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 83 |                        | <a href="#">diegodfrf</a> , <a href="#">JVApn</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 84 |                        | <a href="#">Cheers and hth.</a> - <a href="#">Alf</a> , <a href="#">Isak Combrinck</a> , <a href="#">manlio</a> , <a href="#">Matthew Brien</a> , <a href="#">Wen Qin</a> , <a href="#">Wolf</a> , <a href="#">ΦΧοcε Περεύπα</a> ツ                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 85 | <a href="#">I / O.</a> | <a href="#">anderas</a> , <a href="#">ankit dassor</a> , <a href="#">Anonymous1847</a> , <a href="#">AProgrammer</a> ,                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

|    |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----|--|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    |  | Bakhtiar Hasan, bitek, Chachmu, ComicSansMS, didiz, Dietmar Kühl, Dr t, Emanuel Vintilă, Galik, honk, Hurkyl, Jérémie Bolduc, John Strood, JVApen, Loki Astari, manlio, Mathieu K., MikeMB, mindriot, Nicol Bolas, nwp, patmanpato, Rakete1111, RomCoo, Serikov, sheng09, shrike, svgspr, Алексей Неудачин                                                                                                                                                                                                                          |
| 86 |  | Abyx, Ajay, Alexey Voytenko, anderas, Barry, CaffeineToCode, Christopher Oezbek, Cody Gray, ComicSansMS, cplearner, Daksh Gupta, Danh, Daniele Pallastrelli, DeepCoder, Edward, emlai, foxcub, Francis Cugler, honk, Jack Zhou, Jared Payne, Jarod42, Johan Lundberg, Johannes Schaub - litb, jotik, Justin, JVApen, Kerrek SB, King's jester, Loki Astari, manlio, Marco A., MC93, Menasheh, Meysam, PcAF, Rakete1111, Reuben Thomas, Richard Dally, rodrigo, Roland, sami1592, sth, Sumurai8, tysonite, user3684240, Xirema, Yakk |
| 87 |  | Perette Barella, Sergey                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 88 |  | didiz, Nicol Bolas                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 89 |  | jotik, Roland, ThyReaper                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 90 |  | Ami Tavory, AndreiM, Ben Steffan, Brian, Cody Gray, cshu, Dovahkiin, Elias Kosunen, emlai, Emma X, FedeWar, fefe, ggrr, GIRISH kuniyal, Hiura, Jeremi Podlasek, Johannes Schaub - litb, JVApen, kd1508, Ken Y-N, manetsus, manlio, Marco A., Mat, mceo, Motti, Naor Hadar, nbro, Nicol Bolas, Peter, Rakete1111, ralismark, RamenChef, Sebastian Ärleryd, Tannin, Trevor Hickey, Tyler Durden                                                                                                                                       |
| 91 |  | AndreiM, Brian, Jarod42, Yakk                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 92 |  | Ami Tavory, celtschk, Florian, Jahid, Jason Watkins, Justin, JVApen, Nathan Osman, RamenChef, VermillionAzure                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 93 |  | Ami Tavory, Barry, Daniel, Duly Kinsky, Edgar Rokyan, Guillaume Pascal, Jarod42, NinjaDeveloper, Patryk Obara, Peter, Riom                                                                                                                                                                                                                                                                                                                                                                                                          |
| 94 |  | Barry, Benjy Kessler, Brian, callyalater, cb4, celtschk, CodeMouse92, Colin Basnett, DeepCoder, Diligent Key Presser, Eldritch Cheese, eXPerience, FedeWar, Gabriel, Greg, Holt, honk, J_T, Johannes Schaub - litb, Justin, JVApen, Loki Astari, M. Viaz, manlio, Maxito, MSalters, Nicol Bolas, Pontus Gagge, Praetorian, Rakete1111, Ricardo Amores, Ryan Haining, Sergey, SirGuy, Smeeheey, Sumurai8, user1887915, W.F., WMios, Wolf, πάντα ρῆϊ                                                                                  |

|     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 95  |                         | <a href="#">Artalus</a> , <a href="#">Barry</a> , <a href="#">celtschk</a> , <a href="#">Daniele Pallastrelli</a> , <a href="#">Edward</a> , <a href="#">Igor Oks</a> , <a href="#">Jarod42</a> , <a href="#">Johan Lundberg</a> , <a href="#">manlio</a> , <a href="#">Yakk</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 96  |                         | <a href="#">Nicol Bolas</a> , <a href="#">Владимир Стрелец</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 97  |                         | <a href="#">anotherGatsby</a> , <a href="#">Brian</a> , <a href="#">JVApn</a> , <a href="#">mkluwe</a> , <a href="#">Qchmq5</a> , <a href="#">RamenChef</a> , <a href="#">Tejendra</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 98  |                         | <a href="#">Xirema</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 99  |                         | <a href="#">Barry</a> , <a href="#">krOoze</a> , <a href="#">OliPro007</a> , <a href="#">Reuben Thomas</a> , <a href="#">TriskaJMJ</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 100 | C ++                    | <a href="#">Antonio Barreto</a> , <a href="#">datosh</a> , <a href="#">didiz</a> , <a href="#">Jarod42</a> , <a href="#">JVApn</a> , <a href="#">Nikola Vasilev</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 101 |                         | <a href="#">Brian</a> , <a href="#">Cid1025</a> , <a href="#">Jarod42</a> , <a href="#">Roland</a> , <a href="#">sigalor</a> , <a href="#">sth</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 102 | std :: integer_sequence | <a href="#">Dietmar Kühl</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103 | std :: string           | <a href="#">1337ninja</a> , <a href="#">3442</a> , <a href="#">Andrea Corbelli</a> , <a href="#">Barry</a> , <a href="#">Bim</a> , <a href="#">caps</a> , <a href="#">Christopher Oezbek</a> , <a href="#">cppler</a> , <a href="#">crea7or</a> , <a href="#">Curious</a> , <a href="#">drov</a> , <a href="#">Edward</a> , <a href="#">Emil Rowland</a> , <a href="#">emlai</a> , <a href="#">Fantastic Mr Fox</a> , <a href="#">fbrereto</a> , <a href="#">ggrr</a> , <a href="#">Holt</a> , <a href="#">honk</a> , <a href="#">immerhart</a> , <a href="#">Jack</a> , <a href="#">Jahid</a> , <a href="#">Jerry Coffin</a> , <a href="#">Jonathan Mee</a> , <a href="#">jotik</a> , <a href="#">JPNotADragon</a> , <a href="#">jpo38</a> , <a href="#">Justin</a> , <a href="#">JVApn</a> , <a href="#">Ken Y-N</a> , <a href="#">Leandros</a> , <a href="#">Loki Astari</a> , <a href="#">manetsus</a> , <a href="#">manlio</a> , <a href="#">Marc.2377</a> , <a href="#">Matthew</a> , <a href="#">Matthieu M.</a> , <a href="#">Meysam</a> , <a href="#">Michael Gaskill</a> , <a href="#">mpromonet</a> , <a href="#">Niall</a> , <a href="#">Null</a> , <a href="#">Rakete1111</a> , <a href="#">RamenChef</a> , <a href="#">Richard Dally</a> , <a href="#">SajithP</a> , <a href="#">Serikov</a> , <a href="#">sigalor</a> , <a href="#">Skipper</a> , <a href="#">Soapy</a> , <a href="#">sth</a> , <a href="#">T.C.</a> , <a href="#">Tharindu Kumara</a> , <a href="#">Trevor Hickey</a> , <a href="#">user1336087</a> , <a href="#">user2176127</a> , <a href="#">W.F.</a> , <a href="#">Wolf</a> , <a href="#">Yakk</a> |
| 104 | std :: iomanip          | <a href="#">kiner_shah</a> , <a href="#">Nikola Vasilev</a> , <a href="#">Yakk</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 105 | std ::                  | <a href="#">demonplus</a> , <a href="#">Marco A.</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 106 | std ::Modifiers         | <a href="#">Nikola Vasilev</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 107 | std ::                  | <a href="#">Nikola Vasilev</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 108 | std ::                  | <a href="#">Barry</a> , <a href="#">diegodfrf</a> , <a href="#">Jahid</a> , <a href="#">Jared Payne</a> , <a href="#">JVApn</a> , <a href="#">Null</a> , <a href="#">Yakk</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 109 | std ::                  | <a href="#">Andrea Corbelli</a> , <a href="#">ankit dassor</a> , <a href="#">ChrisN</a> , <a href="#">CinCout</a> , <a href="#">ComicSansMS</a> , <a href="#">CygnusX1</a> , <a href="#">davidsheldon</a> , <a href="#">diegodfrf</a> , <a href="#">Fantastic Mr Fox</a> , <a href="#">foxcub</a> , <a href="#">Galik</a> , <a href="#">honk</a> , <a href="#">jmmut</a> , <a href="#">manetsus</a> , <a href="#">manlio</a> , <a href="#">Meysam</a> , <a href="#">Naveen Mittal</a> , <a href="#">Null</a> , <a href="#">Peter</a> , <a href="#">Richard Dally</a> , <a href="#">rick112358</a> , <a href="#">Savan Morya</a> , <a href="#">user1336087</a> , <a href="#">vdaras</a> , <a href="#">Volka</a> , <a href="#">Wyzard</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 110 | std ::                  | <a href="#">Ajay</a> , <a href="#">Bim</a> , <a href="#">demonplus</a> , <a href="#">kiner_shah</a> , <a href="#">Nikola Vasilev</a> , <a href="#">Ravi Chandra</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 111 | std ::                  | <a href="#">2power10</a> , <a href="#">A. Sarid</a> , <a href="#">Aaron Stein</a> , <a href="#">alain</a> , <a href="#">Alex Logan</a> , <a href="#">Ami Tavory</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

|     |        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     |        | , anatolyg, anderas, Andy, AndyG, arunmoezhi, Bakhtiar Hasan, Barry, Benjamin Lindley, bluefog, bone, CHess, CinCout, Cody Gray, Colin Basnett, ComicSansMS, Community, cute_ptr, Daksh Gupta, Daniel, Daniel Stradowski, Dario, David G., David Yaw, DeepCoder, diegodfrf, dkg, Dr t, Duly Kinsky, Ed Cottrell, Edward, ehudt, emlai, enrico.bacis, Falias, Fantastic Mr Fox, Fox, foxcub, gaazkam, Galik, gartenriese, granmirupa, Holt, honk, Hurkyl, iliketocode, immerhart, Isak Combrinck, Jarod42, Jason Watkins, JHBonarius, Johan Lundberg, John Slegers, jotik, jpo38, JVApen, Kevin Katzke, krOoze, Loki Astari, lordjohncena, manetsus, manlio, Marco A., Matt, Michael Gaskill, Misha Brukman, MotKohn, Motti, mtk, NageN, Niall, Nicol Bolas, Null, patmanpato, Paul Beckingham, paul-g, Ped7g, Praetorian, Pyves, R. Martinho Fernandes, Rakete1111, Randy Taylor, Richard Dally, Roddy, Romain Vincent, Rushikesh Deshpande, Ryan Hilbert, Saint-Martin, Samar Yadav, Samer Tufail, Sayakiss, Serikov, Shoe, silvergasp, Skipper, solidcell, Stephen, sth, strangeqargo, T.C., Tamarous, theo2003, Tom, towi, Trevor Hickey, TriskalJM, user1336087, user2176127, Vladimir Gamalyan, Wolf, Yakk |
| 112 | std :: | Yakk                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 113 | std :: | CinCout, Daksh Gupta, Dinesh Khandelwal, Error - Syntactical Remorse, Malcolm, Nikola Vasilev, plasmacel                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114 |        | Barry, Cheers and hth. - Alf, ChemiCalChems, David Doria, didiz, Guillaume Racicot, Justin Time, JVApen                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 115 |        | Alejandro, amchacon, Brian, CaffeineToCode, ComicSansMS, Dair, defube, didiz, Diligent Key Presser, Galik, James Adkison, james large, Jason Watkins, Jeremi Podlasek, mpromonet, Niall, nwp, Rakete1111, Stephen Cross, Sumurai8, Yakk, ysdx, Yuushi                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 116 |        | didiz, Galik, JVApen                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 117 |        | 4444, Adhokshaj Mishra, Ami Tavory, ArchbishopOfBanterbury, Barry, Ben Steffan, celtschk, Curious, Donald Duck, Dr t, elvis.dukaj, Fantastic Mr Fox, Florian, greatwolf, Griffin, Isak Combrinck, Jahid, Jarod42, Jason Watkins, Johan Lundberg, jotik, Justin, Justin Time, JVApen, madduci, Malick, manetsus, manlio, Matt, Michael Gaskill, Morten Kristensen, MSD, muXXmit2X, n.m., Nathan Osman, Nemanja Boric, Peter, Quirk, Richard Dally, Sergey, Tharindu Kumara, Toby, Trygve Laugstøl, VermillionAzure                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 118 |        | Brian                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

|     |         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 119 |         | <a href="#">0x5f3759df</a> , <a href="#">Daksh Gupta</a> , <a href="#">Johan Lundberg</a> , <a href="#">Justin Time</a> , <a href="#">Motti</a> , <a href="#">Sergey</a> , <a href="#">T.C.</a>                                                                                                                                                                                                                                                                                                                                                                                        |
| 120 |         | <a href="#">Ajay</a> , <a href="#">BigONotation</a> , <a href="#">celtschk</a> , <a href="#">defube</a> , <a href="#">Jarod42</a> , <a href="#">Jonathan Lee</a> , <a href="#">Roland</a> , <a href="#">T.C.</a> , <a href="#">Yakk</a>                                                                                                                                                                                                                                                                                                                                                |
| 121 | Elision | <a href="#">Nicol Bolas</a> , <a href="#">TartanLlama</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 122 |         | <a href="#">amanuel2</a> , <a href="#">Roland</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 123 |         | <a href="#">Brian</a> , <a href="#">RamenChef</a> , <a href="#">start2learn</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 124 |         | <a href="#">Anonymous1847</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 125 |         | <a href="#">Brian</a> , <a href="#">Justin Time</a> , <a href="#">Omnifarious</a> , <a href="#">RamenChef</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 126 |         | <a href="#">Cheers and hth.</a> - <a href="#">Alf</a> , <a href="#">sorosh_sabz</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 127 |         | <a href="#">Brian</a> , <a href="#">Daniel Käfer</a> , <a href="#">Emmanuel Mathi-Amorim</a> , <a href="#">marquesm91</a> , <a href="#">RamenChef</a>                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 128 |         | <a href="#">Barry</a> , <a href="#">chrisb2244</a> , <a href="#">cute_ptr</a> , <a href="#">Daniel Jour</a> , <a href="#">Edgar Rokyan</a> , <a href="#">EvgeniyZh</a> , <a href="#">fbrereto</a> , <a href="#">Gal Dreiman</a> , <a href="#">Gaurav Kumar Garg</a> , <a href="#">GIRISH kuniyal</a> , <a href="#">honk</a> , <a href="#">Hurkyl</a> , <a href="#">JPNotADragon</a> , <a href="#">JVApn</a> , <a href="#">Mike H-R</a> , <a href="#">Null</a> , <a href="#">Oz.</a> , <a href="#">Sergey</a> , <a href="#">Serikov</a> , <a href="#">tilz0R</a> , <a href="#">Yakk</a> |
| 129 |         | <a href="#">Brian</a> , <a href="#">define cindy const</a> , <a href="#">Torbjörn</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 130 |         | <a href="#">amanuel2</a> , <a href="#">Justin Time</a> , <a href="#">RamenChef</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 131 |         | <a href="#">an0o0nym</a> , <a href="#">Brian</a> , <a href="#">didiz</a> , <a href="#">JVApn</a> , <a href="#">start2learn</a> , <a href="#">turonl</a>                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 132 |         | <a href="#">ArchbishopOfBanterbury</a> , <a href="#">Archie Gertsman</a> , <a href="#">Ates Goral</a> , <a href="#">Barry</a> , <a href="#">Brian</a> , <a href="#">Candlemancer</a> , <a href="#">chrisb2244</a> , <a href="#">defube</a> , <a href="#">enzom83</a> , <a href="#">James Adkison</a> , <a href="#">Rakete1111</a> , <a href="#">Sergey</a> , <a href="#">start2learn</a> , <a href="#">Xeverous</a> , <a href="#">Yakk</a>                                                                                                                                             |
| 133 |         | <a href="#">Barry</a> , <a href="#">Brian</a> , <a href="#">didiz</a> , <a href="#">Johannes Schaub - litb</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 134 |         | <a href="#">didiz</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 135 |         | <a href="#">asantacreu</a> , <a href="#">Cody Gray</a> , <a href="#">Edward</a> , <a href="#">Jarod42</a> , <a href="#">JVApn</a> , <a href="#">RamenChef</a>                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 136 |         | <a href="#">ADITYA</a> , <a href="#">amanuel2</a> , <a href="#">Brian</a> , <a href="#">Danh</a> , <a href="#">John London</a> , <a href="#">Justin Time</a> , <a href="#">JVApn</a> , <a href="#">Kerrek SB</a> , <a href="#">Loki Astari</a> , <a href="#">manlio</a> , <a href="#">Marco A.</a> , <a href="#">Nicol Bolas</a> , <a href="#">OliPro007</a> , <a href="#">Rakete1111</a> , <a href="#">RamenChef</a> , <a href="#">Roland</a> , <a href="#">start2learn</a>                                                                                                           |
| 137 |         | <a href="#">Ha.</a> , <a href="#">manlio</a> , <a href="#">merlinND</a> , <a href="#">Sumurai8</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 138 |         | <a href="#">Justin Time</a> , <a href="#">RamenChef</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

|     |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 139 |   | <a href="#">alain</a> , <a href="#">callyalater</a> , <a href="#">Cheers and hth.</a> , <a href="#">- Alf</a> , <a href="#">CygnusX1</a> , <a href="#">Fantastic Mr Fox</a> , <a href="#">Fox</a> , <a href="#">Francisco P.</a> , <a href="#">Ian Ringrose</a> , <a href="#">immerhart</a> , <a href="#">InitializeSahib</a> , <a href="#">Johan Lundberg</a> , <a href="#">Justin</a> , <a href="#">Justin Time</a> , <a href="#">Ken Y-N</a> , <a href="#">Kieran Chandler</a> , <a href="#">krOoze</a> , <a href="#">manlio</a> , <a href="#">Marco A.</a> , <a href="#">Maxito</a> , <a href="#">n.m.</a> , <a href="#">Nicol Bolas</a> , <a href="#">Peter</a> , <a href="#">phandinhan</a> , <a href="#">Richard Dally</a> , <a href="#">Sean</a> , <a href="#">signal</a> , <a href="#">silvergasp</a> , <a href="#">Sumurai8</a> , <a href="#">T.C.</a> , <a href="#">Tanjim Hossain</a> , <a href="#">tenpercent</a> , <a href="#">The Philomath</a> , <a href="#">Владимир Стрелец</a> , <a href="#">パスカル</a> |
| 140 |   | <a href="#">deepmax</a> , <a href="#">Error - Syntactical Remorse</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 141 |   | <a href="#">RamenChef</a> , <a href="#">VermillionAzure</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 142 | / | <a href="#">Alexey Voytenko</a> , <a href="#">anderas</a> , <a href="#">aquirdturtle</a> , <a href="#">Brian</a> , <a href="#">callyalater</a> , <a href="#">chrisb2244</a> , <a href="#">Colin Basnett</a> , <a href="#">Dan Hulme</a> , <a href="#">darkpsychic</a> , <a href="#">Dragma</a> , <a href="#">Fantastic Mr Fox</a> , <a href="#">Firas Moalla</a> , <a href="#">Jarod42</a> , <a href="#">Jerry Coffin</a> , <a href="#">jotik</a> , <a href="#">Justin Time</a> , <a href="#">Kerrek SB</a> , <a href="#">Nicol Bolas</a> , <a href="#">Null</a> , <a href="#">OliPro007</a> , <a href="#">PcAF</a> , <a href="#">Ph03n1x</a> , <a href="#">pingul</a> , <a href="#">Rakete1111</a> , <a href="#">Sándor Mátyás Márton</a> , <a href="#">Sergey</a> , <a href="#">silvergasp</a> , <a href="#">Skywrath</a> , <a href="#">Yakk</a>                                                                                                                                                                       |
| 143 |   | <a href="#">Barry</a> , <a href="#">Brian</a> , <a href="#">Emmanuel Mathi-Amorim</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 144 |   | <a href="#">Andrea Chua</a> , <a href="#">Jim Clark</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 145 |   | <a href="#">Brian</a> , <a href="#">celtschk</a> , <a href="#">greatwolf</a> , <a href="#">Jarod42</a> , <a href="#">Yakk</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 146 |   | <a href="#">Jarod42</a> , <a href="#">silvergasp</a> , <a href="#">TartanLlama</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 147 |   | <a href="#">4444</a> , <a href="#">Brian</a> , <a href="#">JVApen</a> , <a href="#">Nikola Vasilev</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |