

無料電子ブック

学習

C Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#c

1

1: C 2

2

2

C 2

3

CAPI 3

3

Examples 4

4

C 4

1 4

5

5

GCC 5

clang 6

Microsoft C 6

6

"" KR C 6

2: - 8

Examples 8

8

8

9

3: 2D 12

Examples 12

2D 12

2D 18

4: Typedef 19

19

19

19
Examples 20
typedef 20
typedef 21
21
21
22
typedef 22
5: Valgrind 24
24
24
Examples 24
24
24
- 24
Valgrind 25
6: X 27
27
27
Examples 27
printfsX 27
28
X 28
29
X4 29
enum 29
7: 31
31
31
31
31
Examples 32

32
32
33
34
35
8: 36
36
36
Examples 36
36
9: 38
38
38
38
Examples 38
gccasm 38
gccasm 39
gcc 40
10: 41
Examples 41
41
main.c 41
source1.c 41
source2.c 41
headerfile.h 42
11: 43
43
Examples 43
43
45
45
46

47
12: 48
48
48
Examples 48
48
strerror 48
49
13: 50
50
50
50
Examples 51
51
51
GNU getopt 52
14: 56
56
56
Examples 56
/ * * / 56
// 57
57
58
15: 59
59
Examples 59
59
60
61
16: 62
62

62
62
Examples 62
goto 62
63
63
63
64
17: 66
66
66
66
67
67
67
68
68
Examples 68
typedef 68
68
69
extern 70
71
_Thread_local 71
18: 73
73
73
C11C 73
C11C 73
Examples 73
73

1 74
19: 76
..... 76
Examples 76
..... 76
..... 78
C99 79
..... 79
..... 79
..... 80
..... 81
..... 81
20: 83
..... 83
..... 83
Examples 83
CppUTest 83
..... 84
CMocka 85
21: 87
..... 87
..... 87
..... 87
..... 87
Examples 87
..... 87
..... 88
..... 89
..... 90
..... 91
22: 92

92
92
92
92
Examples 93
93
fprintf 94
94
getline 95
example.txt 96
96
getline() 96
98
fscanf 100
101
23: 103
103
Examples 103
stdbool.h 103
#define 103
_Bool 104
104
typedefbool 105
24: / 107
Examples 107
107
<inttypes.h>uintptr_t 107
108
2 108
109
printf 110
111

112

25: 114

114

114

Examples 114

114

116

117

118

#if 0 118

119

119

119

120

121

FOREACH 124

C ++C ++C__cplusplus - name mangling 127

128

129

26: IPC 131

131

Examples 131

131

1.1 132

1.2 133

27: 136

136

Examples 136

136

136

136

#pragma once 137

137
137
137
138
139
139
IWYU 139
140
141
28: 142
142
142
142
Examples 142
142
142
sizeof 143
143
143
144
/ 145
145
146
147
148
149
149
149
150
150
void * 150
Const 151

151
151
153
153
153
154
154
155
156
void 157
29: 159
159
159
159
Examples 159
C11 159
30: 161
161
161
161
161
Examples 161
161
163
163
164
164
164
165
reallocptr0freeptr 166
167

alloca 168
169
169
31: 170
170
170
170
170
170
Topoliges 170
170
171
171
171
171
172
172
Examples 173
173
174
n 174
176
177
178
32: C 181
Examples 181
181
- void 181
33: 184
184
184

184

Examples....184

....184

....185

....185

34: 187

....187

Examples....187

....187

....187

....188

Xorshift Generation....189

35: 190

....190

....190

....190

Examples....191

"signal"....191

36: 193

....193

Examples....193

....193

===....193

....195

\ 01....195

....196

....197

realloc....197

....198

....199

....200

....202

....

204

"" 206

208

209

scanf 209

#define 211

211

213

- 214

" 215

double 216

37: 217

217

Examples 217

217

1 217

2 218

Typedef enum 218

219

typename 220

38: 221

Examples 221

C 221

222

223

223

223

224

224

225

39: 226

226

Examples....226
....226
....226
40: 228
Examples....228
//....228
41: /forwhiledo-while....230
....230
....230
/....230
/....230
Examples....230
....230
while....231
Do-While....231
for....232
....233
....233
42: 235
....235
....235
....235
....235
Examples....236
va_list....236
va_list....237
`printf`....237
....240
43: 242
....242
....242

242
Examples 243
243
243
243
44: 245
245
245
245
245
245
245
246
247
247
247
248
248
248
248
249
249
249
249
249
I / O 250
250
250
251
251
Examples 251
251

252

0 252

..... 252

45: 253

..... 253

Examples 253

C 253

..... 254

..... 255

..... 256

Typedef 259

C 260

46: 265

..... 265

Examples 265

..... 265

47: 267

..... 267

Examples 267

..... 267

..... 267

..... 268

..... 268

48: 271

..... 271

..... 271

Examples 271

strlen 271

strcpystrcat 272

strcmpstrncmpstrcasecmpstrncasecmp 273

strtokstrtok_rstrtok_s 274

strchrstrrchr 277

278

278

279

strstr 280

281

282

strspnstrcspn 283

283

283

284

strcpy() 284

snprintf() 285

strncat() 285

strncpy() 286

atoiatof 286

287

strtoX 288

49: 291

291

291

Examples 291

291

292

50: 294

294

294

Examples 295

NULL 295

2 295

return 296

297

298

299
300
300
301
302
302
303
304
2 304
printf 304
305
305
const 306
printfs 306
307
fflush 307
308
getenvstrerrorsetlocale 308
`_Noreturn`` noreturn` 309
51: 311
311
Examples 311
311
Typedef 311
313
315
315
315
316
'struct hack' 316
317
318
318

52: 321
..... 321
..... 321
Examples 321
..... 321
..... 321
..... 322
53: 324
..... 324
..... 324
Examples 324
fmod 324
longfmodffmodl 325
- powpowfpowl 325
54: 327
..... 327
..... 327
..... 327
Examples 329
..... 329
"==" 329
"=" 329
"" 329
">" 330
"<" 330
"> =" 330
"<=" 330
..... 330
..... 332
..... 332
..... 332

332

332

333

333

/ 334

334

AND 334

OR 335

NOT 335

335

/ 335

/ 3 336

337

337

338

338

338

338

338

339

339

339

pointed-to 340

340

340

341

341

341

341

_Alignof 343

344

55: 346

Examples 346
346
346
1 347
56: 349
349
349
Examples 350
350
351
351
351
352
352
353
355
356
57: 357
357
357
Examples 357
/ 357
CC11-§6.5.2.5/ 9 357
358
358
358
358
359
58: 360
360
Examples 360

360

361

59: 363

Examples 363

363

363

364

365

60: 367

Examples 367

if 367

if... else 367

switch 368

if... else elseif... else 370

if... else VS if.. else Ladder 371

61: 373

373

373

373

Examples 374

374

375

375

376

377

378

378

380

382

383

384

62: 385

385

Examples....385

....385

....385

....386

....386

....387

struct....387

63: **389**

....389

....389

Examples....389

....389

....390

....390

typedef....**390**

....391

....**391**

....391

....392

....392

....**393**

....**393**

....393

....393

....**396**

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: c-language

It is an unofficial and free C Language 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 Language.

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をいめる

Cはプログラミングであり、プログラミング、スコープ、をサポートしていますが、システムはくのしないをぎます。、Cはなににマップするをするため、はオペレーティングシステムをむアセンブリでコードされていたアプリケーションや、スーパーコンピュータからみみシステムまでのコンピュータのさまざまなアプリケーションソフトウェア。

レベルのにもかかわらず、はクロスプラットフォームプログラミングをするようにされています。にしてなCプログラムは、ソースコードをほとんどすることなく、にいのコンピュータプラットフォームおよびオペレーティングシステムにコンパイルできます。このは、みみマイクロコントローラからスーパーコンピュータまで、にいプラットフォームでになっています。

CはもともとBell Labsで1969から1973までDennis Ritchieによってされ、 Unixオペレーティングシステムのにされていました。、これまでくののコンピュータアーキテクチャやオペレーティングシステムでなさまざまなベンダのCコンパイラをして、もくされているプログラミングの1つになっています。

なコンパイラ

Cプログラムをコンパイルするプロセスは、コンパイラとオペレーティングシステムによってなります。ほとんどのオペレーティングシステムはコンパイラなしでされるため、インストールするがあります。なコンパイラのはのとおりです。

- GCC、GNUコンパイラ·コレクション
- clangLLVMのCファミリのフロントエンド
- MSVC、Microsoft Visual C / C ++ビルドツール

のドキュメントは、もなコンパイラのいくつかをいめるのをしています。

- Microsoft Visual Cをいめる
- GCCをいめる

コンパイラCのバージョンサポート

コンパイラにはCにする々なレベルのサポートがありますが、C99をにサポートしていないものがたくさんあります。たとえば、2015のリリースでは、MSVCはC99のくをサポートしていますが、のサポートはとわれますやCライブラリ `<tgmath.h>` らはず「にした」をしていますか Wikipedia には、なコンパイラがするサポートをすがあります 。

あるコンパイラにGCCは、コンパイラプロデューサがとする、であるとえられるのをするコンパイラをしている、あるいはしけているが、Cのにはまれていない。これらのはコンパイラであるため、がないとみなすことができ、コンパイラはのコンパイラバージョンでそれらをまたはできます。そのようなのは、に、コンパイラフラグによってすることができる。

さらに、くのは、ターゲットとするまたはプラットフォームによってされたのバージョンのCのみをサポートするコンパイラをっています。

コンパイラをするは、ターゲットでできるバージョンのCをにサポートするコンパイラをすることをおめします。

コードスタイルトピック

Cではがではないためつまり、コードのにはしません、プログラマはコードをみやすくしやすくするためにをすることがよくあります。これをコードスタイルといいます 。これは、ソースコードをするにされるのとガイドラインです。のインデント、スペースやタブの、ブレースの、オペレータやのりのスペースの、のけなどのをいます。

コードスタイルはではカバーされておらず、ににづいていますさまざまなゞがみやすいさまざまなスタイルをつけています。のコードでスタイルにするもなアドバイスは、がもであることです。スタイルをしてそれにすることです。ないには、のスタイルをするのではなく、しばしばプログラマーがするさまざまなきスタイルがあることをするだけでです。

なインデントスタイルには、KRスタイル、Allmanスタイル、GNUスタイルなどがあります。これらのスタイルのいくつかにはなるバリエーションがあります。たとえばAllmanは、のAllmanまたはなAllman-8のいずれかとしてされます。のあるスタイルのは、 Wikipediaにありま す。このようなスタイルは、やがコードにするくのゞがするためににするからられているので、もがスタイルをっているとにコードをむことができます。えば、 GNUフォーマットガイド GNUコーディング

UpperCamelCase、lowerCamelCase、lower_case_with_underscore、ALL_CAPSなどのようなのがあります。これらのスタイルは、さまざまなオブジェクトやでするためにさまざまなでみわされますマクロでは、くの、ALL_CAPSスタイルがされます

なぜなら、Picoのようなもっとのスタイルはされないのにし、SOのドキュメントではKRスタイルがにされることがされています。

CではカバーされていないライブラリとAPIしたがってここではにはなりません

- POSIX APIえば、 PThreads 、 ソケット 、 シグナル をカバーする

バージョン

バージョン		
KR	なし	1978-02-22
C89	ANSI X3.159-1989	1989-12-14

バージョン		
C90	ISO / IEC 98991990	1990-12-20
C95	ISO / IEC 9899 / AMD11995	1995-03-30
C99	ISO / IEC 98991999	1999-12-16
C11	ISO / IEC 98992011	2011-12-15

Examples

こんにちは

"Hello、World"をにするシンプルなCプログラムをするには、 テキストエディタをして、のソースコードをむしいファイル `hello.c` - ファイルは `.c`なければならないをします。

こんにちはC

```
#include <stdio.h>

int main(void)
{
    puts("Hello, World");
    return 0;
}
```

Coliruのライブデモ

このなプログラムを1ずつてみましょう

```
#include <stdio.h>
```

このは、プログラムにライブラリヘッダーファイル`stdio.h`をめるようにコンパイラにします。ヘッダーは、、マクロ、およびデータをむファイルです。ヘッダーファイルをするにインクルードするがあります。このには`stdio.h`がまれているので、 `puts()`puts()びすことができます。

ヘッダーのをしてください。

```
int main(void)
```

このは、のをします。 `main` 、するのと `void` 、なし、このがすの `int` をします。プログラムのは `main()` からまります。

```
{
    …
}
```

はペアでされ、コードブロックがどこでされ、するかをします。それらはくのでることができますが、このはのとをします。

```
    puts("Hello, World");
```

このは`puts()`をびしてテキストをデフォルトではスクリーンにし、そのにをけます。されるは、かっこにまれます。

`"Hello, World"`は、にきまれるです。 Cでは、すべてのリテラルは`"…"`になければなりません。

のをしてください。

Cプログラムでは、すべてのステートメントはセミコロン `;` でするがあります。

```
    return 0;
```

`main()`をしたとき、 `int`すとしてしました。つまり、をすがあります。このでは、0をします。これは、プログラムがにしたことをすためにされます。 `return 0;``return 0;`ステートメントをすると、プロセスはします。

プログラムの

シンプルなテキストエディタには、 `vim`または`gedit` Linux、または`Notepad` Windowsがあります。クロスプラットフォームエディタには、 `Visual Studio Code`または`Sublime Text`もまれています。

エディタは、RTFやそののではなくプレーンテキストファイルをするがあります。

プログラムのコンパイルと

プログラムをするには、まずこのソースファイル `hello.c` をファイルにコンパイルするがありますえば、Unix / Linuxシステムでは`hello` 、Windowsでは`hello.exe` 。 これは、Cのコンパイラをしてわれます。

コンパイルのをる

GCCをしてコンパイルする

GCC GNU Compiler Collectionはくわれているcコンパイラです。それをするには、ターミナルをき、コマンドラインをしてソースファイルのにしてから、のコマンドをします。

```
gcc hello.c -o hello
```

ソースコード `hello.c` にエラーがない、コンパイラはバイナリファイルをします。 バイナリファイルのは、 `-o`コマンドラインオプション `hello` のでします。これがなファイルです。

また、プログラムがしたりしないをきこすのあるをするためのオプション`-Wall -Wextra -Werror`することもできます。このなプログラムではありませんが、これをするです

```
gcc -Wall -Wextra -Werror -o hello hello.c
```

clangコンパイラの

`clang`をってプログラムをコンパイルするには、をできます

```
clang -Wall -Wextra -Werror -o hello hello.c
```

、 `clang`コマンドラインオプションはGCCのオプションにています。

コマンドラインからMicrosoft Cコンパイラをする

Visual StudioをサポートするWindowsシステムでMicrosoft `cl.exe`コンパイラをしている、すべてのがされている、このCのは、コマンドがされるディレクトリでな`hello.exe`をするのコマンドをしてコンパイルされます `cl`は`/W3`ようなオプションがあり、GCCのは`-Wall`などとしています。

```
cl hello.c
```

プログラムの

コンパイルすると、ターミナルで`./hello`とすることによってバイナリファイルをすることができます。に、コンパイルされたプログラムは、コマンドプロンプトに`Hello, World` 、それにくをします。

オリジナル "こんにちは、" KR Cで

はオリジナルの "Hello、World"です。 Brian KernighanとDennis RitchieによるCプログラミング RitchieはBell LabsのCプログラミングのオリジナルであり、「KR」とばれています。

KR

```
#include <stdio.h>

main()
{
    printf("hello, world\n");
}
```

Cプログラミングは、の1978をくではされておらず、C90コードをけれるようされていないり、ほとんどのコンパイラではコンパイルされないがあることにしてください。

`main()`なりのがなく、 `return`がけているため、KRののののはがいとえられます。こののは、いC89にかれたものです。 C89では、 `main`のは`int`にデフォルトされますが、KRのではにされたがされません。 C99のでは、りのはですが、C99 5.1.2.2.3でされたなケースのため、 `main` および`main`のみの`return`をすることはです。これは0をすこととですこれはをします。

とほとんどのポータブルフォーム`main`ホストされているシステムのは、 `int main (void)`プログラムは、のコマンドライン、またはしないとき`int main(int argc, char **argv)`プログラムは、コマンドラインをしないとき。

C90§5.1.2.2.3 プログラム

> のびしから`main`へのは、 `main`によってされたをとして`exit`をびすのとじです。 `main`がをしないりをすると、ホストにされるはです。

C90は§6.6.6.4 **`return`**を

> のない`return`がされ、びしのがびしによってされる、そのはです。をする`}`することは、なしで`return`をすることとじです。

C99§5.1.2.2.3 プログラム

> `main`のりのが`int`とのあるである、のびしから`main`へのりは、 `main`によってされたをとして`exit`をびすこととです。 `main`をする`}`にすると0がされます。りのが`int`とがない、ホストにされるステータスはされません。

オンラインでCをいめるをむ https://riptutorial.com/ja/c/topic/213/cをいめる

2: - のと

Examples

ストリームからみられたの

```
#include <ctype.h>
#include <stdio.h>

typedef struct {
  size_t space;
  size_t alnum;
  size_t punct;
} chartypes;

chartypes classify(FILE *f) {
  chartypes types = { 0, 0, 0 };
  int ch;

  while ((ch = fgetc(f)) != EOF) {
    types.space += !!isspace(ch);
    types.alnum += !!isalnum(ch);
    types.punct += !!ispunct(ch);
  }

  return types;
}
```

classifyは、ストリームからをみり、スペース、、のをえます。それはいくつかのとしをける。

- ストリームからをみむ、はintとしてされます。そうしないと、 EOF ファイルのわりのマーカーとじビットパターンをつのにあいまいさがあるからです。
- えばisspace は、が*unsigned char*または*EOF*マクロのとしてであるとしています。これはfgetcがすものなので、ここでするはありません。
- のりは、ゼロ falseするとゼロ trueするのでのみされます。をカウントするには、このを1または0にするがあります。これは、によってわれます!!。

からをする

```
#include <ctype.h>
#include <stddef.h>

typedef struct {
  size_t space;
  size_t alnum;
  size_t punct;
} chartypes;

chartypes classify(const char *s) {
  chartypes types = { 0, 0, 0 };
  const char *p;
```

```
  for (p= s; p != '\0'; p++) {
    types.space += !!isspace((unsigned char)*p);
    types.alnum += !!isalnum((unsigned char)*p);
    types.punct += !!ispunct((unsigned char)*p);
  }

  return types;
}
```

classifyは、のすべてのをべ、、およびのをえます。それはいくつかのとしをける。

- えばisspace は、が*unsigned char*または*EOF*マクロのとしてであるとしています。
- *pはcharであるため、のとするようにするがあります。
- charは、 signed charまたはunsigned charいずれかとにされています。
- charがunsigned char charとじ、 charのなはすべてunsigned charとしてできるので、はありません。
- charがsigned char charとの、にすにunsigned charにするがあります。そして、こののためにのはわるかもしれませんが、これはまさにこれらのがするものです。
- のりは、ゼロ falseするとゼロ trueするのでのみされます。をカウントするには、このを1または0にするがあります。これは、によってわれます!!。

き

ヘッダーctype.hはCライブラリのです。のとのためのをします。

これらののすべてがつのパラメータ、るintはunsigned charとして*EOF*やのいずれかでなければなりません。

のには 'is'がきます。されたがするをたす、それぞれはのゼロTRUEをします。がたされない、はゼロをしますFALSE。

これらのは、デフォルトのCロケールをしてのようにします。

```
int a;
int c = 'A';
a = isalpha(c); /* Checks if c is alphabetic (A-Z, a-z), returns non-zero here. */
a = isalnum(c); /* Checks if c  is alphanumeric (A-Z, a-z, 0-9), returns non-zero here. */
a = iscntrl(c); /* Checks is c is a control character (0x00-0x1F, 0x7F), returns zero here. */
a = isdigit(c); /* Checks if c is a digit (0-9), returns zero here. */
a = isgraph(c); /* Checks if c has a graphical representation (any printing character except
space), returns non-zero here. */
a = islower(c); /* Checks if c is a lower-case letter (a-z), returns zero here. */
a = isprint(c); /* Checks if c is any printable character (including space), returns non-zero
here. */
a = isupper(c); /* Checks if c is a upper-case letter (a-z), returns zero here. */
a = ispunct(c); /* Checks if c is a punctuation character, returns zero here. */
a = isspace(c); /* Checks if c is a white-space character, returns zero here. */
a = isupper(c); /* Checks if c is an upper-case letter (A-Z), returns non-zero here. */
a = isxdigit(c); /* Checks if c is a hexadecimal digit (A-F, a-f, 0-9), returns non-zero here.
*/
```

C99

```
a = isblank(c); /* Checks if c is a blank character (space or tab), returns non-zero here. */
```

2つのがあります。これらは、 'to'をしてがけられます。これらのはとじをとります。しかし、りはなゼロまたはゼロではありませんが、されたはらかのでされました。

これらのは、デフォルトのCロケールをとして、のようにします。

```
int a;
int c = 'A';

/* Converts c to a lower-case letter (a-z).
 * If conversion is not possible the unchanged value is returned.
 * Returns 'a' here.
 */
a = tolower(c);

/* Converts c to an upper-case letter (A-Z).
 * If conversion is not possible the unchanged value is returned.
 * Returns 'A' here.
 */
a = toupper(c);
```

のは、の127のASCIIセットがタイプによってどのようにされているかをマッピングするcplusplus.comからされています•は、そのがそのでゼロのをすことをします

ASCII		iscntrl	isblank	isspace	イサッパ	イローダー	イソアルファ	isdigit	isxdigit
0x00 .. 0x08	NUL、のコード	•							
0x09	タブ '\ t'	•	•	•					
0x0A .. 0x0D	のコード '\ f'、 '\ v'、 '\ n'、 '\ r'	•		•					
0x0E .. 0x1F	そののコード	•							
0x20	スペース ' '		•	•					
0x21 ..	"$ '* +、 - 。 /								

ASCII		iscntrl	isblank	isspace	イサッパ	イロ―ダ―	イソアルファ	isdigit	isxdigit
0x2F									
0x30 .. 0x39	0123456789							•	•
0x3a .. 0x40	; <=>@								
0x41 .. 0x46	ABCDEF				•		•		•
0x47 .. 0x5A	GHIJKLMNOPQRSTUVWXYZ				•		•		
0x5B .. 0x60	[] ^ _ `								
0x61 .. 0x66	abcdef					•	•		•
0x67 .. 0x7A	ghijklmnopqrstuvwxyz					•	•		
0x7B .. 0x7E	{}								
0x7F	DEL	•							

オンラインで - のとをむ https://riptutorial.com/ja/c/topic/6846/-ctype-h----のと

3: 2Dをにす

Examples

2Dをにす

2をにすことはでなようで、たちはうれしくいています

```
#include <stdio.h>
#include <stdlib.h>

#define ROWS 3
#define COLS 2

void fun1(int **, int, int);

int main()
{
  int array_2D[ROWS][COLS] = { {1, 2}, {3, 4}, {5, 6} };
  int n = ROWS;
  int m = COLS;

  fun1(array_2D, n, m);

  return EXIT_SUCCESS;
}

void fun1(int **a, int n, int m)
{
  int i, j;
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
      printf("array[%d][%d]=%d\n", i, j, a[i][j]);
    }
  }
}
```

しかし、コンパイラここではGCCバージョン4.9.4はそれをうまくできません。

```
$ gcc-4.9 -O3 -g3  -W -Wall -Wextra  -std=c11 passarr.c -o passarr
passarr.c: In function ‘main’:
passarr.c:16:8: warning: passing argument 1 of ‘fun1’ from incompatible pointer type
   fun1(array_2D, n, m);
        ^
passarr.c:8:6: note: expected ‘int **’ but argument is of type ‘int (*)[2]’
 void fun1(int **, int, int);
```

このは2つあります。なは、がポインタではなく、2のさがいわゆるポインタのです。をにすと、ののへのポインタにします.2のは、Cがでソートされているため、のへのポインタにします。

```
#include <stdio.h>
#include <stdlib.h>
```

```
#define ROWS 3
#define COLS 2

void fun1(int (*)[COLS], int, int);

int main()
{
  int array_2D[ROWS][COLS] = { {1, 2}, {3, 4}, {5, 6} };
  int n = ROWS;
  int m = COLS;

  fun1(array_2D, n, m);

  return EXIT_SUCCESS;
}

void fun1(int (*a)[COLS], int n, int m)
{
  int i, j;
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
      printf("array[%d][%d]=%d\n", i, j, a[i][j]);
    }
  }
}
```

らができない、をすがあります 。

```
#include <stdio.h>
#include <stdlib.h>

#define ROWS 3
#define COLS 2

void fun1(int (*)[COLS], int);

int main()
{
  int array_2D[ROWS][COLS] = { {1, 2}, {3, 4}, {5, 6} };
  int rows = ROWS;

  /* works here because array_2d is still in scope and still an array */
  printf("MAIN: %zu\n",sizeof(array_2D)/sizeof(array_2D[0]));

  fun1(array_2D, rows);

  return EXIT_SUCCESS;
}

void fun1(int (*a)[COLS], int rows)
{
  int i, j;
  int n, m;

  n = rows;
  /* Works, because that information is passed (as "COLS").
     It is also redundant because that value is known at compile time (in "COLS"). */
  m = (int) (sizeof(a[0])/sizeof(a[0][0]));
```

```
  /* Does not work here because the "decay" in "pointer decay" is meant
     literally--information is lost. */
  printf("FUN1: %zu\n",sizeof(a)/sizeof(a[0]));

  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
      printf("array[%d][%d]=%d\n", i, j, a[i][j]);
    }
  }
}
```

C99

のはあらかじめされており、コンパイルにされていますが、のCISO / IEC 98991999、ISO / IEC 98992011のはVLATODOlink itのではオプションになっていますが、ほとんどののCコンパイラでサポ―トされていますTODOMS Visual Studioでサポ―トされているかどうかをしてください。

```
#include <stdio.h>
#include <stdlib.h>

/* ALL CHECKS OMMITTED!*/

void fun1(int (*)[], int rows, int cols);

int main(int argc, char **argv)
{
  int rows, cols, i, j;

  if(argc != 3){
     fprintf(stderr,"Usage: %s rows cols\n",argv[0]);
     exit(EXIT_FAILURE);
  }

  rows = atoi(argv[1]);
  cols = atoi(argv[2]);

  int array_2D[rows][cols];

  for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
      array_2D[i][j] = (i + 1) * (j + 1);
      printf("array[%d][%d]=%d\n", i, j, array_2D[i][j]);
    }
  }

  fun1(array_2D, rows, cols);

  exit(EXIT_SUCCESS);
}

void fun1(int (*a)[], int rows, int cols)
{
  int i, j;
  int n, m;

  n = rows;
  /* Does not work anymore, no sizes are specified anymore
  m = (int) (sizeof(a[0])/sizeof(a[0][0])); */
  m = cols;
```

```
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
      printf("array[%d][%d]=%d\n", i, j, a[i][j]);
    }
  }
}
```

これはしません、コンパイラはをう

```
$ gcc-4.9 -O3 -g3  -W -Wall -Wextra  -std=c99 passarr.c -o passarr
passarr.c: In function ‘fun1’:
passarr.c:168:7: error: invalid use of array with unspecified bounds
       printf("array[%d][%d]=%d\n", i, j, a[i][j]);
```

を`void fun1(int **a, int rows, int cols)`することによって、にのびしでエラーがしたは、しになります。そのため、コンパイラはったでもをってしまいます

```
$ gcc-4.9 -O3 -g3  -W -Wall -Wextra  -std=c99 passarr.c -o passarr
passarr.c: In function ‘main’:
passarr.c:208:8: warning: passing argument 1 of ‘fun1’ from incompatible pointer type
   fun1(array_2D, rows, cols);
        ^
passarr.c:185:6: note: expected ‘int **’ but argument is of type ‘int (*)[(sizetype)(cols)]’
 void fun1(int **, int rows, int cols);
```

たちはいくつかのやりでることができます。その1つは、すべてをしてなポインタジャグリングをすることです。

```
#include <stdio.h>
#include <stdlib.h>

/* ALL CHECKS OMMITTED!*/

void fun1(int (*)[], int rows, int cols);

int main(int argc, char **argv)
{
  int rows, cols, i, j;

  if(argc != 3){
     fprintf(stderr,"Usage: %s rows cols\n",argv[0]);
     exit(EXIT_FAILURE);
  }

  rows = atoi(argv[1]);
  cols = atoi(argv[2]);

  int array_2D[rows][cols];
  printf("Make array with %d rows and %d columns\n", rows, cols);
  for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
      array_2D[i][j] = i * cols + j;
      printf("array[%d][%d]=%d\n", i, j, array_2D[i][j]);
    }
```

```
  }

  fun1(array_2D, rows, cols);

  exit(EXIT_SUCCESS);
}

void fun1(int (*a)[], int rows, int cols)
{
  int i, j;
  int n, m;

  n = rows;
  m = cols;

  printf("\nPrint array with %d rows and %d columns\n", rows, cols);
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
      printf("array[%d][%d]=%d\n", i, j, *( (*a) + (i * cols + j)));
    }
  }
}
```

あるいは、たちはそれをしくい、なを`fun1`ます。これをうには`fun1`へのをするがあります。のサイズはののになければなりません。よりみやすくするために、をするもそのをしています。

```
#include <stdio.h>
#include <stdlib.h>

/* ALL CHECKS OMMITTED!*/

void fun1(int rows, int cols, int (*)[]);

int main(int argc, char **argv)
{
  int rows, cols, i, j;

  if(argc != 3){
     fprintf(stderr,"Usage: %s rows cols\n",argv[0]);
     exit(EXIT_FAILURE);
  }

  rows = atoi(argv[1]);
  cols = atoi(argv[2]);

  int array_2D[rows][cols];
  printf("Make array with %d rows and %d columns\n", rows, cols);
  for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
      array_2D[i][j] = i * cols + j;
      printf("array[%d][%d]=%d\n", i, j, array_2D[i][j]);
    }
  }

  fun1(rows, cols, array_2D);

  exit(EXIT_SUCCESS);
}
```

```
void fun1(int rows, int cols, int (*a)[cols])
{
  int i, j;
  int n, m;

  n = rows;
  m = cols;

  printf("\nPrint array with %d rows and %d columns\n", rows, cols);
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
      printf("array[%d][%d]=%d\n", i, j, a[i][j]);
    }
  }
}
```

これは、のがではないとのをっているの々にはにえます。これはあまりにならないだけで、ポインタをしてをすようにします。

```
#include <stdio.h>
#include <stdlib.h>

/* ALL CHECKS OMMITTED!*/

void fun1(int rows, int cols, int **);

int main(int argc, char **argv)
{
  int rows, cols, i, j;


  if(argc != 3){
     fprintf(stderr,"Usage: %s rows cols\n",argv[0]);
     exit(EXIT_FAILURE);
  }

  rows = atoi(argv[1]);
  cols = atoi(argv[2]);

  int array_2D[rows][cols];
  printf("Make array with %d rows and %d columns\n", rows, cols);
  for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
      array_2D[i][j] = i * cols + j;
      printf("array[%d][%d]=%d\n", i, j, array_2D[i][j]);
    }
  }
  // a "rows" number of pointers to "int". Again a VLA
  int *a[rows];
  // initialize them to point to the individual rows
  for (i = 0; i < rows; i++) {
      a[i] = array_2D[i];
  }

  fun1(rows, cols, a);

  exit(EXIT_SUCCESS);
}
```

```
void fun1(int rows, int cols, int **a)
{
  int i, j;
  int n, m;

  n = rows;
  m = cols;

  printf("\nPrint array with %d rows and %d columns\n", rows, cols);
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
      printf("array[%d][%d]=%d\n", i, j, a[i][j]);
    }
  }
}
```

フラットアレイを2Dアレイとしてする

しばしばもなは、2Dおよびそれのをフラットメモリとしてすことです。

```
/* create 2D array with dimensions determined at runtime */
double *matrix = malloc(width * height * sizeof(double));

/* initialise it (for the sake of illustration we want 1.0 on the diagonal) */
int x, y;
for (y = 0; y < height; y++)
{
    for (x = 0; x < width; x++)
    {
        if (x == y)
            matrix[y * width + x] = 1.0;
        else
            matrix[y * width + x] = 0.0;
    }
}

/* pass it to a subroutine */
 manipulate_matrix(matrix, width, height);


/* do something with the matrix, e.g. scale by 2 */
void manipulate_matrix(double *matrix, int width, int height)
{
    int x, y;

    for (y = 0; y < height; y++)
    {
        for (x = 0; x < width; x++)
        {
            matrix[y * width + x] *= 2.0;
        }
    }
}
```

オンラインで2Dをにすをむ https://riptutorial.com/ja/c/topic/6862/2dをにす

4: Typedef

き

typedefメカニズムは、ののエイリアスのをにします。しいタイプはされません。くの、typedefをしてコードのをさせたり、のまたはにエイリアスをけたり、またはポインタのをしたりします。

Cでは、typedefは「ストレージクラス」としてされています。staticまたはexternようなストレージクラスがれることがある、にします。

- typedef existing_nameエイリアス;

---

typedefの

typedefはなCプログラムののにつながるがあります。

typedefの

また、タグのないtypedefは、ヘッダファイルのをにすきなです。

してください

```
#ifndef FOO_H
#define FOO_H 1

#define FOO_DEF (0xDEADBABE)

struct bar; /* forward declaration, defined in bar.h*/

struct foo {
    struct bar *bar;
};

#endif
```

このようなでは、typedefsをせずに、コンパイルにfoo.hをインクルードしてFOO_DEFをすることがFOO_DEFます。fooのbarメンバーをしようとしない、bar.hファイルをめるはありません。

Typedefと#define

#defineは、typedefていtypedefが、のいがあるさまざまなデータのエイリアスをするためにもされるCのプリプロセッサディレクティブです。

- typedefは#defineをってのエイリアスをするためだけににをえることにられています。
- typedefはコンパイラによってされtypedefが、#defineはプリプロセッサによってされます。
- #define cptr char *

ろに`cptr a, b;`くことにしてください`cptr a, b; typedef char *cptr;`とじことをしません`typedef char *cptr; cptr a, b;`き`cptr a, b;`。 `#define`では、 `b`はの`char`ですが、 `typedef`つポインタでもあり`typedef`。

Examples

との**typedef**

`struct`エイリアスをけることができます

```
typedef struct Person {
    char name[32];
    int age;
} Person;

Person person;
```

structをするなとべて、プログラマーは`struct`のインスタンスをするたびに`struct`をつはありません。

`Person struct Person`とはにというは、のセミコロンまでされていないことにしてください。したがって、じへのポインタをむがあるリンクされたリストとツリーでは、のいずれかをするがあります。

```
typedef struct Person {
    char name[32];
    int age;
    struct Person *next;
} Person;
```

または

```
typedef struct Person Person;

struct Person {
    char name[32];
    int age;
    Person *next;
};
```

`union`の`typedef`のはにています。

```
typedef union Float Float;

union Float
{
    float f;
    char  b[sizeof(float)];
};
```

`float`

をするバイトをするために、これにたをすることができます。

typedefのな

データにいをけるため

のわりに

```
long long int foo;
struct mystructure object;
```

1つはできます

```
/* write once */
typedef long long ll;
typedef struct mystructure mystruct;

/* use whenever needed */
ll foo;
mystruct object;
```

これにより、タイプがプログラムでもされるになけのがります。

の

データタイプのは、アーキテクチャによってなります。たとえば、 `int`はあるでは2バイトで、のでは4バイトです。プログラムがしくするために4バイトタイプをするがあるとします。

1つのでは、 `int`のサイズを2バイトとし、 `long`のサイズを4バイトとします。のでは、 `int`のサイズを4バイト、 `long`さを8バイトとします。プログラムが2のをしてされている、

```
/* program expecting a 4 byte integer */
int foo; /* need to hold 4 bytes to work */
/* some code involving many more ints */
```

プログラムをのでするには、すべての`int`を`long`にするがあります。

```
/* program now needs long */
long foo; /*need to hold 4 bytes to work */
/* some code involving many more longs - lot to be changed */
```

これをけるために、 `typedef`

```
/* program expecting a 4 byte integer */
typedef int myint; /* need to declare once - only one line to modify if needed */
myint foo; /* need to hold 4 bytes to work */
/* some code involving many more myints */
```

に、プログラムをべるのではなく、 `typedef`ステートメントだけをするがあり`typedef`。

<stdint.h>ヘッダーとする<inttypes.h>ヘッダーは、さまざまなサイズのにしてtypedefをしてのをしtypedefこれらのは、サイズのがなのコードでは、しばしばのです。たとえば、 uint8_tはなし8ビットです。 int64_tはき64ビットです。 uintptr_tは、オブジェクトへのポインタをするのになきさのなしです。これらのタイプはにはオプションですが、できないことはまれです。 uint_least16_t なくとも16ビットのなしとint_fast32_t 32ビットのきのようなバリエーションがあります。また、 intmax_tとuintmax_tは、でサポートされているのです。これらのタイプはです。

をするか、またはみやすさをさせる

データのセットにのがある、 typedefをしてのあるをけることができます。さらに、がされるようにデータのプロパティがされた、プログラムをべるわりにtypedefだけをするがありtypedef。

ポインタの**typedef**

typedefをすると、ポインタのをにすることができます。たちがいくつかのをっているとしましょう。それらのはすべてじシグネチャをち、をってさまざまなでかをします

```
#include<stdio.h>

void print_to_n(int n)
{
    for (int i = 1; i <= n; ++i)
        printf("%d\n", i);
}

void print_n(int n)
{
    printf("%d\n, n);
}
```

これで、 typedefをってprinterというのポインタをできます。

```
typedef void (*printer_t)(int);
```

これは、のintをとり、もさないへのポインタのprinter_tというのをします。このは、ののシグネチャにします。それをするために、されたのをし、のの1つをすポインタをりてます。

```
printer_t p = &print_to_n;
void (*p)(int) = &print_to_n; // This would be required without the type
```

に、ポインタがすをびす

```
p(5);           // Prints 1 2 3 4 5 on separate lines
(*p)(5);        // So does this
```

したがって、 typedefは、ポインタをうに、よりなをにします。これは、へのなど、よりなでポインタがされているに、よりらかになります。

されたポインタなしでポインタをパラメータとしてるをしている、は、

```
void foo (void (*printer)(int), int y){
    //code
    printer(y);
    //code
}
```

ただし、 `typedef`ではのようになり`typedef`。

```
void foo (printer_t printer, int y){
    //code
    printer(y);
    //code
}
```

に、はポインタをすことができます。また、 `typedef`すると、をよりにすることができます。

なは、 `<signal.h>` `signal`です。そのはCから

```
void (*signal(int sig, void (*func)(int)))(int);
```

これは2つのをとるです`int`ととして`int`をとり、もさないへのポインタで、2ののようにへのポインタをします。

へのポインタのエイリアスとして`SigCatcher`をした

```
typedef void (*SigCatcher)(int);
```

`signal()`をすることができます

```
SigCatcher signal(int sig, SigCatcher func);
```

として、これはしやすいCがをうをすることをしなかったとしても。 `signal`は2つの、り`int`と`SigCatcher`、そしてそれがし`SigCatcher` - `SigCatcher`けるへのポインタである`int`をもしませんが。

へのポインタに`typedef`をすると、がになりますが、でコードをするのにをくもありますので、してなをしてください。 ポインタもしてください。

オンラインでTypedefをむ https://riptutorial.com/ja/c/topic/2681/typedef

5: Valgrind

- valgrind プログラム オプション < テスト

Valgrindは、Cプログラムのメモリにするエラーをするためにできるデバッグツールです。Valgrindは、りてられたをえたきみやみみ、 `free()` へのなびしをむ、なポインタなどのエラーをするためにできます。また、メモリプロファイリングをうをしてアプリケーションをするためにもできます。

はhttp://valgrind.orgをしてください。

Examples

ランニング グヴァル グリンド

```
valgrind ./my-program arg1 arg2 < test-input
```

これにより、プログラムがされ、りられたりておよびりてのレポートがされます。また、されていないメモリの、なへのの、mallocをってりてられたブロックのをきめたり、ブロックをしたりするといったなエラーについてします。

フラグの

のようなテストをさらにオンにすることもできます。

```
valgrind -q --tool=memcheck --leak-check=yes ./my-program arg1 arg2 < test-input
```

くのオプションのについては、valgrind --helpをするか、のについてはhttp://valgrind.org/のドキュメントをしてください。

われたバイト - でれる

mallocをびすがフリーではないプログラムはのとおりです。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    char *s;

    s = malloc(26); // the culprint

    return 0;
}
```

ながなければ、valgrindはこのエラーをしません。

しかし、 `--leak-check=yes`または`--tool=memcheck`にすると、プログラムがデバッグモードでコンパイルされた、メモリリークのとなったがされ、されます。

```
$ valgrind -q --leak-check=yes ./missing_free
==4776== 26 bytes in 1 blocks are definitely lost in loss record 1 of 1
==4776==    at 0x4024F20: malloc (vg_replace_malloc.c:236)
==4776==    by 0x80483F8: main (missing_free.c:9)
==4776==
```

プログラムがデバッグモードでコンパイルされていないGCCの`-g`フラグなど、するのどこでリークがしたのか、はされません。

これは、たちにって、そのにりてられたブロックをて、それがなぜされなかったのかをるためにしようとします。

Valgrindのにするもなエラー

Valgrindは、`(file.c:line_no)`でのにエラーがしたをします。 valgrindのエラーはのようにまとめられています。

```
ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

もなエラーはのとおりです。

1. なみり/きみエラー

```
==8451== Invalid read of size 2
==8451==    at 0x4E7381D: getenv (getenv.c:84)
==8451==    by 0x4EB1559: __libc_message (libc_fatal.c:80)
==8451==    by 0x4F5256B: __fortify_fail (fortify_fail.c:37)
==8451==    by 0x4F5250F: __stack_chk_fail (stack_chk_fail.c:28)
==8451==    by 0x40059C: main (valg.c:10)
==8451==  Address 0x700000007 is not stack'd, malloc'd or (recently) free'd
```

これは、コードがプログラムにしていないメモリにアクセスしめるときにします。アクセスされるメモリのサイズも、どのがされたかをします。

2. されていないの

```
==8795== 1 errors in context 5 of 8:
==8795== Conditional jump or move depends on uninitialised value(s)
==8795==    at 0x4E881AF: vfprintf (vfprintf.c:1631)
==8795==    by 0x4E8F898: printf (printf.c:33)
==8795==    by 0x400548: main (valg.c:7)
```

エラーによると、7における`main`の`valg.c`、びしがする`printf()`ためにされていないさ`printf`。

3. メモリのな

```
==8954== Invalid free() / delete / delete[] / realloc()
==8954==    at 0x4C2EDEB: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8954==    by 0x4005A8: main (valg.c:10)
==8954==  Address 0x5203040 is 0 bytes inside a block of size 240 free'd
==8954==    at 0x4C2EDEB: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8954==    by 0x40059C: main (valg.c:9)
==8954==  Block was alloc'd at
==8954==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8954==    by 0x40058C: main (valg.c:7)
```

valgrindによると、コードは`valg.c` *10*でにメモリをし2、 *9*ですでにされていたのにし、 *7*ではブロックにメモリがりてられまし た。

オンラインでValgrindをむ https://riptutorial.com/ja/c/topic/2674/valgrind

6: Xマクロ

き

Xマクロは、コードのりしをにえ、データ/コードのをするためのプリプロセッサベースのです。のデータセットにづくののマクロは、のマスターマクロをしてグループをすることによってサポートされ、そのマクロのテキストは、データにして1つのマクロののからなる。のマクロは、に`X()`というがけられているため、テクニックのです。

Xマクロスタイルのマスターマクロのユーザーは、の`X()`マクロと、そのでマスターマクロをするのをすることがされます。したがって、マスターのマクロは、ユーザーの`X()`のにってされます。このようにして、ソースファイルのなコードのは、「をりさない」DRYというえにされているように、`X()`テキストで1しかされません。

さらに、`X()`をし、さらに1またはマスターマクロをすることにより、Xマクロはするデータとコードのをにすることができます。マクロをするとデータがされますたとえば、またはメンバとして。のはするコードをする。

「Xマクロ」というはマクロのなにしますが、テクニックはそののにしません。そのわりに、のなマクロをできます。

には

- Xマクロにするソースファイルはみにくくなります。
- すべてのマクロのように、Xマクロはにテキストです。つまり、にのをするものではありません。そして
- Xマクロはコードをします。びしにづくとして、Xマクロはにコードをきくします。

Xマクロのいは、Dr. Dobbs http://www.drdobbs.com/the-new-cx-macros/184401387の Randy Meyersの[X-Macros]にあります。

Examples

printfsにXマクロをにする

```
/* define a list of preprocessor tokens on which to call X */
#define X_123 X(1) X(2) X(3)

/* define X to use */
#define X(val) printf("X(%d) made this print\n", val);
X_123
#undef X
/* good practice to undef X to facilitate reuse later on */
```

このでは、プリプロセッサはのコードをします。

```
printf("X(%d) made this print\n", 1);
printf("X(%d) made this print\n", 2);
printf("X(%d) made this print\n", 3);
```

と

```
/* declare items of the enum */
#define FOREACH \
      X(item1) \
      X(item2) \
      X(item3) \
/* end of list */

/* define the enum values */
#define X(id) MyEnum_ ## id,
enum MyEnum { FOREACH };
#undef X

/* convert an enum value to its identifier */
const char * enum2string(int enumValue)
{
    const char* stringValue = NULL;
#define X(id) if (enumValue == MyEnum_ ## id) stringValue = #id;
    FOREACH
#undef X
    return stringValue;
}
```

に、コ―ドにされたをして、をしてをにできます。

```
printf("%s\n", enum2string(MyEnum_item2));
```

Xマクロをとしてえる

Xマクロのアプロ―チは、 "X"マクロのをマスタ―マクロのにすることによってしすることができます。これには、マクロのをし、マクロを "X"マクロとしてできるようにするがあります。

にXマクロとじように、マスタ―マクロは、そのマクロにのをつのリストをします。このバリエ―ションでは、そのようなマクロはのようにされます。

```
/* declare list of items */
#define ITEM_LIST(X) \
      X(item1) \
      X(item2) \
      X(item3) \
/* end of list */
```

のようにアイテムをするコ―ドをするかもしれません

```
/* define macro to apply */
#define PRINTSTRING(value) printf( #value "\n");

/* apply macro to the list of items */
```

```
ITEM_LIST(PRINTSTRING)
```

それはこのコ―ドにされます

```
printf( "item1" "\n"); printf( "item2" "\n"); printf( "item3" "\n");
```

"X"がマスタ―マクロのみみであるXマクロとはに、このスタイルでは、でとしてされるマクロこのでは`PRINTSTRING`をにすることはまたはましくないことさえあります。

コ―ド

X-Macrosはコ―ドの、コ―ドの、リストの、やオブジェクト、のなどにできます。

ここではXマクロをして、4つのコマンドとそののマップをとしてむをします

に、enumのをすることができます。

```
/* All our commands */
#define COMMANDS(OP) OP(Open) OP(Close) OP(Save) OP(Quit)

/* generate the enum Commands: {cmdOpen, cmdClose, cmdSave, cmdQuit, }; */
#define ENUM_NAME(name) cmd##name,
enum Commands {
  COMMANDS(ENUM_NAME)
};
#undef ENUM_NAME

/* generate the string table */
#define COMMAND_OP(name) #name,
const char* const commandNames[] = {
  COMMANDS(COMMAND_OP)
};
#undef COMMAND_OP

/* the following prints "Quit\n": */
printf("%s\n", commandNames[cmdQuit]());
```

に、をenumでびすジャンプテ―ブルをすることもできます。

これには、すべてのがじシグネチャをつがあります。それらがをらずにintをすならば、これをのヘッダ―にれます

```
/* declare all functions as extern */
#define EXTERN_FUNC(name) extern int doCmd##name(void);
COMMANDS(EXTERN_FUNC)
#undef EXTERN_FUNC

/* declare the function pointer type and the jump table  */
```

```
typedef int (*CommandFunc)(void);
extern CommandFunc commandJumpTable[];
```

のがヘッダーにまれているとすると、のすべてがなるコンパイルになることがあります。

```
/* generate the jump table */
#define FUNC_NAME(name) doCmd##name,
CommandFunc commandJumpTable[] = {
  COMMANDS(FUNC_NAME)
};
#undef FUNC_NAME

/* call the save command like this: */
int result = commandJumpTable[cmdSave]();

/* somewhere else, we need the implementations of the commands */
int doCmdOpen(void) {/* code performing open command */}
int doCmdClose(void) {/* code performing close command */}
int doCmdSave(void) {/* code performing save command */}
int doCmdQuit(void) {/* code performing quit command */}
```

のコードでされているこののは、ChromiumでのGPUコマンドのディスパッチです。

オンラインでXマクロをむ https://riptutorial.com/ja/c/topic/628/xマクロ

7: アサーション

き

アサーションは、アサーションにソフトウェアがしたに、されたがでなければならないというです。もなのは、にされるなアサーションです。ただし、 アサーションはコンパイルにチェックされます。

- アサート
- static_assert、メッセージ
- _Static_assert、メッセージ

パラメーター

パラメータ	
スカラーの。	
メッセージ	メッセージにめるリテラル。

`assert`と`static_assert`は`assert.h`されたマクロです。

`assert`のは、ライブラリでされていないマクロ`NDEBUG`します。 `NDEBUG`がされている、 `assert`はノーオペレーションです

```
#ifdef NDEBUG
#  define assert(condition) ((void) 0)
#else
#  define assert(condition) /* implementation defined */
#endif
```

オピニオンは、 `NDEBUG`をにプロダクションのコンパイルにするかどうかによってなります。

- プロのキャンプは、としている`assert`びしが`abort`とがユーザにとってではないので、アサーションメッセージは、エンドユーザーにとってではありません。コードでチェックするなをっているは、するがあります`if/else`のび`exit`または`quick_exit`プログラムをします。 `abort`とはに、これらはプログラムが `atexit`または`at_quick_exit`されたをしてらかのクリーンアップをうことをにします。
- コンキャンプは、 `assert`コールはプロダクションコードではしてすべきではないと`assert`ますが、もしそうであれば、チェックされるはかにっていることをし、がされるとします。したがって、アサーションをプロダクションコードでアクティブにするがいです。なぜなら、それらがすると、はすでにんでいるからです。
- もう1つのは、チェックをにするアサーションのシステムをし、 `abort`がなと「しないエラ

— - テクニカルサポートにしてください」がかもしれないのいをすることです。

`static_assert`はキーワードである`_Static_assert`されます。はコンパイルにチェックされるため、`condition`はでなければなりません。これをとので々にうはありません。

Examples

と

アサーションのユースケースの1つは、とです。これは、によって およびをするためににであり る。として、さはにゼロまたはであるため、このはゼロまたはのをすがあります。

```
#include <stdio.h>
/* Uncomment to disable `assert()` */
/* #define NDEBUG */
#include <assert.h>

int length2 (int *a, int count)
{
    int i, result = 0;

    /* Precondition: */
    /* NULL is an invalid vector */
    assert (a != NULL);
    /* Number of dimensions can not be negative.*/
    assert (count >= 0);

    /* Calculation */
    for (i = 0; i < count; ++i)
    {
        result = result + (a[i] * a[i]);
    }

    /* Postcondition: */
    /* Resulting length can not be negative. */
    assert (result >= 0);
    return result;
}

#define COUNT 3

int main (void)
{
    int a[COUNT] = {1, 2, 3};
    int *b = NULL;
    int r;
    r = length2 (a, COUNT);
    printf ("r = %i\n", r);
    r = length2 (b, COUNT);
    printf ("r = %i\n", r);
    return 0;
}
```

アサーション

アサーションは、そのコードにしたときにファクトがでなければならないとするためにされるステートメントです。アサーションは、されるがたされることをするのにちます。アサーションにされたがの、アクションはありません。 falseのは、コンパイラフラグによってなります。アサーションがになっていると、ったがするとすぐにプログラムがします。それらがになっているときはもしません。アサーションは、リリースではしばしばになっていますが、ビルドやデバッグビルドでアサーションをにし、リリースビルドでアサーションをにするのがです。 がエラーよりもいかいかは、プログラムによってなります。アサーションは、プログラミングエラーをするためにのみしてください。、いパラメータがされます。

```
#include <stdio.h>
/* Uncomment to disable `assert()` */
/* #define NDEBUG */
#include <assert.h>

int main(void)
{
    int x = -1;
    assert(x >= 0);

    printf("x = %d\n", x);
    return 0;
}
```

`NDEBUG`はです

```
a.out: main.c:9: main: Assertion `x >= 0' failed.
```

`NDEBUG`なの

```
x = -1
```

すべてのアサーションをオンまたはオフにしてコードをにコンパイルできるように、 `NDEBUG`グローバルにすることをおめします。これをうなは、 `NDEBUG`をコンパイラのオプションとしてするか、ヘッダー `config.h` でることです。

アサーション

C11

アサーションは、コードのコンパイルにがであるかどうかをチェックするためにされます。そうでない、コンパイラはエラーメッセージをし、コンパイルプロセスをするがあります。

アサーションは、ではなくコンパイルにチェックされるアサーションです。はでなければならず、falseのはコンパイラエラーがします。チェックされるのはで、2のはリテラルでなければなりません。

assertとはなり、 `_Static_assert`はキーワードです。なマクロ`static_assert`は、 `<assert.h>`されています。

```
#include <assert.h>

enum {N = 5};
_Static_assert(N == 5, "N does not equal 5");
static_assert(N > 10, "N is not greater than 10");  /* compiler error */
```

C99

C11は、アサーションのサポートはありませんでした。ただし、C99では、コンパイルのがfalseのにコンパイルエラーをきこすマクロでなアサーションをエミュレートできます。 `_Static_assert` とはなり、2のパラメータはなトークンであるがあり、そのためをしてをできます。アサーションにした、そのはになでされていたため、コンパイラエラーにがられます。

```
#define STATIC_MSG(msg, l) STATIC_MSG2(msg, l)
#define STATIC_MSG2(msg,l) on_line_##l##__##msg
#define STATIC_ASSERT(x, msg) extern char STATIC_MSG(msg, __LINE__) [(x)?1:-1]

enum { N = 5 };
STATIC_ASSERT(N == 5, N_must_equal_5);
STATIC_ASSERT(N > 5, N_must_be_greater_than_5); /* compile error */
```

C99では、ブロックののにをすることができなかったので、このマクロのについてはにでなければなりません。

なコードのアサーション

に、のコードパスがフローのをげるがあるときは、 `assert(0)` を`assert(0)` て、そのようながっていることをすことができます。

```
switch (color) {
    case COLOR_RED:
    case COLOR_GREEN:
    case COLOR_BLUE:
        break;

    default:
        assert(0);
}
```

`assert()` マクロのがfalseを`assert()` たびに、マクロはをエラーストリームにきみ、プログラムをします。このには、 `assert()` ステートメントのファイルとがまれており、デバッグににちます。 `NDEBUG` は、マクロ `NDEBUG` することによってにすることができます。

エラーがしたときにプログラムをさせるもう1つのは、ライブラリ `exit` 、 `quick_exit` または `abort` です。 `exit` と `quick_exit` はあなたのにすことができるをとります。 `abort()` したがって `assert` はプログラムのなとなり、にされるクリーンアップはされません。

`assert()` のなは、デバッグをにすることです。 `abort()` をびすと、 `abort()` ようににすることはできませんが、デバッグがされないことがあります。いくつかのでは、のコンストラクトをするとです。

```
if (color == COLOR_RED || color == COLOR_GREEN) {
   ...
} else if (color == COLOR_BLUE) {
   ...
} else {
   assert(0), abort();
}
```

アサートがになると、 `assert()`コールはデバッグをし、プログラムをします。はして`abort()`びしにしません。アサーションがになると、 `assert()`コールはもせず、 `abort()`がびされます。これにより、プログラムはにこのエラーでします。アサートのとは、デバッグがされるかどうかにかかわらずをえます。

あなたは、このようなままにすることはありません`assert`デバッグは、エンドユーザーにとってではないためとするので、コードを`abort`にインストールされているクリーンアップハンドラあまりにもひどいで`exit`または`quick_exit`することを。

アサートエラーメッセージ

アサーションとともにエラーメッセージをできるトリックがします。、のようなコードをします

```
void f(void *p)
{
    assert(p != NULL);
    /* more code */
}
```

アサーションがしたは、エラーメッセージがています

    アサーションがしましたp= NULL、ファイルmain.c、5

ただし、AND `&&` をしてエラーメッセージをすることもできます

```
void f(void *p)
{
    assert(p != NULL && "function f: p cannot be NULL");
    /* more code */
}
```

、アサーションがした、エラーメッセージはのようになります

    アサーションがしましたp= NULL && "fpはNULLにできません"、ファイルmain.c、5

これがなぜするかについてのは、リテラルがにゼロtrueとされるためです。ブールに`&& 1`をしてもはありません。したがって、 `&& "error message"`しても、コンパイラはしたをします。

オンラインでアサーションをむ https://riptutorial.com/ja/c/topic/555/アサーション

8: アトミック

- `#ifdef __STDC_NO_ATOMICS__`
- `# error this implementation needs atomics`
- `#endif`
- `#include <stdatomic.h>`
- なし_Atomicカウンタ= ATOMIC_VAR_INIT0;

CのとしてのAtomicsは、C11でなオプションのです。

それらのは、なるスレッドでされるへののないアクセスをすることです。がなければ、2つのスレッドがにアクセスすると、のはになります。えば、インクリメント `++` は、いくつかのアセンブラ、みし、、およびストアにすることができる。のスレッドがじをっている、その2つのシーケンスがみい、したにつながるがあります。

- をくすべてのオブジェクトは`_Atomic`でできます。
- これらの `++`や`*=` は、すべてであることがされています。
- ジェネリックとしてされるのがいくつかあります `atomic_compare_exchange`。
- スレッドなるスレッドからアクセスされた、それらにアクセスすることでデータがしないことがされます。
- シグナルハンドラタイプは、それらのすべてのがステートレスであればロックフリーとばれます。この、のフローとシグナルハンドラのでをするためにすることもできます。
- ロックフリーであることがされているデータは`atomic_flag`です。これは、なテストとのハードウエアのにするをうのタイプです。

をするためののはC11のスレッドインタフェースでできます。にスレッドがなデータやコードのクリティカルセクションにアクセスすることをにするmutexの`mtx_t`があります。アトミックができないは、これらをしてレースをするがあります。

Examples

アトミックと

をすることなく、なるスレッドでアトミックににアクセスできます。

```
/* a global static variable that is visible by all threads */
static unsigned _Atomic active = ATOMIC_VAR_INIT(0);


int myThread(void* a) {
  ++active;         // increment active race free
  // do something
```

```
  --active;         // decrement active race free
  return 0;
}
```

にしてされているすべてのオブジェクトをするはされ、それらにアクセスするなるスレッドのにつながりません。

- アトミックオブジェクトのは、、のよりもいにくなります。これには、なロードまたはストアもまれます。したがって、なにのみしてください。
- `a = a+1;`などののと`a = a+1;`には3つのオペレーションが`a`ます。はロード、に、にストアです。これはレースフリーではありません。`a += 1;`のみ`a += 1;`および`a++;`あります。

オンラインでアトミックをむ https://riptutorial.com/ja/c/topic/4924/アトミック

9: インラインアセンブリ

インラインアセンブリは、Cソースコードのでアセンブリをするです。 ISO Cでは、インラインアセンブリのサポートはありません。ではないため、インラインアセンブリのはコンパイラによってなります。はサポートされていますが、インラインアセンブリをするはほとんどなく、くのがありません。

1. パフォーマンスのためののアセンブリをすることによって、コンパイラによってされたアセンブリコードよりもれたパフォーマンスをすることができます。これらのパフォーマンスのはまれであることにしてください。ほとんどの、Cコードをべえるだけでパフォーマンスがし、オプティマイザがそのをたすことができます。
2. ハードウェアインタフェースデバイスドライバまたはプロセッサのスタートアップコードは、しいレジスタにアクセスし、のがのですることをオペレーションののですためめに、いくつかのアセンブリコードをとするがあります。

1. コンパイラのインラインアセンブリのは、あるコンパイラからのコンパイラにじであることがされていません。のコンパイラでサポートされるはずのインラインアセンブリをしてコードをするは、プリプロセッサマクロ `#ifdef` をして、されているコンパイラをします。に、サポートされているコンパイラごとに々のインラインアセンブリセクションをします。
2. プロセッサーの x86プロセッサーのインラインアセンブリーはできず、ARMプロセッサーでするはずです。インラインアセンブリは、のプロセッサまたはプロセッサファミリにされています。なるプロセッサでサポートしたいインラインアセンブリがあるは、プリプロセッサマクロをして、コードがコンパイルされているプロセッサをし、なアセンブリコードセクションをします。
3. のパフォーマンスのインラインアセンブリは、のプロセッサクロックにづいてをしてすることができます。プログラムがよりなクロックをつプロセッサにコンパイルされている、アセンブリコードがどおりにされないことがあります。

Examples

gccasmサポート

gccをったなアセンブリのサポートには、のがあります。

```
asm [ volatile ] ( AssemblerInstructions )
```

ここで、 `AssemblerInstructions`は、されたプロセッサのアセンブリコードです。 volatileキーワードはオプションであり、gccはなasmのでコードをしないため、のもありません。 `AssemblerInstructions`は、のアセンブリをめることができます。 asmルーチンがCのになければならないは、asmステートメントがされます。のは、GCCのマニュアルです。

```
/* Note that this code will not compile with -masm=intel */
```

```
#define DebugBreak() asm("int $3")
```

このでは、コードののでDebugBreak()をすると、アセンブリint $3がされます。 gccはなasmのコードをしなくても、オプティマイザはききしたasmをかすことができることにしてください。のでするがあるアセンブリがあるは、それらを1つのasmステートメントにめます。

gccasmサポート

gccのasmサポートには、のがあります。

```
asm [volatile] ( AssemblerTemplate
                 : OutputOperands
                 [ : InputOperands
                 [ : Clobbers ] ])

 asm [volatile] goto ( AssemblerTemplate
                       :
                       : InputOperands
                       : Clobbers
                       : GotoLabels)
```

ここAssemblerTemplateアセンブラについてのである、 OutputOperandsアセンブリコードですることができるのCである、 InputOperandsパラメータとしてされるのCである、 Clobbersアセンブリコードによってされるリストまたはレジスタであり、 GotoLabelsアセンブリコードでされるgotoラベルです。

フォーマットはCでされ、インラインアセンブリのよりなです。は、ARMプロセッサの16ビットおよび32ビットのをバイトスワップするLinuxカーネルのです。

```
/* From arch/arm/include/asm/swab.h in Linux kernel version 4.6.4 */
#if __LINUX_ARM_ARCH__ >= 6

static inline __attribute_const__ __u32 __arch_swahb32(__u32 x)
{
    __asm__ ("rev16 %0, %1" : "=r" (x) : "r" (x));
    return x;
}
#define __arch_swahb32 __arch_swahb32
#define __arch_swab16(x) ((__u16)__arch_swahb32(x))

static inline __attribute_const__ __u32 __arch_swab32(__u32 x)
{
    __asm__ ("rev %0, %1" : "=r" (x) : "r" (x));
    return x;
}
#define __arch_swab32 __arch_swab32

#endif
```

asmセクションはxをそのおよびパラメータとしてします。 Cは、されたをします。

されたasmでは、gccはCコードのにするのとじにってasmブロックのアセンブリをすることがで

きます。 asmセクションをしないは、asmセクションに`volatile`キーワードをします。

gccマクロのインラインアセンブリ

アセンブリーのをマクロのにれ、をびすようにマクロをすることができます。

```
#define mov(x,y) \
{ \
    __asm__ ("l.cmov %0,%1,%2" : "=r" (x) : "r" (y), "r" (0x0000000F)); \
}

/// some definition and assignment
unsigned char sbox[size][size];
unsigned char sbox[size][size];

///Using
mov(state[0][1], sbox[si][sj]);
```

Cコードにめまれたインラインアセンブリをすると、プログラムのをできます。これは、AESのようなアルゴリズムのようなになにちます。たとえば、AESアルゴリズムでとされるなシフトの、Cの`Rotate Right`シフトのわりにの`Rotate Right`シフトをすることができ`>>`。

'AES256'のでは、 'AddRoundKey'にのようながあります

```
unsigned int w;          // 32-bit
unsigned char subkey[4]; // 8-bit, 4*8 = 32

subkey[0] = w >> 24;     // hold 8 bit, MSB, leftmost group of 8-bits
subkey[1] = w >> 16;     // hold 8 bit, second group of 8-bit from left
subkey[2] = w >> 8;      // hold 8 bit, second group of 8-bit from right
subkey[3] = w;           // hold 8 bit, LSB, rightmost group of 8-bits

/// subkey <- w
```

に`w`のビットを`subkey`にします。

たちは、1つのアセンブリをつ3つのシフト+りて、1つのアサインCにすることができます`Rotate Right`を。

```
__asm__ ("l.ror  %0,%1,%2" : "=r" (* (unsigned int *) subkey)  : "r" (w), "r" (0x10));
```

はまったくじです。

オンラインでインラインアセンブリをむ https://riptutorial.com/ja/c/topic/4263/インラインアセンブリ

10: インライン

Examples

のソースファイルでされるインライン

にびされるさなの、びしにするオーバーヘッドは、そののののかなりのになります。パフォーマンスをさせるの1つは、オーバーヘッドをすることです。

このでは、3つのソースファイルで4つのおよび`main()`をします。これらのうち2つ`plusfive()`と`timestwo()`は、それぞれ "source1.c"と "source2.c"にあるの2つからびされます。`main()`がまれているので、のがあります。

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "headerfile.h"

int main(void) {
    int start = 3;
    int intermediate = complicated1(start);
    printf("First result is %d\n", intermediate);
    intermediate = complicated2(start);
    printf("Second result is %d\n", intermediate);
    return 0;
}
```

source1.c

```
#include <stdio.h>
#include <stdlib.h>
#include "headerfile.h"

int complicated1(int input) {
    int tmp = timestwo(input);
    tmp = plusfive(tmp);
    return tmp;
}
```

source2.c

```
#include <stdio.h>
#include <stdlib.h>
#include "headerfile.h"

int complicated2(int input) {
    int tmp = plusfive(input);
    tmp = timestwo(tmp);
```

```
    return tmp;
}
```

headerfile.h

```
#ifndef HEADERFILE_H
#define HEADERFILE_H

int complicated1(int input);
int complicated2(int input);

inline int timestwo(int input) {
  return input * 2;
}
inline int plusfive(int input) {
  return input + 5;
}

#endif
```

は`timestwo`と`plusfive`のによってびされる`complicated1`と`complicated2`なる「」、またはソースファイルであり、。このようにするためには、ヘッダーにそれらをするがあります。

このようにコンパイルすると、gccをします

```
cc -O2 -std=c99 -c -o main.o main.c
cc -O2 -std=c99 -c -o source1.o source1.c
cc -O2 -std=c99 -c -o source2.o source2.c
cc main.o source1.o source2.o -o main
```

をオンにしないとインラインしないコンパイラーがあるため、-O2オプションをします。

`inline`キーワードのは、のシンボルがオブジェクトファイルにされないことです。それのは、のでエラーがし、オブジェクトファイルをリンクしてファイルをします。 `inline`でないと、じシンボルがの`.o`ファイルでされ、 "されたシンボル"エラーがします。

にシンボルがなでは、シンボルがまったくされないというがあります。それにするには2つのがあります。 1つは、インラインの`extern`を`.c`ファイルの1つにすることです。 `source1.c`のをします。

```
extern int timestwo(int input);
extern int plusfive(int input);
```

のとをすることである`static inline`のわりに`inline`。このメソッドは、にののコピーがこのヘッダでされたすべてのオブジェクトファイルでされるというがあります。

オンラインでインラインをむ https://riptutorial.com/ja/c/topic/7427/インライン

11: エイリアシングとなタイプ

エイリアシングルールのとオブジェクトのなタイプにすることは、2つのなることであり、させるべきではありません。

- エイリアシングは、じオブジェクトをする2つのポインタ`a`と`b`のプロパティで、 `b a == b`です。
- データオブジェクトのなは、Cがそのオブジェクトにしてなをするためにされます。に、なは、2つのポインタがいにエイリアスできるかどうかをするためにされます。

つのポインタをしてオブジェクトをするので、エイリアシングは、のであることができるとう、のポインタをしてされているオブジェクトをすることができ`a b`。あなたのCコンパイラは、そのとにかかわらず、ポインタがにいをエイリアスできるとしなければならない、くののがわれ、くのプログラムがくされます。

Cのなエイリアシングは、どのオブジェクトがいにエイリアスをうまたはしないかをコンパイラのケースがするをします。データポインタにはにすべき2つのがあります。

> のいをしないり、じタイプをつ2つのポインタがエイリアスになることがあります。
>
> なるをつ2つのポインタは、2つののなくとも1つがでないり、エイリアスできません。

ここでははは、々のようなタイプのをにくことを`const`はえば、ある`a double*`と`b`ある`const double*`、コンパイラはがあることをとしなければならない`*a`されることがあり`*b`。

2のルールにすると、なをくがあります。ここで、なエイリアシングにするということは、にはじオブジェクトをすコンパイラ`b`、なるの2つのポインタ`a`と`b`をすることをします。コンパイラは、になるオブジェクトへの2をとることができる、とのそのアイデアされません`*b`あなたはをしてオブジェクトをした`*a`。

そうすると、プログラムのがになります。したがって、Cはポインタのににしいをし、ってそのようながするのをぐのにちます。

> ソースまたはターゲットタイプが`void`でないり、なるタイプをつポインター—のすべてのポインター—はでなければなりません。

つまり、 `const`などのをターゲットタイプにするだけのをわないり、はキャストがです。

にポインタのやキャストをけると、エイリアスのからあなたをします。あなたがにそれらをとしないり、これらのケースはにです。できるだけけるべきです。

Examples

タイプには、タイプではアクセスできません。

オブジェクトが、スレッド、またはでされ、 `char` 、 `unsigned char` 、または`signed char`いずれかのタイプをつオブジェクトのは、タイプではアクセスできません。のでは、 `char`は`int`としてされ、そのは`int`ポインタ`b`ごとにです。

```
int main( void )
{
    char a[100];
    int* b = ( int* )&a;
    *b = 1;

    static char c[100];
    b = ( int* )&c;
    *b = 2;

    _Thread_local char d[100];
    b = ( int* )&d;
    *b = 3;
}
```

なをつデ―タオブジェクトには、ではないのからアクセスすることはできません。ここではのは`int`なので、これはされません。

とポインタのサイズがっていても、このルールからされるわけではありませんが、はです。

これはに、Cでは、 `malloc`またはのがけったバッファをうように、なるのポインタをしてできるのバッファオブジェクトをするがないことをします。

のとじをするしいは、 `union`をすることです。

```
typedef union bufType bufType;
union bufType {
   char c[sizeof(int[25])];
   int i[25];
};

int main( void )
{
    bufType a = { .c = { 0 } }; // reserve a buffer and initialize
    int* b = a.i;      // no cast necessary
    *b = 1;

    static bufType a = { .c = { 0 } };
    int* b = a.i;
    *b = 2;

    _Thread_local bufType a = { .c = { 0 } };
    int* b = a.i;
    *b = 3;
}
```

ここで、 `union`は、コンパイラがからバッファがなるビュ―をしてアクセスできることをっていることをします。これには、すでにバッファに " `int` "のポインタがあり、ポインタがない "view" `ai`

があるというもあります。

なタイプ

データオブジェクトのなタイプは、それにけられたのタイプのするです。

```
// a normal variable, effective type uint32_t, and this type never changes
uint32_t a = 0.0;

// effective type of *pa is uint32_t, too, simply
// because *pa is the object a
uint32_t* pa = &a;

// the object pointed to by q has no effective type, yet
void* q = malloc(sizeof uint32_t);
// the object pointed to by q still has no effective type,
// because nobody has written to it
uint32_t* qb = q;
// *qb now has effective type uint32_t because a uint32_t value was written
*qb = 37;

// the object pointed to by r has no effective type, yet, although
// it is initialized
void* r = calloc(1, sizeof uint32_t);
// the object pointed to by r still has no effective type,
// because nobody has written to or read from it
uint32_t* rc = r;
// *rc now has effective type uint32_t because a value is read
// from it with that type. The read operation is valid because we used calloc.
// Now the object pointed to by r (which is the same as *rc) has
// gained an effective type, although we didn't change its value.
uint32_t c = *rc;

// the object pointed to by s has no effective type, yet.
void* s = malloc(sizeof uint32_t);
// the object pointed to by s now has effective type uint32_t
// because an uint32_t value is copied into it.
memcpy(s, r, sizeof uint32_t);
```

の、そのオブジェクトへのポインタ`uint32_t*`っているはありませんでした。の`uint32_t`オブジェクトをコピーしたというでです。

なエイリアシングにする

のコードでは、に`float`と`uint32_t`サイズがじであるとします。

```
void fun(uint32_t* u, float* f) {
    float a = *f
    *u = 22;
    float b = *f;
    print("%g should equal %g\n", a, b);
}
```

`u`と`f`はなるをつため、コンパイラはなるオブジェクトをしているとなすことができます。 `a`と`b`2つののに`*f`がされているはないので、コンパイラはコードを

```
void fun(uint32_t* u, float* f) {
    float a = *f
    *u = 22;
    print("%g should equal %g\n", a, a);
}
```

つまり、 `*f` 2のロ―ドをにすることができます。

このを ""とふと、

```
 float fval = 4;
 uint32_t uval = 77;
 fun(&uval, &fval);
```

すべてうまくき、かがき

    4は4にしくなければならない

されます。しかし、じポインタをチ―トしてすと、それをした、

```
 float fval = 4;
 uint32_t* up = (uint32_t*)&fval;
 fun(up, &fval);
```

なエイリアシングにします。その、はになります。コンパイラが2のアクセスをになったものにしていて、プログラムがにできないになった、はのようになります。

をする

じの2つのポインタをつ、コンパイラはもすることができず、に`*e`へのが`*f`できるとするがあります

```
void fun(float* e, float* f) {
    float a = *f
    *e = 22;
    float b = *f;
    print("is %g equal to %g?\n", a, b);
}

float fval = 4;
float eval = 77;
 fun(&eval, &fval);
```

すべてうまくき、かがき

    4は4にしいか

されます。じポインタをすと、プログラムはしいことをい、します

    4は22にしいか

これは、たちがいることを、いくつかののでっていれば、であることがすることができます`e`と`f`じデ―タオブジェクトをすことはありません。ポインタパラメ―タに`restrict`をすることで、そのをさせることができます

```
void fan(float*restrict e, float*restrict f) {
    float a = *f
    *e = 22;
    float b = *f;
    print("is %g equal to %g?\n", a, b);
}
```

に、コンパイラはに`e`と`f`がのオブジェクトをしているとします。

バイトの

オブジェクトがなタイプをっているしたら、のが、でないり、あなたは、ののポインタをしてそれをしようとするべきではない`char`、 `signed char`または`unsigned char`。

```
#include <inttypes.h>
#include <stdio.h>

int main(void) {
  uint32_t a = 57;
  // conversion from incompatible types needs a cast !
  unsigned char* ap = (unsigned char*)&a;
  for (size_t i = 0; i < sizeof a; ++i) {
    /* set each byte of a to 42 */
    ap[i] = 42;
  }
  printf("a now has value %" PRIu32 "\n", a);
}
```

これはなプログラムで、

はがある707406378

これはのでします。

- `unsigned char`でられる々のバイトにしてアクセスがわれるため、はにされます。
- オブジェクトへの2つのビュー`a`およびthrough `*ap`、aliasですが、 `ap`はへのポインタなので、なエイリアシングはされません。したがって、コンパイラは`a`のが`for`ループでされているがあるとしなければなりません。 `a`のされたは`a`されたバイトからされなければなりません。
- `a`、 `uint32_t`のタイプにはパディングビットはありません。のそのすべてのビットは、ここでは`707406378`のをカウントし、トラップはありません。

オンラインでエイリアシングとなタイプをむ https://riptutorial.com/ja/c/topic/1301/エイリアシングとなタイプ

12: エラー

- #include <errno.h>
- int errno; / ** /
- #include <string.h>
- char * strerrorint errnum;
- #include <stdio.h>
- void perrorconst char * s;

ことをにっているerrnoずしもではなく、は、それがされたがあるかをすだけであること。スレッドインタフェースをつのくのシステムでは、 errnoはのスレッドにしてローカルなオブジェクトにするマクロです。

Examples

エラー

ライブラリがすると、errnoになエラーコードがされることがよくあります。 Cでは、errnoをするためになくとも3つのがです。

EDOM	ドメインエラー
ERANGE	レンジエラー
EILSEQ	なマルチバイトシーケンス

strerror

perrorがでない、 <string.h>からstrerrorをびすことによって、ユーザーがめるエラーをることができます。

```
int main(int argc, char *argv[])
{
    FILE *fout;
    int last_error = 0;

    if ((fout = fopen(argv[1], "w")) == NULL) {
        last_error = errno;
         /* reset errno and continue */
         errno = 0;
    }

    /* do some processing and try opening the file differently, then */


    if (last_error) {
```

```
        fprintf(stderr, "fopen: Could not open %s for writing: %s",
                argv[1], strerror(last_error));
        fputs("Cross fingers and continue", stderr);
    }

    /* do some other processing */

    return EXIT_SUCCESS;
}
```

ユーザーがめるエラーメッセージを`stderr`には、<stdio.h>から`perror`をびします。

```
int main(int argc, char *argv[])
{
   FILE *fout;

   if ((fout = fopen(argv[1], "w")) == NULL) {
      perror("fopen: Could not open file for writing");
      return EXIT_FAILURE;
   }
return EXIT_SUCCESS;
}
```

これにより、 `errno`のにするエラーメッセージがされ`errno`。

オンラインでエラーをむ https://riptutorial.com/ja/c/topic/2486/エラー

13: コマンドライン

- int mainint argc、char * argv []

パラメーター

パラメータ	
argc	count - コマンドラインからプログラムにえられたスペースでられたのとプログラムにされます。
argv	vector - コマンドラインでえられたおよびプログラムをむ`char` -pointersのにされます。

「ホストされた」のタイプ - 「フリースタンド」とはにでするACプログラムは、 `main`をえていなければなりません。にはのようにされています。

```
int main(int argc, char *argv[])
```

`argv`は、 `char **argv`としてすることもできます。はじです。また、パラメータはのローカルであるためできますが、 `argc`と`argv`はのものであり、これらのをするがあります。

コードがをしない`main`については、 `int main(void)`してください。

のパラメータは、プログラムのにされます。

- `argc`は、コマンドラインからプログラムにえられたスペースでられたのとプログラムにされます。
- `argv`は、コマンドラインでえられたおよびプログラムをむ`char` -pointersのです。
- いくつかのシステムはコマンドラインを "シェルので"しますが、のものはしません。 Unixでは、ユーザが`myprogram *.txt`とすると、プログラムはテキストファイルのリストをけります。 Windowsでは " `*.txt` "をけります。

`argv`をするに、 `argc`のをチェックするがあります。には、 `argc`は`0`になり、 `argc`が0の、はなく、`argv[0] argv[argc]`はNULLポインタです。このにした、ホストされたではなシステムになります。に、にしいことですが、プログラムについてのがないこともありえます。その、 `argv[0][0] == '\0'` - プログラムはであるかもしれません。

のようにプログラムをするとします。

```
./some_program abba banana mamajam
```

`argc`は`4`にしく、コマンドラインはのようになります。

- プログラムがホストからである、 `argv[0]`は`"./some_program"` プログラムをします。それのはの`""`します。
- `argv[1]`は`"abba"`します。
- `argv[2]`は`"banana"`します。
- `argv[3]`は`"mamajam"`します。
- `argv[4]`には`NULL`がまれてい`NULL`。

また、からのなについては、cおよびC ++で`main()`すがあるのもてください。

Examples

コマンドラインの

をけったら、のようにそれらをすることができます

```
int main(int argc, char **argv)
{
    for (int i = 1; i < argc; i++)
    {
        printf("Argument %d: [%s]\n", i, argv[i]);
    }
}
```

ノート

1. `argv`パラメータは、 `char *argv[]`としてすることもできます。
2. `argv[0]` は 、プログラムのにじてプログラムがまれることがあります。の "の"コマンドラインは`argv[1]`にあり、これがループ`i`が1にされるです。
3. printでは、 `argv[i]`わりに`*(argv + i)`うことができます。じことがされますが、よりです。
4. のをむかっこは、とをするのにちます。にの、、キャリッジリターン、またはそののながある、これはにです。このプログラムのいくつかのは、リストににまれているものをするがあるにシェルスクリプトをデバッグするのになツールですしかし、ほぼのなシェルのがあります。

をプログラムにし、にする

のコードは、をプログラムにし、コードはをにしようとします `long`。

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <limits.h>

int main(int argc, char* argv[]) {

    for (int i = 1; i < argc; i++) {
```

```
        printf("Argument %d is: %s\n", i, argv[i]);

        errno = 0;
        char *p;
        long argument_numValue = strtol(argv[i], &p, 10);

        if (p == argv[i]) {
            fprintf(stderr, "Argument %d is not a number.\n", i);
        }
        else if ((argument_numValue == LONG_MIN || argument_numValue == LONG_MAX) && errno ==
ERANGE) {
            fprintf(stderr, "Argument %d is out of range.\n", i);
        }
        else {
            printf("Argument %d is a number, and the value is: %ld\n",
                   i, argument_numValue);
        }
    }
    return 0;
}
```

- strtolはなをします
- strtolのしいい

GNU getoptツールをう

アプリケーションのコマンドラインオプションは、Cによるコマンドラインとはなるいをけません。それらは、LinuxまたはUnixではにダッシュ - でまるです。

LinuxやUnixでglibcをうと、 getoptツールをってりのからコマンドラインオプションをに、、することができます。

これらのツールは、オプションがGNUコーディングにってフォーマットされることをしています。これはPOSIXがコマンドラインオプションののためにしたもののです。

のは、GNU getoptツールをしてコマンドラインオプションをするをしています。

```
#include <stdio.h>
#include <getopt.h>
#include <string.h>

/* print a description of all supported options */
void usage (FILE *fp, const char *path)
{
    /* take only the last portion of the path */
    const char *basename = strrchr(path, '/');
    basename = basename ? basename + 1 : path;

    fprintf (fp, "usage: %s [OPTION]\n", basename);
    fprintf (fp, "  -h, --help\t\t"
                 "Print this help and exit.\n");
    fprintf (fp, "  -f, --file[=FILENAME]\t"
                 "Write all output to a file (defaults to out.txt).\n");
    fprintf (fp, "  -m, --msg=STRING\t"
                 "Output a particular message rather than 'Hello world'.\n");
```

```
}

/* parse command-line options and print message */
int main(int argc, char *argv[])
{
    /* for code brevity this example just uses fixed buffer sizes for strings */
    char filename[256] = { 0 };
    char message[256] = "Hello world";
    FILE *fp;
    int help_flag = 0;
    int opt;

    /* table of all supported options in their long form.
     * fields: name, has_arg, flag, val
     * `has_arg` specifies whether the associated long-form option can (or, in
     * some cases, must) have an argument. the valid values for `has_arg` are
     * `no_argument`, `optional_argument`, and `required_argument`.
     * if `flag` points to a variable, then the variable will be given a value
     * of `val` when the associated long-form option is present at the command
     * line.
     * if `flag` is NULL, then `val` is returned by `getopt_long` (see below)
     * when the associated long-form option is found amongst the command-line
     * arguments.
     */
    struct option longopts[] = {
        { "help", no_argument, &help_flag, 1 },
        { "file", optional_argument, NULL, 'f' },
        { "msg", required_argument, NULL, 'm' },
        { 0 }
    };

    /* infinite loop, to be broken when we are done parsing options */
    while (1) {
        /* getopt_long supports GNU-style full-word "long" options in addition
         * to the single-character "short" options which are supported by
         * getopt.
         * the third argument is a collection of supported short-form options.
         * these do not necessarily have to correlate to the long-form options.
         * one colon after an option indicates that it has an argument, two
         * indicates that the argument is optional. order is unimportant.
         */
        opt = getopt_long (argc, argv, "hf::m:", longopts, 0);

        if (opt == -1) {
            /* a return value of -1 indicates that there are no more options */
            break;
        }

        switch (opt) {
        case 'h':
            /* the help_flag and value are specified in the longopts table,
             * which means that when the --help option is specified (in its long
             * form), the help_flag variable will be automatically set.
             * however, the parser for short-form options does not support the
             * automatic setting of flags, so we still need this code to set the
             * help_flag manually when the -h option is specified.
             */
            help_flag = 1;
            break;
        case 'f':
            /* optarg is a global variable in getopt.h. it contains the argument
```

```
             * for this option. it is null if there was no argument.
             */
            printf ("outarg: '%s'\n", optarg);
            strncpy (filename, optarg ? optarg : "out.txt", sizeof (filename));
            /* strncpy does not fully guarantee null-termination */
            filename[sizeof (filename) - 1] = '\0';
            break;
        case 'm':
            /* since the argument for this option is required, getopt guarantees
             * that aptarg is non-null.
             */
            strncpy (message, optarg, sizeof (message));
            message[sizeof (message) - 1] = '\0';
            break;
        case '?':
            /* a return value of '?' indicates that an option was malformed.
             * this could mean that an unrecognized option was given, or that an
             * option which requires an argument did not include an argument.
             */
            usage (stderr, argv[0]);
            return 1;
        default:
            break;
        }
    }

    if (help_flag) {
        usage (stdout, argv[0]);
        return 0;
    }

    if (filename[0]) {
        fp = fopen (filename, "w");
    } else {
        fp = stdout;
    }

    if (!fp) {
        fprintf(stderr, "Failed to open file.\n");
        return 1;
    }

    fprintf (fp, "%s\n", message);

    fclose (fp);
    return 0;
}
```

これは`gcc`でコンパイルすることができます

```
gcc example.c -o example
```

3つのコマンドラインオプション `--help` 、 -- `--file` 、 -- `--msg` を`--msg`ます。すべてが「い」 `-h` 、 `-f` 、および`-m` もっています。 "file"と "msg"オプションはどちらもをけれます。 "msg"オプションをした、そのはです。

オプションのはのようにされます。

- `--option=value` いのオプションの
- `-ovalue -o"value"`または`-o"value"` オプションの

オンラインでコマンドラインをむ https://riptutorial.com/ja/c/topic/1285/コマンドライン

14: コメント

き

コメントは、コードをむにかをすためにされます。コメントはコンパイラによってとしてわれ、コードののでもされません。 Cのコメントには2つのがあり、の/* */としい//です。いくつかのドキュメントシステムでは、なのコメントをしてコードのドキュメンテーションをしています。

- /*...*/
- //... C99のみ

Examples

/**/られたコメント

コメントはスラッシュ、アスタリスク /* でまり、アスタリスクのにスラッシュ */ がくとすぐにします。これらののみわせののすべてがコメントであり、コンパイラによってにされるとしてわれます。

```
/* this is a comment */
```

のコメントはコメントです。この/*のコメントは、のようにのにまたがることができます

```
/* this is a
multi-line
comment */
```

にはではありませんが、のコメントをするなスタイルでは、のにくとアスタリスクをしいに/*と*/をけてべます。

```
/*
 * this is a
 * multi-line
 * comment
 */
```

なアスタリスクは、コメントになをぼさず、するスラッシュはありません。

これらの/*タイプのコメントは、それぞれの、コードの、またはコードですることができます。

```
/* this comment is on its own line */
if (x && y) { /*this comment is at the end of a line */
    if ((complexCondition1) /* this comment is within a line of code */
            && (complexCondition2)) {
        /* this comment is within an if, on its own line */
    }
```

```
}
```

コメントはネストできません。これは、そのの`/*`はコメントのとしてされ、の`*/`がコメントのとしてわれるためです。ののコメントはしません 。

```
/* outer comment, means this is ignored => /* attempted inner comment */ <= ends the comment,
not this one => */
```

このタイプのコメントをむコードのブロックをコメントするには、そうでなければネストされます。の「プリプロセッサをしたコメントの」をしてください。

//られたコメント

C99

C99では、C ++スタイルのコメントをしました。このタイプのコメントは2つのスラッシュでまり、までされます。

```
// this is a comment
```

このタイプのコメントでは、のコメントはされませんが、いくつかの1のコメントを1つずつしてコメントブロックをすることはです。

```
// each of these lines are a single-line comment
// note how each must start with
// the double forward-slash
```

このタイプのコメントは、それのまたはコードのにできます。ただし、 のまでされるため、コードでされないがあります

```
// this comment is on its own line
if (x && y) { // this comment is at the end of a line
    // this comment is within an if, on its own line
}
```

プリプロセッサをったコメント

のコードは、プリプロセッサディレクティブ`#if 0`と`#endif`をして "コメントアウト"することもできます。これは、コードにのコメントがまれていて、それのにはネストされないにです。

```
#if 0 /* Starts the "comment", anything from here on is removed by preprocessor */

/* A large amount of code with multi-line comments */
int foo()
{
    /* lots of code */
    ...

    /* ... some comment describing the if statement ... */
```

```
    if (someTest) {
        /* some more comments */
        return 1;
    }

    return 0;
}

#endif /* 0 */

/* code from here on is "uncommented" (included in compiled executable) */
...
```

トリグラフによるとし

C99

いている//りのコメントを、らのされるにをぼすをうことがです。 1つのタイプの

```
int x = 20;  // Why did I do this??/
```

の/はtypoでしたが、 \にされます。これは??/がをするためです。

??/ trigraphはにはである\さです。これは、のがののき、つまりコメントのきであるとコンパイラがえていることをします。

```
int foo = 20; // Start at 20 ??/
int bar = 0;

// The following will cause a compilation error (undeclared variable 'bar')
// because 'int bar = 0;' is part of the comment on the preceding line
bar += foo;
```

オンラインでコメントをむ https://riptutorial.com/ja/c/topic/10670/コメント

15: シーケンスポイント

ISO / IEC 9899201xプログラミング **- C**

オブジェクトにアクセスする、オブジェクトをする、ファイルをする、またはこれらののいずれかをうをびすことは、ののであるです。

AとBののののシーケンスポイントのは、Aにするすべてののおよびが、Bにするすべてののおよびのにけられることをする。

Cのオンライン2011プレリリースCのシーケンスポイントのなリストをにします 。

シーケンスポイント

1 5.1.2.3でしたシーケンスポイントはのとおりです。

- ファンクションコールののとファンクションデジグネータのの。 6.5.2.2。
- のの1オペランドと2オペランドののAND `&&` 6.5.13;OR `||` 6.5.14;コンマ , 6.5.17 。
- きののオペランドのの `?` :2オペランドと3オペランドのどちらかがされる6.5.15 。
- なのわり6.7.6;
- なのとされるのなの。はなですリテラル6.7.9のではない。6.8.3の。の `if`または `switch` 6.8.4; `while`または `do`の6.8.5。 `for`6.8.5.3のオプションの。 `return`6.8.6.4のオプションの。
- ライブラリがる7.1.4。
- された7.21.6,7.29.2にけられたアクションの。
- へのびしのおよび、およびへのびしと、そのびしにとしてされたオブジェクトの7.22.5とのにもします。

Examples

シーケンス

のがけられています 。

```
a && b
a || b
a , b
a ? b : c
for ( a ; b ; c ) { ... }
```

すべてのにおいて、 `a`はにされ、 `b`または `c`いずれかがされるにすべてのがされます。 4のケースで

は、 bまたはc 1つのみがされます。のケースでは、 bがにされ、 cがされるにすべてのがされます。

てのにおいて、のaののにされbはc のbびcにされています。 a

したがって、

```
x++ && x++
x++ ? x++ : y++
(x = f()) && x != 0
for ( x = 0; x < 10; x++ ) { ... }
y = (x++, x++);
```

よくされたるいをつ。

けられていない

C11

のはけされていません 。

```
a + b;
a - b;
a * b;
a / b;
a % b;
a & b;
a | b;
```

のでは、のにされるか、またはにa b 、 bにすることができる、またはそれらは、いくつかのにする、それらはさらにされてもよいです。 a

びしについてものがされます。

```
f(a, b);
```

ここでないだけとa b unsequencedすなわちされ,びしのオペレータがシーケンスポイントをしない だけでなく、 f 、びされるをする。

は、のにされてもよく、のまでされてもよい。

のような

```
x++ & x++;
f(x++, x++); /* the ',' in a function call is *not* the same as the comma operator */
x++ * x++;
a[i] = i++;
```

または

```
x++ & x;
f(x++, x);
x++ * x;
a[i++] = i;
```

のをこす

- オブジェクトのおよびそれにするのアクセスは、けされなければならない
- 1がされるおよびはされていない。

1のの。

にされた

f(a)ようなびしは、にのとここではfとa とのびしののシーケンスポイントをします。 2つのびしがけされていない、2つのびしはにけられています。 つまり、がのにされ、がされていません。

```
unsigned counter = 0;

unsingned account(void) {
   return counter++;
}

int main(void) {
   printf("the order is %u %u\n", account(), account());
}
```

printfのにこのに2のcountercounterことはです。どのびしがにるのかかりません。はであるため、それはわるがあり、することはできません。したがって、プリントアウトはのようになります。

は0 1です

または

は1 0です

びしをせずにとのステートメント

```
   printf("the order is %u %u\n", counter++, counter++); // undefined behavior
```

counter 2つののにシーケンスポイントがないため、のがあります。

オンラインでシーケンスポイントをむ https://riptutorial.com/ja/c/topic/1275/シーケンスポイント

16: ジャンプ

- り。 / *のからります。 valは、のりのにされるののにすることができます。 * /
- る; / *のvoidからります。 * /
- ブレーク; / *ステートメントループのわりをえてにびしたり、ものswitchからびしたりします。 * /
- する; / *ステートメントループのににジャンプします。 * /
- LBLにく。 / *ラベルLBLにジャンプします。 * /
- LBL *statement* / *じののステートメント。 * /

これらはキーワードでCにされたジャンプです。

Cには、データ、 `jmp_buf` 、およびCライブラリびし`setjmp`と`longjmp`されたのジャンプ*long jump*があります。

もしてください

ステートメント/ループfor、while、do-while

Examples

gotoをしてれになったループからびす

れになったループからびすには、、ブールをし、ループでこのをチェックするがあります。たちが`i`と`j`しているとすれば、このようにえるかもしれません

```
size_t i,j;
for (i = 0; i < myValue && !breakout_condition; ++i) {
    for (j = 0; j < mySecondValue && !breakout_condition; ++j) {
        ... /* Do something, maybe modifying breakout_condition */
            /* When breakout_condition == true the loops end */
     }
}
```

しかし、Cでは`goto`がされていますが、このはです。ループのにされたラベルをすることで、ループからにりすことができます。

```
size_t i,j;
for (i = 0; i < myValue; ++i) {
    for (j = 0; j < mySecondValue; ++j) {
        ...
        if(breakout_condition)
          goto final;
    }
}
final:
```

しかし、しばしばこのがれた`return`、 `return`わりにするほうがよいでしょう。このは、プログラミングにおいても「」とえられている。

`goto`がにつのは、エラーハンドラにジャンプすることです。

```
ptr = malloc(N *  x);
if(!ptr)
  goto out_of_memory;

/* normal processing */
free(ptr);
return SUCCESS;

out_of_memory:
 free(ptr); /* harmless, and necessary if we have further errors */
 return FAILURE;
```

`goto`をすると、のプログラムフローとはのエラーフローがされます。しかし、それはまた、なにおいて「」とえられている。

リターンをう

をす

にされるケースの1つ `main()`

```
#include <stdlib.h> /* for EXIT_xxx macros */

int main(int argc, char ** argv)
{
  if (2 < argc)
  {
    return EXIT_FAILURE; /* The code expects one argument:
                            leave immediately skipping the rest of the function's code */
  }

  /* Do stuff. */

  return EXIT_SUCCESS;
}
```

そのの

1. りのが`void` `void *`やするをまないのの、 `return`にはするがまれてはいけません。つまり、されるreturnは`return;`。
2. `void`りをつの、 `return`はなしではしてはならない。
3. のために`main()`とのみのため`main()` な `return`はC99にとされていません。がにした`}`、の`0`がされます。この`return`する`return`はいだとうもいます。のはにそれをすることをする。

もさない

`void`からの`void`

```
void log(const char * message_to_log)
{
  if (NULL == message_to_log)
  {
    return; /* Nothing to log, go home NOW, skip the logging. */
  }

  fprintf(stderr, "%s:%d %s\n", __FILE__, _LINE__, message_to_log);

  return; /* Optional, as this function does not return a value. */
}
```

ブレ―クとをする

すぐに`continue`なにみりまたは`break`ユ―ザのやファイルのわりに

```
#include <stdlib.h> /* for EXIT_xxx macros */
#include <stdio.h>  /* for printf() and getchar() */
#include <ctype.h> /* for isdigit() */

void flush_input_stream(FILE * fp);


int main(void)
{
  int sum = 0;
  printf("Enter digits to be summed up or 0 to exit:\n");

  do
  {
    int c = getchar();
    if (EOF == c)
    {
      printf("Read 'end-of-file', exiting!\n");

      break;
    }

    if ('\n' != c)
    {
      flush_input_stream(stdin);
    }

    if (!isdigit(c))
    {
      printf("%c is not a digit! Start over!\n", c);

      continue;
    }

    if ('0' == c)
    {
```

```
      printf("Exit requested.\n");

      break;
    }

    sum += c - '0';

    printf("The current sum is %d.\n", sum);
  } while (1);

  return EXIT_SUCCESS;
}

void flush_input_stream(FILE * fp)
{
  size_t i = 0;
  int c;
  while ((c = fgetc(fp)) != '\n' && c != EOF) /* Pull all until and including the next new-
line. */
  {
    ++i;
  }

  if (0 != i)
  {
    fprintf(stderr, "Flushed %zu characters from input.\n", i);
  }
}
```

オンラインでジャンプをむ https://riptutorial.com/ja/c/topic/5568/ジャンプ

17: ストレージクラス

き

クラスは、またはのスコープをするためにされます。のストレージクラスをることで、プログラムのにそののをできます。

- [auto | register | static | extern] <データ> <> [= <>];
- [_Thread_local | extern _Thread_local | _Thread_local] <データ> <> [= <>]; / *>= C11 * /
-
- typedef int *foo* ;
- extern int *foo* [2];

ストレージクラスは、のトップレベルのにされるキーワードです。これらのキーワードをすると、ファイルスコープでされているかブロックスコープでされているかによって、されたオブジェクトのとリンケージにします。

キーワード		リンケージ	
`static`			ファイルスコープのオブジェクトのリンケージをします。ブロックスコープのオブジェクトのをします。
`extern`			したがって、をつファイルスコープでされたオブジェクトにはのため、しています。イニシャライザなしでファイルスコープのでされる、そのがのでされ、リンクにされることをします。
`auto`			ブロックスコープでされたオブジェクトにはされ、したがってがあります。
`register`			をつオブジェクトにのみします。がレジスタにされるべきであるというヒントをします。されるは1つができないことがある& 、そのようなのオペレータ「のアドレス」、したがってオブジェクトがエイリアスすることができません。
`typedef`			にはストレージクラスではありませんが、なからはじようにします。のいは、されたがオブジェクトではなくであることです。
`_Thread_local`		/	C11でされ、スレッドをします。ブロックスコープでされるは、`extern`または`static`もみ`static`。

これらのキーワードがされているでも、すべてのオブジェクトにはするスコープになくとリンケージファイルスコープでのにのみするがあります。

トップレベル `int` 、 `unsigned` 、 `short`などおよびトップレベル `const` 、 `volatile` にするストレージクラスのはされないため、これらのはともです。

```
int static const unsigned a = 5; /* bad practice */
static const unsigned int b = 5; /* good practice */
```

しかし、にクラスをいてからをし、 `void` 、 `char` 、 `int` 、 `signed long` 、 `unsigned long long` 、 `long double` ...をくことをお`char`ます。

すべてのストレージクラスがのスコープでであるとはりません。

```
register int x; /* legal at block scope, illegal at file scope */
auto int y; /* same */

static int z; /* legal at both file and block scope */
extern int a; /* same */

extern int b = 5; /* legal and redundant at file scope, illegal at block scope */

/* legal because typedef is treated like a storage class specifier syntactically */
int typedef new_type_name;
```

はまたはのいずれかです。されたオブジェクトの、そのスコープとストレージクラスによってされます。

ストレージ

なをつは、プログラムのをしてきており、ファイルスコープ `static`またはしとブロックスコープに`static`にくのでることができます。これらは、プログラムのにオペレーティングシステムによってりてられ、され、プロセスがするとされます。には、にはそのような `data` 、 `bss` 、および `rodata` のセクションがあり、ファイルのこれらのセクションがのでメモリにマップされます。

スレッドの

C11

このはC11でされました。これはのCではできませんでした。コンパイラのには、のセマンティクスをつをするものがあります。えば、gccがサポート`__thread`っていなかったのCでな `_Thread_local`。

スレッドをつは、ファイルスコープとブロックスコープのでできます。ブロックスコープでされているは、 `static`または`extern`ストレージもします。そのは、それがされたスレッドのです。これは、のストレージとにできるのストレージです。

をつは、ブロックスコープでまたはそののブロックですることができます。これらは、ファンクションまたはブロックのからまでのにのみできます。がスコープからられるとからるかブロックをれることによって、そのストレージはにりてがされます。ポインタからのじへのそれのはであり、のにつながります。

なでは、は、のスタックフレームまたはレジスタののオフセットにされます。

リンクとリンク

リンケージはファイルスコープでされたオブジェクトとにのみし、なるのにします。リンケージをつオブジェクトは、のすべてのでされますながまれている。リンケージをつオブジェクトはのにはされず、されているでのみできます。

Examples

typedef

のについてしいをします。そのは、のそれをしています。

```
/* Byte can be used wherever `unsigned char` is needed */
typedef unsigned char Byte;

/* Integer is the type used to declare an array consisting of a single int */
typedef int Integer[1];

/* NodeRef is a type used for pointers to a structure type with the tag "node" */
typedef struct node *NodeRef;

/* SigHandler is the function pointer type that gets passed to the signal function. */
typedef void (*SigHandler)(int);
```

にはストレージクラスではありませんが、 `typedef`キーワードがされている、のストレージクラスはされていないため、コンパイラはこれを1と`typedef`。

`typedef`はであり、 `#define`マクロできえるべきではありません。

```
typedef int newType;
newType *ptr;        // ptr is pointer to variable of type 'newType' aka int
```

しかしながら、

```
#define int newType
newType *ptr;        // Even though macros are exact replacements to words, this doesn't
result to a pointer to variable of type 'newType' aka int
```

オート

このクラスは、がをすることをす。つまり、がされたスコープがすると、でされるオブジェクトはもはやではなくなります。

グローバルスコープや`static`staticれていないすべてのオブジェクトは、にデフォルトでがあるため、このキーワードはにであり、しないでください。

```
int foo(void)
{
    /* An integer with automatic storage duration. */
    auto int i = 3;

    /* Same */
    int j = 5;

    return 0;
} /* The values of i and j are no longer able to be used. */
```

`static`ストレージ・クラスは、ファイルののによってなるをたします。

1. をそののみにするにはscope = file。

   ```
   /* No other translation unit can use this variable. */
   static int i;

   /* Same; static is attached to the function type of f, not the return type int. */
   static int f(int n);
   ```

2. ののびしでするためにデータをするにはscope = block

   ```
   void foo()
   {
       static int a = 0; /* has static storage duration and its lifetime is the
                          * entire execution of the program; initialized to 0 on
                          * first function call */
       int b = 0; /* b has block scope and has automatic storage duration and
                   * only "exists" within function */

       a += 10;
       b += 10;

       printf("static int a = %d, int b = %d\n", a, b);
   }

   int main(void)
   {
       int i;
       for (i = 0; i < 5; i++)
       {
           foo();
       }

       return 0;
   }
   ```

   このコードはをします

```
 static int a = 10, int b = 10
 static int a = 20, int b = 10
 static int a = 30, int b = 10
 static int a = 40, int b = 10
 static int a = 50, int b = 10
```

は、のなるスレッドからびされたでもそのをします。

C99

3. をすためにのパラメ―タでされるは、のとnullパラメ―タがであることがされます。

```
/* a is expected to have at least 512 elements. */
void printInts(int a[static 512])
{
    size_t i;
    for (i = 0; i < 512; ++i)
        printf("%d\n", a[i]);
}
```

なまたはnullのポインタさえもは、コンパイラによってずしもチェックされるわけではなく、ながない、コンパイラはらかのでするはありません。プログラマが512のまたはヌルポインタをす、のがになります。これをすることはなので、そのようなにそのパラメ―タ―のをすときは、ながです。

extern

のでされているそしてのリンケ―ジをつ オブジェクト またはをするためにされます。に、するオブジェクトまたはがされていないモジュ―ルでされるオブジェクトまたはをするためにされます。

```
/* file1.c */
int foo = 2;  /* Has external linkage since it is declared at file scope. */
```

```
/* file2.c */
#include <stdio.h>
int main(void)
{
    /* `extern` keyword refers to external definition of `foo`. */
    extern int foo;
    printf("%d\n", foo);
    return 0;
}
```

C99

C99の`inline`キ―ワ―ドのにより、しいことがこります。

```
/* Should usually be place in a header file such that all users see the definition */
/* Hints to the compiler that the function `bar` might be inlined */
/* and suppresses the generation of an external symbol, unless stated otherwise. */
inline void bar(int drink)
```

```
{
    printf("You ordered drink no.%d\n", drink);
}

/* To be found in just one .c file.
   Creates an external function definition of `bar` for use by other files.
   The compiler is allowed to choose between the inline version and the external
   definition when `bar` is called. Without this line, `bar` would only be an inline
   function, and other files would not be able to call it. */
extern void bar(int);
```

オブジェクトへのアクセスはなりでなければならないことをコンパイラにらせる。コンパイラがにヒントをするかどうかは、です。にそれを`auto`ととしてうかもしれません。

`register`で`register`ているすべてのオブジェクトでは、なるプロパティがあるため、アドレスはできません。これにより、 `register`はのをするためのれたツ―ルになります。

```
register size_t size = 467;
```

せずされるのあるのにコ―ドをすことができないため、 エィリアスにならないオブジェクトです。

このプロパティは、

```
register int array[5];
```

そののへのポインタにはできません `array` `&array[0]`わり`&array[0]`。 これは、そのようなのにアクセスすることができず、をにすことができないことをします。

、 `register`クラスでされたののなは、 `sizeof`です。のはののののアドレスをとします。そのため、のサイズはにたないので、 `register`キ―ワ―ドでするのははではありません。これは`register`キ―ワ―ドなしでもにうことができます。

`register`クラスは、ブロックでされ、いでアクセスされるにしています。えば、

```
/* prints the sum of the first 5 integers*/
/* code assumed to be part of a function body*/
{
    register int k, sum;
    for(k = 1, sum = 0; k < 6; sum += k, k++);
        printf("\t%d\n",sum);
}
```

C11

`_Alignof`は、 `register`でもできます。

_Thread_local

C11

これはC11でされたしいで、マルチスレッドされました。これはのCではできません。

スレッドのをします。 `_Thread_local`でされたは、そのオブジェクトがそのスレッドにしてローカルであり、そのが、そのスレッドがされたスレッドのであることをします。 `static`または`extern`とにすることもできます。

```
#include <threads.h>
#include <stdio.h>
#define SIZE 5

int thread_func(void *id)
{
    /* thread local variable i. */
    static _Thread_local int i;

    /* Prints the ID passed from main() and the address of the i.
     * Running this program will print different addresses for i, showing
     * that they are all distinct objects. */
    printf("From thread:[%d], Address of i (thread local): %p\n", *(int*)id, (void*)&i);

    return 0;
}

int main(void)
{
    thrd_t id[SIZE];
    int arr[SIZE] = {1, 2, 3, 4, 5};

    /* create 5 threads. */
    for(int i = 0; i < SIZE; i++) {
        thrd_create(&id[i], thread_func, &arr[i]);
    }

    /* wait for threads to complete. */
    for(int i = 0; i < SIZE; i++) {
        thrd_join(id[i], NULL);
    }
}
```

オンラインでストレージクラスをむ https://riptutorial.com/ja/c/topic/3597/ストレージクラス

18: スレッドネイティブ

- `#ifndef __STDC_NO_THREADS__`
- `# include <threads.h>`
- `#endif`
- `void call_once(once_flag *flag, void (*func)(void));`
- `int cnd_broadcast(cnd_t *cond);`
- `void cnd_destroy(cnd_t *cond);`
- `int cnd_init(cnd_t *cond);`
- `int cnd_signal(cnd_t *cond);`
- `int cnd_timedwait(cnd_t *restrict cond, mtx_t *restrict mtx, const struct timespec *restrict ts);`
- `int cnd_wait(cnd_t *cond, mtx_t *mtx);`
- `void mtx_destroy(mtx_t *mtx);`
- `int mtx_init(mtx_t *mtx, int type);`
- `int mtx_lock(mtx_t *mtx);`
- `int mtx_timedlock(mtx_t *restrict mtx, const struct timespec *restrict ts);`
- `int mtx_trylock(mtx_t *mtx);`
- `int mtx_unlock(mtx_t *mtx);`
- `int thrd_create(thrd_t *thr, thrd_start_t func, void *arg);`
- `thrd_t thrd_current(void);`
- `int thrd_detach(thrd_t thr);`
- `int thrd_equal(thrd_t thr0, thrd_t thr1);`
- `_Noreturn void thrd_exit(int res);`
- `int thrd_join(thrd_t thr, int *res);`
- `int thrd_sleep(const struct timespec *duration, struct timespec* remaining);`
- `void thrd_yield(void);`
- `int tss_create(tss_t *key, tss_dtor_t dtor);`
- `void tss_delete(tss_t key);`
- `void *tss_get(tss_t key);`
- `int tss_set(tss_t key, void *val);`

C11スレッドはオプションです。それらのは`__STDC__NO_THREAD__`でテストできます。20167、このはC11をサポートしていないすべてのCライブラリではまだされていません。

C11スレッドをサポートすることがられているCライブラリはのとおりです。

- モスル

C11スレッドをサポートしていないCライブラリ

- GNU libc

Examples

のスレッドをする

```
#include <stdio.h>
#include <threads.h>
#include <stdlib.h>

struct my_thread_data {
   double factor;
};

int my_thread_func(void* a) {
   struct my_thread_data* d = a;
   // do something with d
   printf("we found %g\n", d->factor);
   // return an success or error code
   return d->factor > 1.0;
}


int main(int argc, char* argv[argc+1]) {
    unsigned n = 4;
    if (argc > 1) n = strtoull(argv[1], 0, 0);
    // reserve space for the arguments for the threads
    struct my_thread_data D[n];     // can't be initialized
    for (unsigned i = 0; i < n; ++i) {
         D[i] = (struct my_thread_data){ .factor = 0.5*i, };
    }
    // reserve space for the ID's of the threads
    thrd_t id[4];
    // launch the threads
    for (unsigned i = 0; i < n; ++i) {
         thrd_create(&id[i], my_thread_func, &D[i]);
    }
    // Wait that all threads have finished, but throw away their
    // return values
    for (unsigned i = 0; i < n; ++i) {
         thrd_join(id[i], 0);
    }
    return EXIT_SUCCESS;
}
```

1つのスレッドによる

くの、のスレッドによってアクセスされるすべてのデータは、スレッドがされるにされるがあります。これにより、すべてのスレッドがクリアでし、 はしません。

これがなは、 `once_flag`と`call_once`をできます

```
#include <threads.h>
#include <stdlib.h>

// the user data for this example
double const* Big = 0;

// the flag to protect big, must be global and/or static
static once_flag onceBig = ONCE_INIT;

void destroyBig(void) {
   free((void*)Big);
}
```

```
void initBig(void) {
    // assign to temporary with no const qualification
    double* b = malloc(largeNum);
    if (!b) {
       perror("allocation failed for Big");
       exit(EXIT_FAILURE);
    }
    // now initialize and store Big
    initializeBigWithSophisticatedValues(largeNum, b);
    Big = b;
    // ensure that the space is freed on exit or quick_exit
    atexit(destroyBig);
    at_quick_exit(destroyBig);
}

// the user thread function that relies on Big
int myThreadFunc(void* a) {
   call_once(&onceBig, initBig);
   // only use Big from here on
   ...
   return 0;
}
```

once_flagは、じデータBigをするのあるなるスレッドをするためにされます。 call_onceへのびしは、

- initBigはちょうど1びされます
- call_onceは、 initBigへのそのようなびしがじスレッドまたはのスレッドによってわれるまでブロックします。

りてのほか、えばびされたにうためのなものは、のようなスレッドデータをにされmtx_tはcnd_tして、にすることができないmtx_initはcnd_initそれぞれ、。

オンラインでスレッドネイティブをむ https://riptutorial.com/ja/c/topic/4432/スレッド-ネイティブ-

19: データ

- `char`は1バイトであるがありますが、ほとんどののコンピュータプラットフォームでは8ビットとされていますが、1バイトは8ビットはオクテットともばれますであるはありません 。の`char`あたりのビットは、 `<limits.h>`でされている`CHAR_BIT`マクロによってされます。POSIXでは、1バイトが8ビットであるがあります。
- のはまばらにするがあります。Cのみみはすべてのアーキテクチャでにされています。のは、ネットワーキングなどのなサイズのがになにのみしてください。

Examples

と

されたは、これらののものであるがあります `short`の`int`または`long`はオプションです。

```
signed char c = 127; /* required to be 1 byte, see remarks for further information. */
signed short int si = 32767; /* required to be at least 16 bits. */
signed int i = 32767; /* required to be at least 16 bits */
signed long int li = 2147483647; /* required to be at least 32 bits. */
```

C99

```
signed long long int li = 2147483647; /* required to be at least 64 bits */
```

これらのきのそれぞれは、なしバージョンをちます。

```
unsigned int i = 65535;
unsigned short = 2767;
unsigned char = 255;
```

しかし、すべてのタイプの`char` `signed`たのバージョンがされる`signed`または`unsigned`がされています。 `char`は`signed char`と`unsigned char`とはなる3のをし、はプラットフォームによってなります。

さまざまなタイプのCジャーゴンのリテラルとばれます は、それぞれのまたはにづいて、なるベースとなるできむことができます。

```
/* the following variables are initialized to the same value: */
int d = 42;   /* decimal constant (base10) */
int o = 052;  /* octal constant (base8) */
int x = 0xaf; /* hexadecimal constants (base16) */
int X = 0XAf; /* (letters 'a' through 'f' (case insensitive) represent 10 through 15) */
```

はに`signed`ます。 16は`0x`または`0X`でまり、8は`0`まり`0` 。の2つは、が`unsigned`きのにまるかどうかによって`unsigned` `signed`または`unsigned`です。

```
/* suffixes to describe width and signedness : */
```

```
long int i = 0x32; /* no suffix represent int, or long int */
unsigned int ui = 65535u; /* u or U represent unsigned int, or long int */
long int li = 655361; /* l or L represent long int */
```

がなければ、はそのにするのをちます。つまり、 INT_MAXはなはlong、それのはlong longよりきい10です。

ヘッダーファイル<limits.h>は、のようにのがされています。それらのは、にすものとじかそれのきさでなければならない。

マクロ	タイプ	
CHAR_BIT	ビットフィールドではないのオブジェクトバイト	8
SCHAR_MIN	signed char	-127 / - 2 7 -1
SCHAR_MAX	signed char	+127 / 2 7 - 1
UCHAR_MAX	unsigned char	255/2 8-1
CHAR_MIN	char	
CHAR_MAX	char	
SHRT_MIN	short int	-32767 / - 215 - 1
SHRT_MAX	short int	+32767 / 2 15から 1
USHRT_MAX	unsigned short int	2の65535 16から 1
INT_MIN	int	-32767 / - 215 - 1
INT_MAX	int	+32767 / 2 15から 1
UINT_MAX	unsigned int	2の65535 16から 1
LONG_MIN	long int	-2147483647 / - 2 31 - 1
LONG_MAX	long int	+2147483647 / 2 31 - 1
ULONG_MAX	unsigned long int	4294967295/2 32-1

C99

マクロ	タイプ	
LLONG_MIN	long long int	-9223372036854775807 / - 263 - 1
LLONG_MAX	long long int	92233720368547758072 63から 1
ULLONG_MAX	unsigned long long int	2の18446744073709551615 64から 1

もしのcharでされる、のCHAR_MINとじでなければならないSCHAR_MINのCHAR_MAXとじでなければならないSCHAR_MAX。 のオブジェクトのcharないでされる、のCHAR_MIN 0でなければならないとのCHAR_MAXとじでなければならないUCHAR_MAX。

C99

C99には、ののをむしいヘッダー—<stdint.h>されました。よりなについては、のをしてください。

リテラル

Cのリテラルは、リテラルゼロでわるです。

```
char* str = "hello, world"; /* string literal */

/* string literals can be used to initialize arrays */
char a1[] = "abc"; /* a1 is char[4] holding {'a','b','c','\0'} */
char a2[4] = "abc"; /* same as a1 */
char a3[3] = "abc"; /* a1 is char[3] holding {'a','b','c'}, missing the '\0' */
```

リテラルはできません には.rodataなどのみりメモリにくことができます。をしようとすると、のがします。

```
char* s = "foobar";
s[0] = 'F'; /* undefined behaviour */

/* it's good practice to denote string literals as such, by using `const` */
char const* s1 = "foobar";
s1[0] = 'F'; /* compiler error! */
```

のリテラルはコンパイルにされます。つまり、これらのようなをくことができます。

C99

```
/* only two narrow or two wide string literals may be concatenated */
char* s = "Hello, " "World";
```

C99

```
/* since C99, more than two can be concatenated */
/* concatenation is implementation defined */
char* s1 = "Hello" ", " "World";

/* common usages are concatenations of format strings */
char* fmt = "%" PRId16; /* PRId16 macro since C99 */
```

とじリテラルは、なるセットをサポートします。

```
/* normal string literal, of type char[] */
char* s1 = "abc";

/* wide character string literal, of type wchar_t[] */
wchar_t* s2 = L"abc";
```

C11

```
/* UTF-8 string literal, of type char[] */
char* s3 = u8"abc";

/* 16-bit wide string literal, of type char16_t[] */
char16_t* s4 = u"abc";

/* 32-bit wide string literal, of type char32_t[] */
char32_t* s5 = U"abc";
```

C99

C99

ヘッダ<stdint.h>はいくつかののをします。これらのはオプションであり、プラットフォームがするのをち、するきがのの2のをつにのみされます。

タイプのヒントについては、をしてください。

```
/* commonly used types include */
uint32_t u32 = 32; /* exactly 32-bits wide */

uint8_t u8 = 255;  /* exactly 8-bits wide */

int64_t i64 = -65  /* exactly 64 bit in two's complement representation */
```

Cは、3つのの、っている`float`、 `double`、および`long double`。

```
float f = 0.314f;        /* suffix f or F denotes type float */
double d = 0.314;        /* no suffix denotes double */
long double ld = 0.3141; /* suffix l or L denotes long double */

/* the different parts of a floating point definition are optional */
double x = 1.; /* valid, fractional part is optional */
double y = .1; /* valid, whole-number part is optional */

/* they can also defined in scientific notation */
double sd = 1.2e3; /* decimal fraction 1.2 is scaled by 10^3, that is 1200.0 */
```

ヘッダ<float.h>は、のさまざまなをします。

はです。しかし、のプラットフォームarm、x86、x86_64、MIPSのほとんどはIEEE 754をします。

Cには、の3つのオプションのもあります。

の

Cのなのは、がされたオブジェクトをののようにすることをすることです。

じとをつののセットは、でされます。

- ポインタをす`*`「」。
- をすbinary `[]` "array subscription"。
- をす1 + n-ary `()` "びし"。
- リストされたのとをオーバーライドする`()`グループ。

の3つのは、のとをっています。

オペレーター		
`[]` サブスクリプション	1	からへ
`()` びし	1	からへ
`*`	2	からへ

をするときには、からにかってし、のにってしいでするをするがあります。のアプリケーションは、ののできえることができます。

`thing[X]`	のサイズの`X`の...
`thing(t1, t2, t3)`	`t1` 、 `t2` 、 `t3`をしてる...
`*thing`	へのポインタ...

したがって、ののまりはにでまり、のにあるでわります。

```
char *names[20];
```

`[]`は`*`よりもされるので、はのとおりです`names`は、 `char`へのポインタのサイズ20のです。

```
char (*place)[10];
```

をきするためにをするには、 `*`にされている `place`サイズ10のへのポインタである`char`。

```
int fn(long, short);
```

ここでするはありません`fn`は`long` 、 `short` 、 `int`をすです。

```
int *fn(void);
```

に`()`がされます`fn`は`void`をって`int`へのポインタをすです。

```
int (*fp)(void);
```

()のをオーバーライドするfpは、 voidをってintをすへのポインタです。

```
int arr[5][8];
```

はルールのではありません。 []は、のにってからのでされますarrは、 intのサイズ8ののサイズ5のです。

```
int **ptr;
```

2つのはしいをつため、がになります。はからのでされますptrはintへのポインタへのポインタです。

の

カンマは、のステートメントでのをるために、セパレータ*はカンマのようにはしませんとしてできます。のステートメントには、5つのがまれています。

```
int fn(void), *ptr, (*fp)(int), arr[10][20], num;
```

のでされたオブジェクトはのとおりです。

- fn voidをってintをす。
- ptr intへのポインタ。
- fp のへのポインタintとすint。
- arr intのサイズ20ののサイズ10の。
- num int。

の

はをするので、は、オブジェクトにしてなと、そののなののからすることもできます。にあるタイプは、すべてのをしたにられるです。

```
/*
 * Subscripting "arr" and dereferencing it yields a "char" result.
 * Particularly: *arr[5] is of type "char".
 */
char *arr[20];

/*
 * Calling "fn" yields an "int" result.
 * Particularly: fn('b') is of type "int".
 */
int fn(char);

/*
 * Dereferencing "fp" and then calling it yields an "int" result.
```

```
 * Particularly: (*fp)() is of type "int".
 */
int (*fp)(void);

/*
 * Subscripting "strings" twice and dereferencing it yields a "char" result.
 * Particularly: *strings[5][15] is of type "char"
 */
char *strings[10][20];
```

オンラインでデータをむ https://riptutorial.com/ja/c/topic/309/データ

20: テストフレームワーク

き

くのは、テストをして、ソフトウェアがりにすることをします。テストは、よりきなソフトウェアのさなをチェックし、がりにすることをします。テストフレームワークは、セットアップ/ティアダウンサービスをし、テストをすることでテストをにします。

Cでできるテストフレームワークはたくさんあります。たとえば、UnityはなCフレームワークです。 Cコードをテストするには、C ++のテストフレームワークをすることがよくあります。くのC ++テストフレームワークもあります。

テストハーネス

TDD - テスト

Cでのメカニズムのテスト

1. リンクの
2. ポインタ
3. プリプロセッサ
4. されたリンクおよびポインタ

CでされるC ++テストフレームワークにするここでするように、Cプログラムをテストするためにc ++フレームワークをすることは、になです 。

Examples

CppUTest

Cpputestは、CとC ++のユニットテストのxUnitスタイルのフレームワークです。 C ++でかれており、とシンプルさをとしています。メモリリークの、モックの、Googleテストとそのテストのをサポートしています。 Visual StudioとEclipse CDTのヘルパースクリプトとサンプルプロジェクトがしています。

```
#include <CppUTest/CommandLineTestRunner.h>
#include <CppUTest/TestHarness.h>


TEST_GROUP(Foo_Group) {}

TEST(Foo_Group, Foo_TestOne) {}

/* Test runner may be provided options, such
   as to enable colored output, to run only a
   specific test or a group of tests, etc. This
```

```
   will return the number of failed tests. */

int main(int argc, char ** argv)
{
    RUN_ALL_TESTS(argc, argv);
}
```

テストグループには、 `setup()` `teardown()`メソッドと`teardown()`メソッドがあります。 `setup`メソッドはテストのにびされ、 `teardown()`メソッドはにびされます。どちらもオプションで、どちらかをですることができます。のメソッドやもグループでされ、そのグループのすべてのテストでできます。

```
TEST_GROUP(Foo_Group)
{
    size_t data_bytes = 128;
    void * data;

    void setup()
    {
        data = malloc(data_bytes);
    }

    void teardown()
    {
        free(data);
    }

    void clear()
    {
        memset(data, 0, data_bytes);
    }
}
```

ユニティテストフレームワーク

UnityはユニットテストのためのxUnitスタイルのテストフレームワークです。これはCでにかれており、があり、く、シンプルで、かでがあります。みみシステムのテストににつようにされています。

のりをチェックするなテストケースは、のようになります

```
void test_FunctionUnderTest_should_ReturnFive(void)
{
    TEST_ASSERT_EQUAL_INT( 5, FunctionUnderTest() );
}
```

なテストファイルはのようになります。

```
#include "unity.h"
#include "UnitUnderTest.h" /* The unit to be tested. */

void setUp (void) {} /* Is run before every test, put unit init calls here. */
void tearDown (void) {} /* Is run after every test, put unit clean-up calls here. */
```

```
void test_TheFirst(void)
{
    TEST_IGNORE_MESSAGE("Hello world!"); /* Ignore this test but print a message. */
}

int main (void)
{
    UNITY_BEGIN();
    RUN_TEST(test_TheFirst); /* Run the test. */
    return UNITY_END();
}
```

Unityにはいくつかのサンプルプロジェクト、makefile、およびよりいテストファイルのをけするいくつかのRubyレーキスクリプトがしています。

CMocka

CMockaは、モックオブジェクトをサポートするCのためのされたユニットテストフレームワークです。のCライブラリのみをとし、さまざまなコンピューティングプラットフォームみみをむやさまざまなコンパイラでします。それはモック、APIドキュメント、およびさまざまなでのテストにするチュートリアルをっています。

```
#include <stdarg.h>
#include <stddef.h>
#include <setjmp.h>
#include <cmocka.h>

void null_test_success (void ** state) {}

void null_test_fail (void ** state)
{
    assert_true (0);
}

/* These functions will be used to initialize
   and clean resources up after each test run */
int setup (void ** state)
{
    return 0;
}

int teardown (void ** state)
{
    return 0;
}


int main (void)
{
    const struct CMUnitTest tests [] =
    {
        cmocka_unit_test (null_test_success),
        cmocka_unit_test (null_test_fail),
    };

    /* If setup and teardown functions are not
```

```
       needed, then NULL may be passed instead */

    int count_fail_tests =
        cmocka_run_group_tests (tests, setup, teardown);

    return count_fail_tests;
}
```

オンラインでテストフレームワークをむ https://riptutorial.com/ja/c/topic/6779/テストフレームワーク

21: ビットフィールド

き

Cのほとんどのはのバイトのサイズをっています。ビットフィールドは、バイトをずしもしないのです。それらはののビットとすることができる。のビットフィールドをのにパックすることができる。それらはCのですが、がするくのがあります。らはCのでもがいの1つです。

- のsize;

パラメータ

パラメータ	
	`signed`、 `unsigned`、 `int`または`_Bool`
	のこのフィールドの
サイズ	このフィールドにするビット

ビットフィールドののなタイプ`signed`、 `unsigned signed`、 `unsigned`または`_Bool`です。プレーン`int`ができますが、では§6.7.2¶5 ビットフィールドにしては、*`int signed int`*とじまたは*`unsigned int`*とじをするかどうかはです。

のはのでされるかもしれませんが、それらをうことはがありません。

Examples

ビットフィールド

なビットフィールドをして、のビットをむのあるものをすることができます。

```
struct encoderPosition {
   unsigned int encoderCounts : 23;
   unsigned int encoderTurns  : 4;
   unsigned int _reserved     : 5;
};
```

このでは、23ビットのと4ビットのマルチターンをするエンコーダをえます。ビットフィールドは、のビットにするデータをするハードウェアとのインタフェースにされることがい。もう1つのは、FPGAとので、FPGAが32ビットセクションでデータをメモリにきみ、ハードウェアのみりをにします。

```
struct FPGAInfo {
    union {
        struct bits {
            unsigned int bulb1On  : 1;
            unsigned int bulb2On  : 1;
            unsigned int bulb1Off : 1;
            unsigned int bulb2Off : 1;
            unsigned int jetOn    : 1;
        };
        unsigned int data;
   };
};
```

このでは、々のビットのデ―タにアクセスしたり、デ―タパケットをきむことFPGAのをエミュレ―トするのためににされるをしました。たちはこのようなビットにアクセスすることができます

```
FPGAInfo fInfo;
fInfo.data = 0xFF34F;
if (fInfo.bits.bulb1On) {
    printf("Bulb 1 is on\n");
}
```

これはですが、C996.7.2.1、アイテム10

> ユニットのビットフィ―ルドのりてからまたはからへは、されています。

このようにビットフィ―ルドをするときは、エンディアンをするがあります。そのため、マシンのエンディアンをチェックするためにプリプロセッサディレクティブをするがあります。これのはのとおりです。

```
typedef union {
    struct bits {
#if defined(WIN32) || defined(LITTLE_ENDIAN)
    uint8_t commFailure :1;
    uint8_t hardwareFailure :1;
    uint8_t _reserved :6;
#else
    uint8_t _reserved :6;
    uint8_t hardwareFailure :1;
    uint8_t commFailure :1;
#endif
    };
    uint8_t data;
} hardwareStatus;
```

ビットフィ―ルドをさなとしてう

```
#include <stdio.h>

int main(void)
{
    /* define a small bit-field that can hold values from 0 .. 7 */
    struct
```

```
    {
        unsigned int uint3: 3;
    } small;

    /* extract the right 3 bits from a value */
    unsigned int value = 255 - 2; /* Binary 11111101 */
    small.uint3 = value;          /* Binary      101 */
    printf("%d", small.uint3);

    /* This is in effect an infinite loop */
    for (small.uint3 = 0; small.uint3 < 8; small.uint3++)
    {
        printf("%d\n", small.uint3);
    }

    return 0;
}
```

ビットフィールドアライメント

ビットフィールドは、よりさいフィールドをすることができます。ビットフィールドは、バイトレベルまたはワードレベルのマスクでされます。のでは、8バイトのがされます。

```
struct C
{
    short s;            /* 2 bytes */
    char  c;            /* 1 byte */
    int   bit1 : 1;     /* 1 bit */
    int   nib  : 4;     /* 4 bits padded up to boundary of 8 bits. Thus 3 bits are padded */
    int   sept : 7;     /* 7 Bits septet, padded up to boundary of 32 bits. */
};
```

コメントには1つのなレイアウトがされていますが、ではアドレスなストレージユニットのアライメントがされていないため 、のレイアウトもです。

のないビットフィールドのサイズはですが、やはできません。

ゼロのビットフィールドにはをけることができず、のフィールドをビットフィールドのデータでされたにえます。これは、ビットフィールドのビットをパディングすることによってされる。

'A'のサイズは1バイトです。

```
struct A
{
    unsigned char c1 : 3;
    unsigned char c2 : 4;
    unsigned char c3 : 1;
};
```

Bでは、ののないビットフィールドは2ビットをスキップします。ゼロビットフィールドの$_{c2}$せる$_{c3}$チャーからするSO 3ビットがにスキップされ$_{c2}$び$_{c3}$ 3つのパディングビットがにする$_{c4}$。このようなのサイズは2バイトです。

```
struct B
{
    unsigned char c1 : 1;
    unsigned char    : 2;    /* Skips 2 bits in the layout */
    unsigned char c2 : 2;
    unsigned char    : 0;    /* Causes padding up to next container boundary */
    unsigned char c3 : 4;
    unsigned char c4 : 1;
};
```

ビットフィールドはいつにつのですか

ビットフィールドは、とに、くのを1つのオブジェクトにまとめるためにされます。これにより、メモリがされ、みみでにちます。

```
 e.g. consider the following variables having the ranges as given below.
 a --> range 0 - 3
 b --> range 0 - 1
 c --> range 0 - 7
 d --> range 0 - 1
 e --> range 0 - 1
```

これらのを々にするは、それぞれがなくとも8ビットのでなければならず、なスペースのは5バイトになります。さらに、は8ビットなし0-255のをすることはありません。ここでは、ビットフィールドをできます。

```
typedef struct {
   unsigned int a:2;
   unsigned int b:1;
   unsigned int c:3;
   unsigned int d:1;
   unsigned int e:1;
} bit_a;
```

のビットフィールドはのとじようにアクセスされます。プログラマは、がでかれていることにするがあります。の、はです。

```
int main(void)
{
   bit_a bita_var;
   bita_var.a = 2;              // to write into element a
   printf ("%d",bita_var.a);    // to read from element a.
   return 0;
}
```

くの、プログラマはビットフィールドのセットをゼロにしたいとう。これはごとにうことができますが、2のがあります。のサイズのなしので、ののをするだけです。に、このなしをゼロにすることによって、ビットフィールドのセットをゼロにすることができる。

```
typedef union {
    struct {
       unsigned int a:2;
```

```
        unsigned int b:1;
        unsigned int c:3;
        unsigned int d:1;
        unsigned int e:1;
    };
    uint8_t data;
} union_bit;
```

はのとおりです

```
int main(void)
{
   union_bit un_bit;
   un_bit.data = 0x00;         // clear the whole bit-field
   un_bit.a = 2;               // write into element a
   printf ("%d",un_bit.a);     // read from element a.
   return 0;
}
```

として、ビットフィールドは、されたをることができるくのをつメモリでにされます。

ビットフィールドではない

1. ビットフィールドの、ビットフィールドへのポインタ、およびビットフィールドをすはできません。
2. アドレスはビットフィールドメンバにはできません。
3. ビットフィールドのデータは、フィールドのサイズをするのになでなければなりません。
4. `sizeof()`はビットフィールドにできません。
5. ビットフィールドの`typedef`をしてするはありませんただし、ビットフィールドをむの`typedef`をすることはです。

```
typedef struct mybitfield
{
    unsigned char c1 : 20;   /* incorrect, see point 3 */
    unsigned char c2 : 4;    /* correct */
    unsigned char c3 : 1;
    unsigned int x[10]: 5;   /* incorrect, see point 1 */
} A;

int SomeFunction(void)
{
    // Somewhere in the code
    A a = { … };
    printf("Address of a.c2 is %p\n", &a.c2);      /* incorrect, see point 2 */
    printf("Size of a.c2 is %zu\n", sizeof(a.c2)); /* incorrect, see point 4 */
}
```

オンラインでビットフィールドをむ https://riptutorial.com/ja/c/topic/1930/ビットフィールド

22: ファイルとストリーム

- #include <stdio.h> / *のセクションのいずれかをするには、これをインクルードします。* /
- FILE * fopenconst char * path、const char * mode; / *されたモードの パスでファイルのストリームをく* /
- FILE * freopenconst char *パス、const char *モード、FILE *ストリーム; / *されたモードでパスのファイルののストリームをオープンする* /
- int fcloseFILE *ストリーム; / *いているストリームをじる* /
- size_t freadvoid * ptr、size_t size、size_t nmemb、FILE * stream; / * ストリームからそれぞれ*size*バイトの*nmemb*のをみみ、 *ptr*にきみます。みまれたのをします。 * /
- size_t fwriteconst void * ptr、size_t size、size_t nmemb、FILE * stream; / * *size*バイトの*nmemb*をそれぞれ*ptr*からストリームにきみます 。きまれたのをします。 * /
- int fseekFILE *ストリーム、long offset、int whence; / *オフセット*whence*でっにする、 オフセットするストリームのカーソルをし、それがしたは0をします。 * /
- いftellFILE *ストリーム; / *ストリームのからのカーソルのオフセットをします。 * /
- void rewindFILE *ストリーム; / *カーソルをファイルのにします。 * /
- int fprintfFILE * fout、const char * fmt、...; / * foutにprintfのをきみます* /
- FILE * stdin; / *ストリーム* /
- FILE * stdout; / *ストリーム* /
- FILE * stderr; / *エラーストリーム* /

パラメーター

パラメータ	
const char * モード	ファイルバックアップされたストリームのモードをす。なについてはをしてください。
int whence	することができ`SEEK_SET`ファイルのからするには、 `SEEK_END`その、またはからする`SEEK_CUR`のカーソルのをにします。 `SEEK_END`はがありません。

モード

`fopen()`および`freopen()`モードは、のいずれかのになります。

- `"r"` カーソルをファイルのにして、みりモードでファイルをきます。
- `"r+"` ファイルをみきモードでき、カーソルをファイルのにします。
- `"w"` きみモードでファイルをくかします。は0バイトにりてられます。カーソルはファイルのにされます。
- `"w+"` みきモードでファイルをくかします。は0バイトにりてられます。カーソルはファイルのにされます。

- "a" カーソルをファイルのにして、きみモードでファイルをくかします。
- "a+" みきモードでファイルをくかし、みみカーソルをファイルのにします。ただし、はにファイルのにされます。

これらのファイルモードのそれぞれには、ののにbすることができます "rb"または"a+b"または"ab+"。 bは、いがあるシステムのテキストファイルではなく、ファイルをバイナリファイルとしてうことをします。 Unixのようなシステムではいはありません。 Windowsシステムではです。また、Windowsのfopenでは、 bわりになtして 'テキストファイル'をできます。

C11

- "wx" きみモードでテキストファイルをします。 ファイルがしないがあります 。
- "wbx" きみモードでバイナリファイルをします。 ファイルがしないがあります 。

xがする、 xはモードののでなければなりません。

Examples

ファイルをいてファイルにきむ

```
#include <stdio.h>   /* for perror(), fopen(), fputs() and fclose() */
#include <stdlib.h>  /* for the EXIT_* macros */

int main(int argc, char **argv)
{
    int e = EXIT_SUCCESS;

    /* Get path from argument to main else default to output.txt */
    char *path = (argc > 1) ? argv[1] : "output.txt";

    /* Open file for writing and obtain file pointer */
    FILE *file = fopen(path, "w");

    /* Print error message and exit if fopen() failed */
    if (!file)
    {
        perror(path);
        return EXIT_FAILURE;
    }

    /* Writes text to file. Unlike puts(), fputs() does not add a new-line. */
    if (fputs("Output in file.\n", file) == EOF)
    {
        perror(path);
        e = EXIT_FAILURE;
    }

    /* Close file */
    if (fclose(file))
    {
        perror(path);
        return EXIT_FAILURE;
    }
    return e;
```

```
}
```

このプログラムはmainにでえられたでファイルをき、がえられていなければ`output.txt`デフォルトにします。じのファイルがすでにする、そのはされ、ファイルはしいのファイルとしてわれます。ファイルがまだしないは、 `fopen()`びしによってファイルがされます。

`fopen()`びしがらかのですると、 `NULL`を`errno` 、グロ―バル`errno`を`errno`ます。これは、プログラムが`fopen()`びしのにりをテストし、 `fopen()`がしたに`perror()`をできることをします。

`fopen()`びしがすると、な`FILE`ポインタをします。 `fclose()`がびされるまで、このポインタをしてこのファイルをすることができます。

`fputs()`は、されたテキストをいたファイルにきんで、ファイルののをきえます。 `fopen()`とに、`fputs()`はしたにも`errno`をし`errno`が、このはをす`EOF`をしますそうでないはではないをします。

`fclose()`はバッファをフラッシュし、ファイルをじ、 `FILE *`すメモリをします。りは`fputs()`じようにをします`fputs()`したは '0'をします。したは`errno`もし`errno`。

fprintf

`fprintf`は、 `printf`したコンソ―ルとじように、ファイルでできます。えば、あなたがくかもしれないゲ―ムの、、およびをする

```
/* saves wins, losses and, ties */
void savewlt(FILE *fout, int wins, int losses, int ties)
{
    fprintf(fout, "Wins: %d\nTies: %d\nLosses: %d\n", wins, ties, losses);
}
```

システムのあるのはWindowsは、ほとんどのプログラマが「の」とぶものをしません。 UNIXシステムは\ nをってをしますが、\ rと\ nのペアがされます。このシ―ケンスはにCRLFとばれます。ただし、Cをするたびに、これらのにプラットフォ―ムにするをするはありません。 \ nのすべてのインスタンスをしいプラットフォ―ムにするには、ACコンパイラがです。したがって、Windowsコンパイラは\ nを\ r \ nにしますが、UNIXコンパイラはそのまします。

プロセスをする

```
#include <stdio.h>

void print_all(FILE *stream)
{
    int c;
    while ((c = getc(stream)) != EOF)
        putchar(c);
}
int main(void)
{
    FILE *stream;
```

```
    /* call netstat command. netstat is available for Windows and Linux */
    if ((stream = popen("netstat", "r")) == NULL)
        return 1;

    print_all(stream);
    pclose(stream);
    return 0;
}
```

このプログラムは`popen()`をしてプロセス `netstat` をし、プロセスからのすべてのをみみ、にエコーします。

`popen()`はCライブラリにはしませんが、むしろPOSIX Cのです

getlineをってファイルからをする

POSIX Cライブラリは`getline()`をします。このは、のをするバッファをりて、しい、の、およびバッファのサイズをします。

`example.txt`からをするサンプルプログラム

```
#include <stdlib.h>
#include <stdio.h>


#define FILENAME "example.txt"

int main(void)
{
  /* Open the file for reading */
  char *line_buf = NULL;
  size_t line_buf_size = 0;
  int line_count = 0;
  ssize_t line_size;
  FILE *fp = fopen(FILENAME, "r");
  if (!fp)
  {
    fprintf(stderr, "Error opening file '%s'\n", FILENAME);
    return EXIT_FAILURE;
  }

  /* Get the first line of the file. */
  line_size = getline(&line_buf, &line_buf_size, fp);

  /* Loop through until we are done with the file. */
  while (line_size >= 0)
  {
    /* Increment our line count */
    line_count++;

    /* Show the line details */
    printf("line[%06d]: chars=%06zd, buf size=%06zu, contents: %s", line_count,
        line_size, line_buf_size, line_buf);

    /* Get the next line */
    line_size = getline(&line_buf, &line_buf_size, fp);
  }
```

```
  /* Free the allocated line buffer */
  free(line_buf);
  line_buf = NULL;

  /* Close the file now that we are done with it */
  fclose(fp);

  return EXIT_SUCCESS;
}
```

ファイル **example.txt**

```
This is a file
  which has
multiple lines
    with various indentation,
blank lines



a really long line to show that getline() will reallocate the line buffer if the length of a
line is too long to fit in the buffer it has been given,
  and punctuation at the end of the lines.
```

```
line[000001]: chars=000015, buf size=000016, contents: This is a file
line[000002]: chars=000012, buf size=000016, contents:   which has
line[000003]: chars=000015, buf size=000016, contents: multiple lines
line[000004]: chars=000030, buf size=000032, contents:     with various indentation,
line[000005]: chars=000012, buf size=000032, contents: blank lines
line[000006]: chars=000001, buf size=000032, contents:
line[000007]: chars=000001, buf size=000032, contents:
line[000008]: chars=000001, buf size=000032, contents:
line[000009]: chars=000150, buf size=000160, contents: a really long line to show that
getline() will reallocate the line buffer if the length of a line is too long to fit in the
buffer it has been given,
line[000010]: chars=000042, buf size=000160, contents:  and punctuation at the end of the
lines.
line[000011]: chars=000001, buf size=000160, contents:
```

このでは、に`getline()`がびされ、バッファはりてられません。こののびしの、 `getline()`はバッファをりて、のをみり、そののをしいバッファにれます。のびしでは、 `getline()`はじバッファをし、バッファがにまるようになったときにのみバッファをりてします。バッファは、ファイルがするとされます。

のオプションは`getdelim()`です。これは、のをするは`getline()`とじです。これは、ファイルタイプのののが '\ n'でないにのみです。のマルチバイト `"\r\n")` '\ `"\r\n")`はののなので、 `getline()`はWindowsのテキストファイルでもします。

getline()

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
```

```
#include <stdint.h>


#if !(defined _POSIX_C_SOURCE)
typedef long int ssize_t;
#endif

/* Only include our version of getline() if the POSIX version isn't available. */

#if !(defined _POSIX_C_SOURCE) || _POSIX_C_SOURCE < 200809L

#if !(defined SSIZE_MAX)
#define SSIZE_MAX (SIZE_MAX >> 1)
#endif

ssize_t getline(char **pline_buf, size_t *pn, FILE *fin)
{
  const size_t INITALLOC = 16;
  const size_t ALLOCSTEP = 16;
  size_t num_read = 0;

  /* First check that none of our input pointers are NULL. */
  if ((NULL == pline_buf) || (NULL == pn) || (NULL == fin))
  {
    errno = EINVAL;
    return -1;
  }

  /* If output buffer is NULL, then allocate a buffer. */
  if (NULL == *pline_buf)
  {
    *pline_buf = malloc(INITALLOC);
    if (NULL == *pline_buf)
    {
      /* Can't allocate memory. */
      return -1;
    }
    else
    {
      /* Note how big the buffer is at this time. */
      *pn = INITALLOC;
    }
  }

  /* Step through the file, pulling characters until either a newline or EOF. */

  {
    int c;
    while (EOF != (c = getc(fin)))
    {
      /* Note we read a character. */
      num_read++;

      /* Reallocate the buffer if we need more room */
      if (num_read >= *pn)
      {
        size_t n_realloc = *pn + ALLOCSTEP;
        char * tmp = realloc(*pline_buf, n_realloc + 1); /* +1 for the trailing NUL. */
        if (NULL != tmp)
        {
          /* Use the new buffer and note the new buffer size. */
```

```
          *pline_buf = tmp;
          *pn = n_realloc;
        }
        else
        {
          /* Exit with error and let the caller free the buffer. */
          return -1;
        }

        /* Test for overflow. */
        if (SSIZE_MAX < *pn)
        {
          errno = ERANGE;
          return -1;
        }
      }

      /* Add the character to the buffer. */
      (*pline_buf)[num_read - 1] = (char) c;

      /* Break from the loop if we hit the ending character. */
      if (c == '\n')
      {
        break;
      }
    }

    /* Note if we hit EOF. */
    if (EOF == c)
    {
      errno = 0;
      return -1;
    }
  }

  /* Terminate the string by suffixing NUL. */
  (*pline_buf)[num_read] = '\0';

  return (ssize_t) num_read;
}

#endif
```

バイナリファイルをいてきむ

```
#include <stdlib.h>
#include <stdio.h>


int main(void)
{
   result = EXIT_SUCCESS;

   char file_name[] = "outbut.bin";
   char str[] = "This is a binary file example";
   FILE * fp = fopen(file_name, "wb");

   if (fp == NULL)  /* If an error occurs during the file creation */
   {
```

```
    result = EXIT_FAILURE;
    fprintf(stderr, "fopen() failed for '%s'\n", file_name);
  }
  else
  {
    size_t element_size = sizeof *str;
    size_t elements_to_write = sizeof str;

    /* Writes str (_including_ the NUL-terminator) to the binary file. */
    size_t elements_written = fwrite(str, element_size, elements_to_write, fp);
    if (elements_written != elements_to_write)
    {
      result = EXIT_FAILURE;
      /* This works for >=c99 only, else the z length modifier is unknown. */
      fprintf(stderr, "fwrite() failed: wrote only %zu out of %zu elements.\n",
        elements_written, elements_to_write);
      /* Use this for <c99: *
      fprintf(stderr, "fwrite() failed: wrote only %lu out of %lu elements.\n",
        (unsigned long) elements_written, (unsigned long) elements_to_write);
       */
    }

    fclose(fp);
  }

  return result;
}
```

このプログラムはバイナリのテキストをし、 `fwrite`をしてファイル`output.bin`きみます。

じのファイルがすでにする、そのはされ、ファイルはしいのファイルとしてわれます。

バイナリストリームは、デ―タをにできるけられたです。このモ―ドでは、バイトはプログラムとファイルのにもされずにきまれます。

をにくには、ファイルがビッグエンディアンかリトルエンディアンか、サイズは16,32,64ビットであるかどうかをるがあります。ビットシフトとマスキングをして、バイトをしいできすことができます。 Cのは、2のをつことはされていませんほとんどすべてのがいますが。いにも、なしへのでは 2のをすることがされます。したがって、きをバイナリファイルにきむためのコ―ドはしくべきものです。

```
/* write a 16-bit little endian integer */
int fput16le(int x, FILE *fp)
{
    unsigned int rep = x;
    int e1, e2;

    e1 = fputc(rep & 0xFF, fp);
    e2 = fputc((rep >> 8) & 0xFF, fp);

    if(e1 == EOF || e2 == EOF)
        return EOF;
    return 0;
}
```

のは、サイズとバイトオ―ダ―のをえてじパタ―ンにいます。

fscanf

テキストファイルがあり、いくつかのをたすために、そのファイルのすべてのをみたいとしましょう。

file.txt

```
This is just
a test file
to be used by fscanf()
```

これがなです

```
#include <stdlib.h>
#include <stdio.h>

void printAllWords(FILE *);

int main(void)
{
  FILE *fp;

  if ((fp = fopen("file.txt", "r")) == NULL) {
      perror("Error opening file");
      exit(EXIT_FAILURE);
  }

  printAllWords(fp);

  fclose(fp);

  return EXIT_SUCCESS;
}

void printAllWords(FILE * fp)
{
    char tmp[20];
    int i = 1;

    while (fscanf(fp, "%19s", tmp) != EOF) {
        printf("Word %d: %s\n", i, tmp);
        i++;
    }
}
```

はのようになります。

```
Word 1: This
Word 2: is
Word 3: just
Word 4: a
Word 5: test
Word 6: file
Word 7: to
Word 8: be
Word 9: used
```

```
Word 10: by
Word 11: fscanf()
```

ファイルからをみむ

stdio.hヘッダはfgets()をします。このは、ストリームからをみみ、されたにします。このは、n - 1のいずれかをみんだり、'\n'をみんだり、ファイルのわりEOFにしたときに、ストリームからテキストのみみをします。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LINE_LENGTH 80

int main(int argc, char **argv)
{
    char *path;
    char line[MAX_LINE_LENGTH] = {0};
    unsigned int line_count = 0;

    if (argc < 1)
        return EXIT_FAILURE;
    path = argv[1];

    /* Open file */
    FILE *file = fopen(path, "r");

    if (!file)
    {
        perror(path);
        return EXIT_FAILURE;
    }

    /* Get each line until there are none left */
    while (fgets(line, MAX_LINE_LENGTH, file))
    {
        /* Print each line */
        printf("line[%06d]: %s", ++line_count, line);

        /* Add a trailing newline to lines that don't already have one */
        if (line[strlen(line) - 1] != '\n')
            printf("\n");
    }

    /* Close file */
    if (fclose(file))
    {
        return EXIT_FAILURE;
        perror(path);
    }
}
```

のテキストをむファイルへのパスであるをしてプログラムをびします。

```
This is a file
```

```
  which has
multiple lines
    with various indentation,
blank lines



a really long line to show that the line will be counted as two lines if the length of a line
is too long to fit in the buffer it has been given,
  and punctuation at the end of the lines.
```

はのようになります。

```
line[000001]: This is a file
line[000002]:   which has
line[000003]: multiple lines
line[000004]:     with various indentation,
line[000005]: blank lines
line[000006]:
line[000007]:
line[000008]:
line[000009]: a really long line to show that the line will be counted as two lines if the le
line[000010]: ngth of a line is too long to fit in the buffer it has been given,
line[000011]:  and punctuation at the end of the lines.
line[000012]:
```

このになでは、ののさがされているので、いは2としてにカウントされます。 `fgets()`は、びしたコーードが、みまれたのとしてされるメモリをすることをします。

POSIXは`getline()`をにします。わりに、なメモリがあるり、のさのににじてバッファをするためににメモリをりてます。

オンラインでファイルとストリームをむ https://riptutorial.com/ja/c/topic/507/ファイルとストリーム

23: ブール

みの`_Bool`タイプとヘッダ`<stdbool.h>`をするには、CのC99 / C11バージョンをしているがあります。

コンパイラのやエラーをけるため、C89のバージョンのをしているにのみ、 `typedef` / `define`をしてください。

Examples

stdbool.hの

C99

システムヘッダーファイル`stdbool.h`すると、 `bool`をブールデータとしてできます。 `true`は`1`され、 `false`は`0`され`0`。

```
#include <stdio.h>
#include <stdbool.h>

int main(void) {
    bool x = true;  /* equivalent to bool x = 1; */
    bool y = false; /* equivalent to bool y = 0; */
    if (x)  /* Functionally equivalent to if (x != 0) or if (x != false) */
    {
        puts("This will print!");
    }
    if (!y) /* Functionally equivalent to if (y == 0) or if (y == false) */
    {
        puts("This will also print!");
    }
}
```

`bool`は、データ`_Bool`なるらしいスペルです。やポインタをするときにはながあります。

#defineをう

すべてのバージョンのCは、 `0`ののをにして`true`として、`0`を`false`としてにい`false`。 C99のように`_Bool`や`bool`をできないは、Cで`#define`マクロをしてブールデータをシミュレートすることができますが、レガシーコードでもそのようなものをつけることができます。

```
#include <stdio.h>

#define bool int
#define true 1
#define false 0

int main(void) {
    bool x = true;  /* Equivalent to int x = 1; */
```

```
    bool y = false; /* Equivalent to int y = 0; */
    if (x) /* Functionally equivalent to if (x != 0) or if (x != false) */
    {
        puts("This will print!");
    }
    if (!y) /* Functionally equivalent to if (y == 0) or if (y == false) */
    {
        puts("This will also print!");
    }
}
```

これらのマクロのがな`<stdbool.h>`とするがあるので、これをしいコードにしないでください。

みみみみの_Bool

C99

Cバージョン C99にえて、 `_Bool` もネイティブCデータです。 `0` *false*の と `1` *true*の をすることができます。

```
#include <stdio.h>

int main(void) {
    _Bool x = 1;
    _Bool y = 0;
    if(x) /* Equivalent to if (x == 1) */
    {
        puts("This will print!");
    }
    if (!y) /* Equivalent to if (y == 0) */
    {
        puts("This will also print!");
    }
}
```

`_Bool`はですが、のからのにはながあります。は`if`のののにています。では

```
_Bool z = X;
```

- `X`がののである、 `X == 0`なら`z`は`0`なり`X == 0` 。それの、 `z`は`1`なります。
- `X`がポインタの、 `X`がヌルポインタのは`z`は`0`なり、それのは`1`ます。

`bool` 、 `false` 、および`true`よりいスペルをするには、 `<stdbool.h>`をするがあります。

ブールのとポインタ。

すべてのまたはポインタは、「」としてされるでできます。

```
int main(int argc, char* argv[]) {
  if (argc % 4) {
    puts("arguments number is not divisible by 4");
  } else {
```

```
    puts("argument number is divisible by 4");
  }
...
```

argc % 4され、のいずれかにつながる0 、 1 、 2または3 。 の0は「 」でelseにをもたらすのelse 。 のすべてのは "true"で、 ifにります。

```
double* A = malloc(n*sizeof *A);
if (!A) {
   perror("allocation problems");
   exit(EXIT_FAILURE);
}
```

ここでポインタAがされ、ポインタがヌルポインタであればエラーがされ、プログラムはする。

くのは、 A == NULLようなものをくことをみますが、のなのでこのようなポインタのがあると、みにくくなります。

```
char const* s = ....;   /* some pointer that we receive */
if (s != NULL && s[0] != '\0' && isalpha(s[0])) {
   printf("this starts well, %c is alphabetic\n", s[0]);
}
```

これをするには、ののなコードをスキャンし、オペレータのみをするがあります。

```
char const* s = ....;   /* some pointer that we receive */
if (s && s[0] && isalpha(s[0])) {
   printf("this starts well, %c is alphabetic\n", s[0]);
}
```

キャプチャするのがですポインタがな、のがゼロでないかどうかをし、それがかどうかをします。

typedefをってboolをする

ほとんどのデバッガは#defineマクロをしていないが、 enumをチェックすることができるので、のようなことがましいかもしれません。

```
#if __STDC_VERSION__ < 199900L
typedef enum { false, true } bool;
/* Modern C code might expect these to be macros. */
# ifndef bool
#  define bool bool
# endif
# ifndef true
#  define true true
# endif
# ifndef false
#  define false false
# endif
#else
# include <stdbool.h>
```

```
#endif

/* Somewhere later in the code ... */
bool b = true;
```

これにより、Cのバージョンのコンパイラはすることができますが、コードがのCコンパイラでコンパイルされているは、をちます。

については`typedef`、typedefのをよりにするために、`enum`をを

オンラインでブールをむ https://riptutorial.com/ja/c/topic/3336/ブール

24: フォーマットされたI

Examples

オブジェクトへのポインタのの

ポインター—ではなくオブジェクトへのポインター—のをするには、 `p`をします。するようにされ`void`そうでないのをプリントアウトするために、の-pointersを`void`、それがためににするがあります「*キャスト」-pointer `void*`。

```
#include <stdlib.h> /* for EXIT_SUCCESS */
#include <stdio.h>  /* for printf() */

int main(void)
{
  int i;
  int * p = &i;

  printf("The address of i is %p.\n", (void*) p);

  return EXIT_SUCCESS;
}
```

C99

`<inttypes.h>`と`uintptr_t`

C99でポインタをするのは、 `uintptr_t`と`<inttypes.h>`マクロをします。

```
#include <inttypes.h> /* for uintptr_t and PRIXPTR */
#include <stdio.h>    /* for printf() */

int main(void)
{
  int  i;
  int *p = &i;

  printf("The address of i is 0x%" PRIXPTR ".\n", (uintptr_t)p);

  return 0;
}
```

には、にされたポインタをできるはしないがあります `uintptr_t`がしないがあります。には、それはします。へのポインタは、 `uintptr_t`にであるはありません。

は`uintptr_t`タイプがし、そう`intptr_t`タイプは。しかし、なぜアドレスをきとしていたいのかはらかではありません。

KR C89

プレスタンダードヒストリー

KR-CのC89は、 `void*`ヘッダ`<stdlib.h>`もプロトタイプもなく`int main(void)`もありませんでしたなので、ポインタは`long unsigned int`にキャストされ、 `lx`さ/。

のはのためのものです。ではこれはなコードであり、いのるいをきこすがあります。

```
#include <stdio.h> /* optional in pre-standard C - for printf() */

int main()
{
  int  i;
  int *p = &i;

  printf("The address of i is 0x%lx.\n", (long unsigned) p);

  return 0;
}
```

2つのポインタののをオブジェクトにする

オブジェクトへの2つのポインタのをすると、き* 1がられます。したがって、なくとも `d`をしてされます。

C99 `<stddef.h>`は`ptrdiff_t`をしているので、このような "ポインタ"をできるだけのがあることをする。 `ptrdiff_t`をするには、 `t` lengthをします。

C99

```
#include <stdlib.h> /* for EXIT_SUCCESS */
#include <stdio.h> /* for printf() */
#include <stddef.h> /* for ptrdiff_t */


int main(void)
{
  int a[2];
  int * p1 = &a[0], * p2 = &a[1];
  ptrdiff_t pd = p2 - p1;

  printf("p1 = %p\n", (void*) p1);
  printf("p2 = %p\n", (void*) p2);
  printf("p2 - p1 = %td\n", pd);

  return EXIT_SUCCESS;
}
```

はのようになります。

```
p1 = 0x7fff6679f430
p2 = 0x7fff6679f434
p2 - p1 = 1
```

のとしてられるは、ポインタをしたポイントののサイズ、ここでは`int`でスケールされることにしてください。このの`int`のサイズは4です。

---

*1する2つのポインタがじオブジェクトをしていない、そのはです。

の

	の	
`i`、`d`	int	10をする
`u`	なし	10をする
`o`	なし	8をする
`x`	なし	16、をします。
`X`	なし	16、をします。
`f`	ダブル	がされていない、デフォルトの6でfloatをしますな`nan`と`inf`または`infinity`がされ`infinity`
`F`	ダブル	がされていないは、デフォルトの6でfloatをしますな`NAN`と`INF`または`INFINITY`がされます
`e`	ダブル	/をしてをして、がされていないはデフォルトの6でfloatをします。のとな
`E`	ダブル	/をしてをして、がされていないはデフォルトの6でfloatをします。のとな
`g`	ダブル	`f`または`e`いずれかをします*[を]*
`G`	ダブル	`F`または`E`いずれかをします*[を]*
`a`	ダブル	16、をします。
`A`	ダブル	16、をします。
`c`	チャー	1をする

	の	
s	char *	をNULまでするか、されているはでされたさにりてる
p	void *	void -pointerのをしvoid。 void -pointerはにvoid*キャストするがありvoid*。 オブジェクトポインタへのポインタ、ポインタではないポインタ
%	なし	%をする
n	int *	までにされたバイトをintされたint きみます。

ISO / IEC 9899:2011§7.21.6.1¶7にって、さを%nできることにしてください %hhnは、のnが*signed char*へのポインタにされることをします。

のは、デフォルトののために*float*と*double*にされます。§6.5.2.2びし、¶7 プロトタイプのは、にされたパラメータのでをさせます。ののプロモーションは、のにしてされます。 したがって、されるが*float*であっても、 printf()などのは*double*しかされません。

gとGでは、 eとf またはEとF のは、Cとprintf() POSIXでされています。

> をすdoubleは、されたとにじて、 fまたはeまたはGのはFまたはEにされます。 0のはを P 、がされているは6、がゼロのは1をPとします。に、スタイルEがXをつ、
>
> - P> X> = -4の、はスタイルf またはF および(P - (X+1))ます。
> - それの、はスタイルe またはE とP - 1わなければならない。
>
> に、 "フラグがされていないり、のゼロはのからされ、がっていなければはされます。

printf

<stdio.h>を<stdio.h>アクセスすると、printf()は、Cのコンソールにテキストをするためにされるなツールです。

```
printf("Hello world!");
// Hello world!
```

のフォーマットされていないは、かっこのにすることで、でできます。

```
printf("%d is the answer to life, the universe, and everything.", 42);
// 42 is the answer to life, the universe, and everything.

int x = 3;
char y = 'Z';
char* z = "Example";
printf("Int: %d, Char: %c, String: %s", x, y, z);
// Int: 3, Char: Z, String: Example
```

あるいは、、、などは、エスケープ%をして、フォーマットをすまたはの フォーマットをしてできます 。

printf()へのはすべてコンマでられ、これらのはとじでなければなりません。のはされますが、ってされたまたはのはエラーまたはのをきこします。は、リテラルまたはのいずれかです。

にされた、されたがintでされます。それの、はのをします。

さ

C99とC11のでは、 printf()のさをしています。そのはのとおりです。

モディファイア	する	にされます
hh	d、i、o、u、x、またはX	char 、 signed charまたはunsigned char
h	d、i、o、u、x、またはX	short intまたはunsigned short int
l	d、i、o、u、x、またはX	long intまたはunsigned long int
l	a、A、e、E、f、F、g、またはG	double scanf()とののため、C90では
ll	d、i、o、u、x、またはX	long long intまたはunsigned long long int
j	d、i、o、u、x、またはX	intmax_tまたはuintmax_t
z	d、i、o、u、x、またはX	size_tまたはするきPOSIXではssize_t
t	d、i、o、u、x、またはX	ptrdiff_tまたはするなし
L	a、A、e、E、f、F、g、またはG	long double

lengthがののとともにされる、そのはです。

Microsoftはいくつかのなるさをし、にhh 、 j 、 z 、またはtサポートしていません。

モディファイア	する	にされます
I32	d、i、o、x、またはX	__int32
I32	o、u、x、また	unsigned __int32

モディファイア	する	にされます
	はX	
I64	d、i、o、x、またはX	`__int64`
I64	o、u、x、またはX	`unsigned __int64`
	d、i、o、x、またはX	`ptrdiff_t` すなわち、 `__int32` 32ビットプラットフォームで、 `__int64` 64ビットプラットフォームで
	o、u、x、またはX	`size_t` すなわち、 `unsigned __int32` 32ビットプラットフォームで、 `unsigned __int64` 64ビットプラットフォームで
lまたはL	a、A、e、E、f、g、またはG	`long double` がビジュアルC ++では、 `long double`タイプであり、それはじする`double`。
lまたはw	cまたはC	`printf`と`wprintf`をつワイド。 `lc` 、 `lC` 、 `wc`は`wC`は、とである`C`で`printf`とをいて`c`で`wprintf`。
lまたはw	s、S、またはZ	`printf`と`wprintf`をつワイド。 `ls` 、 `lS` 、 `ws`または`wS`タイプとである`S`で`printf`とをいて`s`で`wprintf`。

`C` 、 `S` 、 `Z`と`I` 、 `I32` 、 `I64` 、および`w`さは、Microsoftのです。`l`ためのとして`long double`ではなく`double` 、あなたがないり、いをつけるためにハードされるでしょうが、となる`long double`なるがある`double`。

フォーマットフラグ

CのC11とC99もでは、 `printf()`のフラグをしています。

	コンバージョン	
`-`	すべて	のは、フィールドでめにしなければならない。このフラグがされていない、はめになります。
`+`	き	きのは、に '+'または '-'でまります。このフラグがされていないは、のがされたときにのみがされます。
`<space>`	き	きののがでない、またはきのがでないは、 `<space>`がのにされます。つまり

	コンバージョン	
		、 `<space>`フラグと「 `+` 」フラグのがされた、 `<space>`フラグはされます。
`#`	すべて	をフォームにすることをします。 `o`の、にじて、ののをにゼロにするとがとも0で、1つの0がされる。 `x`または`X`の、0でないには`0x` または`0X` がされます。のために、 `aA` 、 `e` 、 `E` 、 `f` 、 `F` 、 `g` 、および`G`、はにのはにわないでも、をまなければなりません。このフラグがなければ、がくにのみ、これらのにがされます。 `g`と`G`の、のゼロはのようにからしてはならない。のの、はです。
`0`		d、i、o、u、x、X、a、A、e、E、f、F、g、Gのの、またはのにくゼロをして、またはNaNをするをき、スペースパディングをするのではなく、をします。 '0'と '-'フラグがともされた、 '0'フラグはされます。 d、i、o、u、x、およびXの、がされている、 '0'フラグはされます。 '「0」と「 `<apostrophe>` 」ののフラグがされた、グループはゼロめのにされます。のコンバージョンの、はです。

これらのフラグは、 Microsoftでもじでサポートされています 。

`printf()`のPOSIXにはのものがされています。

	コンバージョン	
'	i、d、u、f、F、g、G	ののは、のグループでフォーマットされなければならない。のコンバージョンの、はです。グループがされます。

オンラインでフォーマットされた/をむ https://riptutorial.com/ja/c/topic/3750/フォーマットされた-

25: プリプロセッサとマクロ

き

すべてのプリプロセッサコマンドは、ハッシュポンド`#`まります。 ACマクロは`#define`プリプロセッサディレクティブをって`#define`されたプリプロセッサコマンドです。では、Cプリプロセッサ Cコンパイラのは、そのがされているにマクロのをきえるだけです。

コンパイラがコードでマクロをすると、ながされ、のはされません。このため、プリプロセッサによるはCプログラムのスコープをしません。たとえば、マクロはブロックにされず、ブロックステートメントをする`}`をけません。

プリプロセッサは、ではありません。プリプロセッサだけではできないいくつかのタイプのがあります。

、コンパイラにはコマンドラインフラグまたはのがあり、フェーズのにコンパイルをし、をすることができます。 POSIXプラットフォームでは、このフラグは`-E`。したがって、このフラグをしてgccをすると、されたソースがstdoutにされます。

```
$ gcc -E cprog.c
```

くの、プリプロセッサはのプログラムとしてされます。これはコンパイラによってびされ、そのプログラムのなは`cpp`です。くのプリプロセッサは、デバッグをするためにコンパイルののフェーズでされる、にするなどのサポートをします。プリプロセッサがgccにづいている、-Pオプションはそのようなをします。

```
$ cpp -P cprog.c
```

Examples

きおよびきの

コードのブロックをめるきにするには、プリプロセッサは、えば、いくつかのディレクティブをっている`#if` 、 `#ifdef` 、 `#else` 、 `#endif` 、など。

```
/* Defines a conditional `printf` macro, which only prints if `DEBUG`
 * has been defined
 */
#ifdef DEBUG
#define DLOG(x) (printf(x))
#else
#define DLOG(x)
#endif
```

`#if`

にはのCをできます

```
#if __STDC_VERSION__ >= 201112L
/* Do stuff for C11 or higher */
#elif __STDC_VERSION__ >= 199901L
/* Do stuff for C99 */
#else
/* Do stuff for pre C99 */
#endif
```

`#if`ディレクティブはC `if`とにし、のみをみ、キャストはみません。 1つのの、 `defined( identifier )`サポートします。これは、がされているは`1`をし、それのは`0`をします。

```
#if defined(DEBUG) && !defined(QUIET)
#define DLOG(x) (printf(x))
#else
#define DLOG(x)
#endif
```

の

ほとんどの、アプリケーションのリリースビルドにはできるだけオーバーヘッドがなくなるとされます。しかし、ビルドのテストに、のログとされたにするがちます。

たとえば、テストビルドをうときに、そのについてのログをする`SHORT SerOpPluAllRead(PLUIF *pPif, USHORT usLockHnd)`というがあるとします。しかし、このはのでされており、ログをするときには、のどこからびされているのかをることがのです。

きコンパイルをすると、をするインクルードファイルにのようなことができます。これにより、バージョンのがデバッグバージョンのにきえられます。プリプロセッサは、`SerOpPluAllRead()`へのびしを、`SerOpPluAllRead_Debug()`ます。ファイルとがされているの2つのがあります。

きコンパイルは、をデバッグバージョンでオーバーライドするかどうかをするためにされます。

```
#if 0
// function declaration and prototype for our debug version of the function.
SHORT   SerOpPluAllRead_Debug(PLUIF *pPif, USHORT usLockHnd, char *aszFilePath, int nLineNo);

// macro definition to replace function call using old name with debug function with
additional arguments.
#define SerOpPluAllRead(pPif,usLock) SerOpPluAllRead_Debug(pPif,usLock,__FILE__,__LINE__)
#else
// standard function declaration that is normally used with builds.
SHORT   SerOpPluAllRead(PLUIF *pPif, USHORT usLockHnd);
#endif
```

これにより、`SerOpPluAllRead()`のバージョンを、の`SerOpPluAllRead()`のファイルにファイルとをするバージョンで`SerOpPluAllRead()`ことができます。

なが**1**つあり ます。このをするファイルには、プリプロセッサがをするためにこのアプローチがされるヘッダファイルがまれているがあります。そうしないと、リンカエラーがされます。

のはのようになります。このソ―スは、プリプロセッサが`SerOpPluAllRead()`のを`SerOpPluAllRead_Debug()`にし、2つのの、がびされたファイルのへのポインタ、およびをむようにリストをするようにしますがされているファイルにされます。

```
#if defined(SerOpPluAllRead)
// forward declare the replacement function which we will call once we create our log.
SHORT    SerOpPluAllRead_Special(PLUIF *pPif, USHORT usLockHnd);

SHORT    SerOpPluAllRead_Debug(PLUIF *pPif, USHORT usLockHnd, char *aszFilePath, int nLineNo)
{
    int iLen = 0;
    char  xBuffer[256];

    // only print the last 30 characters of the file name to shorten the logs.
    iLen = strlen (aszFilePath);
    if (iLen > 30) {
        iLen = iLen - 30;
    }
    else {
        iLen = 0;
    }

    sprintf (xBuffer, "SerOpPluAllRead_Debug(): husHandle = %d, File %s, lineno = %d", pPif-
>husHandle, aszFilePath + iLen, nLineNo);
    IssueDebugLog(xBuffer);

    // now that we have issued the log, continue with standard processing.
    return SerOpPluAllRead_Special(pPif, usLockHnd);
}

// our special replacement function name for when we are generating logs.
SHORT    SerOpPluAllRead_Special(PLUIF *pPif, USHORT usLockHnd)
#else
// standard, normal function name (signature) that is replaced with our debug version.
SHORT   SerOpPluAllRead(PLUIF *pPif, USHORT usLockHnd)
#endif
{
    if (STUB_SELF == SstReadAsMaster()) {
        return OpPluAllRead(pPif, usLockHnd);
    }
    return OP_NOT_MASTER;
}
```

ソ―スファイルのインクル―ド

`#include`ディレクティブのもなは、のとおりです。

```
#include <stdio.h>
#include "myheader.h"
```

`#include`は、ステ―トメントをされたファイルのできえます。<>はシステムにインスト―ルされているヘッダ―ファイルをし、""はユ―ザ―のファイルをします。

ののように、マクロはのマクロをすることができます。

```
#if VERSION == 1
    #define INCFILE  "vers1.h"
#elif VERSION == 2
    #define INCFILE  "vers2.h"
    /*  and so on */
#else
    #define INCFILE  "versN.h"
#endif
/* ... */
#include INCFILE
```

マクロ

マクロのもなは、のようにmanifest constantをすることです。

```
#define ARRSIZE 100
int array[ARRSIZE];
```

これは、に10をしてしいをする、 のようなマクロをします。

```
#define TIMES10(A) ((A) *= 10)

double b = 34;
int c = 23;

TIMES10(b);   // good: ((b) *= 10);
TIMES10(c);   // good: ((c) *= 10);
TIMES10(5);   // bad:  ((5) *= 10);
```

は、プログラムテキストのののにわれます。 TIMES10へののびしでは、からのAがbきえられ、されたテキストがびしのわりにかれます。このTIMES10は、

```
#define TIMES10(A) ((A) = (A) * 10)
```

これはAのを2することができるため、ましくないをうがあるからです。

は、がのであるマクロをします。これは、びしのオーバーヘッドなしで、ののあるをし、インラインコードをするというがあります。そのの1つまたは2つを2をむし、びしたによりくのコードをするというがあります。

```
#define max(a, b) ((a) > (b) ? (a) : (b))

int maxVal = max(11, 43);              /* 43 */
int maxValExpr = max(11 + 36, 51 - 7); /* 47 */

/* Should not be done, due to expression being evaluated twice */
int j = 0, i = 0;
int sideEffect = max(++i, ++j);       /* i == 4 */
```

このため、そのようなマクロは、はコードではそのをします。 C11、このようなのびしをけることをにする_Genericがあります。

マクロののなは、とのがしくバインドされ、マクロがびされるコンテキストににまることをします。

エラー

プリプロセッサが`#error`ディレクティブをすると、コンパイルはし、インクルードされたメッセージがされます。

```
#define DEBUG

#ifdef DEBUG
#error "Debug Builds Not Supported"
#endif

int main(void) {
    return 0;
}
```

な

```
$ gcc error.c
error.c: error: #error "Debug Builds Not Supported"
```

コードセクションをブロックする#if 0

をしているコードやににするコードのセクションがあるは、ブロックコメントでコメントアウトすることができます。

```
/* Block comment around whole function to keep it from getting used.
 * What's even the purpose of this function?
int myUnusedFunction(void)
{
    int i = 5;
    return i;
}
*/
```

ただし、ブロックコメントでまれたソースコードのソースにブロックスタイルのコメントがある、のブロックコメントのにはしいブロックコメントがになり、コンパイルのがするがあります。

```
/* Block comment around whole function to keep it from getting used.
 * What's even the purpose of this function?
int myUnusedFunction(void)
{
    int i = 5;

    /* Return 5 */
    return i;
}
*/
```

のでは、のの2との '* /'がコンパイラによってされるため、エラーとともにコンパイルされます。よりなは、ブロックするコードのりに`#if 0`ディレクティブをすることです。

```
#if 0
/* #if 0 evaluates to false, so everything between here and the #endif are
 * removed by the preprocessor. */
int myUnusedFunction(void)
{
    int i = 5;
    return i;
}
#endif
```

このは、コードをしてってみたいときは、すべてのコメントをするよりも "#if 0"をするほうがはるかにです。

もう1つのになは、 `#if 0`コードをコメントアウトすることができます。これはコメントではできません。

`#if 0`をするわりに、 `#defined`されていないをすることもできますが、コードがブロックされているをしくしています。たとえば、なデッドコードとわれるがあるは、のがされていれば、なコードに`#if defined(POSSIBLE_DEAD_CODE)`または`#if defined(FUTURE_CODE_REL_020201)`をすることができます。その、そのソースをまたはにするためにるとき、ソースのセクションはにつけることができます。

トークンペースト

トークンペーストでは、2つのマクロをすることができます。たとえば、 `front##back`は`front##back`し`frontback`。 なは、Win32の`<TCHAR.H>`ヘッダーです。Cでは、ワイドをするために`L"string"`をくことができます。ただし、Windows APIでは`UNICODE #define`するだけで、ワイドといののがです。リテラルをするために、 `TCHAR.H`はこれをします

```
#ifdef UNICODE
#define TEXT(x) L##x
#endif
```

ユーザーが`TEXT("hello, world")`をし、UNICODEがされるたびに、Cプリプロセッサは`L`とマクロをします。 `L`は`"hello, world"`とされ`"hello, world"` `L"hello, world"`ます。

みのマクロ

あらかじめされたマクロは、プログラムをするのないCプリプロセッサによってすでにされているマクロです。には

のマクロ

- `__FILE__`は、のソースファイルのファイルリテラルをします。
-

ののは__LINE__ 、
- __DATE__はコンパイルリテラル、
- コンパイルには__TIME__ リテラル。

__func__ ISO / IEC 9899:2011§6.4.2.2というするみもありますが、これはマクロではありません。

__func__は、ののにあるのように、トランスレータによってにされます。

```
static const char __func__[] = "function-name";
```

しました。ここで、 *function-name*は、をむのです。

__func__ __FILE__ 、 __LINE__ __func__ __FILE__ 、 __LINE__ __func__はデバッグのためににです。えば

```
fprintf(stderr, "%s: %s: %d: Denominator is 0", __FILE__, __func__, __LINE__);
```

C99よりのコンパイラは、 __func__サポートすることもあればサポートしないこともありますし、なるをつマクロをつこともできます。たとえば、gccはC89モードで__FUNCTION__をし__FUNCTION__た。

のマクロではのをねることができます

- __STDC_VERSION__されたCのバージョン。これは、で_yyyymmL_ 201112L C11、が199901L C99のために、それは、C89 / C90のためにされていませんでした
- __STDC_HOSTED__ホストのは1 、そうでないは0 。
- __STDC__ 1、はCにします。

そののマクロではない

ISO / IEC 9899:2011§6.10.9.2マクロ

- __STDC_ISO_10646__フォームの_yyyymmL_ えば199712L。このシンボルがされている、Unicodeのセットのすべてのは、 wchar_tのオブジェクトにされたときに、そののいとじをちます。 Unicodeのセットは、ISO / IEC 10646でされているすべてのと、されたおよびのすべてのおよびからされます。のエンコーディングがされている、マクロはされず、されるのエンコーディングはインプリメンテーションです。

- __STDC_MB_MIGHT_NEQ_WC__1は、 wchar_tのエンコーディングでのとしてされるときに、セットのメンバーがそのにしいコードをつがないことをすためのものです。

- __STDC_UTF_16__のことをすことを1、 char16_t UTF-16でされます。のエンコーディングがされている、マクロはされず、されるのエンコーディングはインプリメ

ンテーションです。

- `__STDC_UTF_32__`のことをすることを1、 `char32_t` UTF-32されます。のエンコーディングがされている、マクロはされず、されるのエンコーディングはインプリメンテーションです。

ISO / IEC 9899:2011§6.10.8.3きマクロ

- `__STDC_ANALYZABLE__`Lのへのをすための1。
- `__STDC_IEC_559__`FIEC 60559のへのをすための1。
- `__STDC_IEC_559_COMPLEX__`GIEC 60559のへのをすための1。
- `__STDC_LIB_EXT1__201112L`。 Kチェックインタフェースでされたのサポートをすことをしています。
- `__STDC_NO_ATOMICS__`がタイプ `_Atomic`をむと`<stdatomic.h>`ヘッダをサポートしていないことをすための1。
- `__STDC_NO_COMPLEX__`がまたは`<complex.h>`ヘッダをサポートしていないことをすための1。
- `__STDC_NO_THREADS__`が`<threads.h>`ヘッダをサポートしていないことをすための1。
- `__STDC_NO_VLA__`がまたはにされたをサポートしていないことをすための1。

しはをむ

ほとんどすべてのヘッダファイルはインクルードガードイディオムにうべきです

my-header-file.h

```
#ifndef MY_HEADER_FILE_H
#define MY_HEADER_FILE_H

// Code body for header file

#endif
```

これにより、ので`#include "my-header-file.h"`を`#include "my-header-file.h"`、、などのがられなくなります。のファイルをしてみてください。

header-1.h

```
typedef struct {
    …
} MyStruct;

int myFunction(MyStruct *value);
```

header-2.h

```
#include "header-1.h"

int myFunction2(MyStruct *value);
```

main.c

```
#include "header-1.h"
#include "header-2.h"

int main() {
    // do something
}
```

このコードにはながありますMyStructのなは2されていますが、これはされていません。 1つのヘッダーファイルにのヘッダーファイルがまれているため、コンパイルエラーがするとがになります。わりにヘッダーガードをした

header-1.h

```
#ifndef HEADER_1_H
#define HEADER_1_H

typedef struct {
    …
} MyStruct;

int myFunction(MyStruct *value);

#endif
```

header-2.h

```
#ifndef HEADER_2_H
#define HEADER_2_H

#include "header-1.h"

int myFunction2(MyStruct *value);

#endif
```

main.c

```
#include "header-1.h"
#include "header-2.h"

int main() {
    // do something
}
```

これはにされます

```
#ifndef HEADER_1_H
#define HEADER_1_H

typedef struct {
    …
} MyStruct;
```

```
int myFunction(MyStruct *value);

#endif

#ifndef HEADER_2_H
#define HEADER_2_H

#ifndef HEADER_1_H // Safe, since HEADER_1_H was #define'd before.
#define HEADER_1_H

typedef struct {
    …
} MyStruct;

int myFunction(MyStruct *value);

#endif

int myFunction2(MyStruct *value);

#endif

int main() {
    // do something
}
```

コンパイラが**header-1.h**の 2のインクルードにすると、 `HEADER_1_H`はのインクルードによってにされていました。エルゴ、それはのようなものです。

```
#define HEADER_1_H

typedef struct {
    …
} MyStruct;

int myFunction(MyStruct *value);

#define HEADER_2_H

int myFunction2(MyStruct *value);

int main() {
    // do something
}
```

したがって、コンパイルエラーはありません。

ヘッダーガードのには、のなるがあります。 `HEADER_2_H_`というのもいれ`HEADER_2_H_`、`MY_PROJECT_HEADER_2_H`ようなプロジェクトがまれるもいます。なのは、プロジェクトのファイルにのヘッダーガードがあるように、それにうようにすることです。

のがヘッダーにまれていない、されたはまたはなになります。そのようなは、のユーザーからのをすのにです。くののために、Cライブラリの`FILE`はとなすことができますI / Oのマクロがのをできるように、はではありません。その、 `header-1.h`は`header-1.h`がまれます。

```
#ifndef HEADER_1_H
#define HEADER_1_H

typedef struct MyStruct MyStruct;

int myFunction(MyStruct *value);

#endif
```

にはタグここでは`MyStruct` - これはタグにあり、typedef`MyStruct`のとは`MyStruct` がであり、 `{ … }` がされているがあります。これは、 "の`struct MyStruct`があり、それに`MyStruct`があり`MyStruct` "とされます。

ファイルでは、のをして、をにすることができます。

```
struct MyStruct {
    …
};
```

C11をしている、 `typedef struct MyStruct MyStruct;`りすことができ`typedef struct MyStruct MyStruct;`コンパイルエラ―はしませんが、のバ―ジョンのCではをいます。したがって、このでは、C11をサポ―トするコンパイラでコ―ドがコンパイルされただけであっても、インクル―ドガ―ドイディオムをすることがです。

くのコンパイラはじをつ`#pragma once`をサポ―トしてい`#pragma once`。

my-header-file.h

```
#pragma once

// Code for header file
```

ただし、 `#pragma once`はCのではないため、コ―ドをするとがします。

いくつかのヘッダ―はインクル―ドガ―ドイディオムをしません。 1つのなは、の`<assert.h>`ヘッダ―です。 1つのにめることができます。そのは、ヘッダがインクル―ドされるたびにマクロ`NDEBUG`がされているかどうかによってなります。にはのがあるかもしれません。そのようなは、ごくわずかなものになります。、ヘッダ―はインクル―ドガ―ドイディオムでされているがあります。

FOREACHの

また、コ―ドをみやすくくためにマクロをすることもできます。えば、リンクやリンクのリスト、キュ―などのいくつかのデ―タにして、Cで`foreach`をするためのマクロをできます。

ここにさながあります。

```
#include <stdio.h>
#include <stdlib.h>

struct LinkedListNode
{
    int data;
    struct LinkedListNode *next;
};

#define FOREACH_LIST(node, list) \
     for (node=list; node; node=node->next)

/* Usage */
int main(void)
{
    struct LinkedListNode *list, **plist = &list, *node;
    int i;

    for (i=0; i<10; i++)
    {
         *plist = malloc(sizeof(struct LinkedListNode));
         (*plist)->data = i;
         (*plist)->next = NULL;
         plist          = &(*plist)->next;
    }

    /* printing the elements here */
    FOREACH_LIST(node, list)
    {
        printf("%d\n", node->data);
    }
}
```

このようなデータのインタフェースをし、 `FOREACH`をのようにくことができます。

```
#include <stdio.h>
#include <stdlib.h>

typedef struct CollectionItem_
{
    int data;
    struct CollectionItem_ *next;
} CollectionItem;

typedef struct Collection_
{
    /* interface functions */
    void* (*first)(void *coll);
    void* (*last) (void *coll);
    void* (*next) (void *coll, CollectionItem *currItem);

    CollectionItem *collectionHead;
    /* Other fields */
} Collection;

/* must implement */
void *first(void *coll)
{
    return ((Collection*)coll)->collectionHead;
}
```

```
/* must implement */
void *last(void *coll)
{
    return NULL;
}

/* must implement */
void *next(void *coll, CollectionItem *curr)
{
    return curr->next;
}

CollectionItem *new_CollectionItem(int data)
{
    CollectionItem *item = malloc(sizeof(CollectionItem));
    item->data = data;
    item->next = NULL;
    return item;
}

void Add_Collection(Collection *coll, int data)
{
    CollectionItem **item = &coll->collectionHead;
    while(*item)
        item = &(*item)->next;
    (*item) = new_CollectionItem(data);
}

Collection *new_Collection()
{
    Collection *nc = malloc(sizeof(Collection));
    nc->first = first;
    nc->last  = last;
    nc->next  = next;
    return nc;
}

/* generic implementation */
#define FOREACH(node, collection)                       \
    for (node  = (collection)->first(collection);       \
         node != (collection)->last(collection);        \
         node  = (collection)->next(collection, node))

int main(void)
{
    Collection *coll = new_Collection();
    CollectionItem *node;
    int i;

    for(i=0; i<10; i++)
    {
         Add_Collection(coll, i);
    }

    /* printing the elements here */
    FOREACH(node, coll)
    {
        printf("%d\n", node->data);
    }
}
```

このをするには、データにこれらのをするだけです。

```
1.  void* (*first)(void *coll);
2.  void* (*last) (void *coll);
3.  void* (*next) (void *coll, CollectionItem *currItem);
```

C ++でコンパイルされたC ++コードでCのをするための__cplusplus - name mangling

のいにより、コンパイラがCコンパイラかC ++コンパイラかによって、インクルードファイルがプリプロセッサとなるをすることがあります。

たとえば、やそののは、Cソースファイルでされていますが、C ++ソースファイルでされています。ののについてのをするためにC ++はマングリングまたはをするため、C ++ソースファイルでされるCはリンクエラーをきこします。 C ++コンパイラーは、C ++のけをして、コンパイラーのされたをします。その、C ++コンパイラがCコンパイラにリンクされているときにからのリンクエラーがされません。

Cコンパイラはネームマングリングをいませんが、C ++コンパイラによってされたすべてのラベルまたはにしてC ++コンパイラがうので、コンパイラをにするためにされたプリプロセッサマクロ`__cplusplus`がされました。

CとC ++ののないコンパイラのこのをするために、マクロ`__cplusplus`はC ++プリプロセッサでされ、Cプリプロセッサでされていません。このマクロは、ソースコードまたはインクルードファイルがC ++またはCとしてコンパイルされているかどうかをするために、きプリプロセッサ`#ifdef`ディレクティブまたは`#if`と`defined()`でできます。

```
#ifdef __cplusplus
printf("C++\n");
#else
printf("C\n");
#endif
```

または、あなたがうことができる

```
#if defined(__cplusplus)
printf("C++\n");
#else
printf("C\n");
#endif
```

C ++ソースファイルでされているCコンパイラでコンパイルされたCソースファイルからのしいをするには、`__cplusplus`をべて`extern "C" { /* ... */ };`ヘッダーファイルがC ++ソースファイルにまれているときにCをするためにされます。しかし、Cコンパイラでコンパイルすると、`extern "C" { */ ... */ };`されていません。このきコンパイルがなのは、`extern "C" { /* ... */ };` C ++ではですが、Cではではありません。

```
#ifdef __cplusplus
// if we are being compiled with a C++ compiler then declare the
// following functions as C functions to prevent name mangling.
extern "C" {
#endif

// exported C function list.
int foo (void);

#ifdef __cplusplus
// if this is a C++ compiler, we need to close off the extern declaration.
};
#endif
```

のようなマクロ

のようなマクロは`inline`にていますが、なデバッグログなどのにです。

```
#ifdef DEBUG
# define LOGFILENAME "/tmp/logfile.log"

# define LOG(str) do {                              \
  FILE *fp = fopen(LOGFILENAME, "a");              \
  if (fp) {                                         \
    fprintf(fp, "%s:%d %s\n", __FILE__, __LINE__, \
                /* don't print null pointer */    \
                str ?str :"<null>");                \
    fclose(fp);                                     \
  }                                                 \
  else {                                            \
    perror("Opening '" LOGFILENAME "' failed");    \
  }                                                 \
} while (0)
#else
  /* Make it a NOOP if DEBUG is not defined. */
# define LOG(LINE) (void)0
#endif


#include <stdio.h>

int main(int argc, char* argv[])
{
    if (argc > 1)
        LOG("There are command line arguments");
    else
        LOG("No command line arguments");
    return 0;
}
```

の `DEBUG`とそうでない、びしは`void`りのとじようにし`void`。これにより、 `if/else`がどおりにされるようになります。

`DEBUG`、これは`do { ... } while(0)do { ... } while(0)`ってされます。それの、 `(void)0`はのないステートメントであり、されます。

のは

```
#define LOG(LINE) do { /* empty */ } while (0)
```

それはすべてのにおいてのものとにである。

GCCをするは、のGNUをしてをすマクロをすることもできます。えば

```
#include <stdio.h>

#define POW(X, Y) \
({ \
        int i, r = 1; \
        for (i = 0; i < Y; ++i) \
                r *= X; \
        r; \ // returned value is result of last operation
})

int main(void)
{
        int result;

        result = POW(2, 3);
        printf("Result: %d\n", result);
}
```

マクロ

C99

をつマクロ

コードをデバッグするためのプリントマクロをしたいとします。このマクロをにげてみましょう

```
#define debug_print(msg) printf("%s:%d %s", __FILE__, __LINE__, msg)
```

のいくつかの

`somefunc()`は、したは-1をし、したは0をし、コードのさまざまなからびされます。

```
int retVal = somefunc();

if(retVal == -1)
{
    debug_printf("somefunc() has failed");
}

/* some other code */

 retVal = somefunc();

if(retVal == -1)
{
    debug_printf("somefunc() has failed");
```

```
}
```

`somefunc()`がされ、なるのあるエラ―タイプにするなるがされるとどうなりますかそれでも、デバッグマクロをしてエラ―をするがあります。

```
debug_printf(retVal);      /* this would obviously fail */
debug_printf("%d",retVal); /* this would also fail */
```

このをするために、 `__VA_ARGS__`マクロがされました。このマクロはのパラメ―タXマクロをします

```
 #define debug_print(msg, ...) printf(msg, __VA_ARGS__) \
                               printf("\nError occurred in file:line (%s:%d)\n", __FILE__,
__LINE)
```

```
int retVal = somefunc();

debug_print("retVal of somefunc() is-> %d", retVal);
```

このマクロでは、のパラメ―タをしてすることができますが、ではパラメ―タのをしています。

```
debug_print("Hey");
```

これは、マクロがなくともも う1つのをとするため、 `debug_print()`は`debug_print()`マクロのカンマのをしないため、エラ―がします。また、 `debug_print("Hey",);`マクロにされたをにしておくことができないので、エラ―がします。

これをするために、 `##__VA_ARGS__`マクロがされました。このマクロはがしない、コ―ドからプリプロセッサによってコンマがされることをしています。

```
 #define debug_print(msg, ...) printf(msg, ##__VA_ARGS__) \
                               printf("\nError occured in file:line (%s:%d)\n", __FILE__,
__LINE)
```

```
 debug_print("Ret val of somefunc()?");
 debug_print("%d",somefunc());
```

オンラインでプリプロセッサとマクロをむ https://riptutorial.com/ja/c/topic/447/プリプロセッサとマクロ

26: プロセスIPC

き

プロセスIPCメカニズムは、なるしたプロセスがいにすることをにする。Cは、IPCメカニズムをしません。したがって、そのようなメカニズムはすべてホストオペレーティングシステムによってされます。 POSIXはなIPCメカニズムをしています。 Windowsはのセットをします。のシステムではのバリアントがされています。

Examples

セマフォ

セマフォは、2つのプロセスでをさせるためにされます。 POSIXでは、セマフォの2つのなるセットをしています。

1. 'System V IPC' - `semctl()` 、 `semop()` 、 `semget()` です。
2. 'POSIXセマフォ' - `sem_close()` 、 `sem_destroy()` 、 `sem_getvalue()` 、 `sem_init()` 、 `sem_open()` 、 `sem_post()` 、 `sem_trywait()` 、 `sem_unlink()`

このでは、System V IPCセマフォについてします。これは、Unix System Vでまったためです。

まず、なヘッダーをめるがあります。いバージョンのPOSIXでは、 `#include <sys/types.h>` でした。のPOSIXとほとんどのシステムはそれをとしません。

```
#include <sys/sem.h>
```

に、とのでキーをするがあります。

```
#define KEY 0x1111
```

このは、のプログラムでじであるがあります。または、じIPCをしません。そのをハードコーディングせずにしたキーをするがあります。

に、コンパイラによっては、このをするがあるとしないがあります。セマフォのでをします。

```
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};
```

に、あなたのみ `semwait` およびげる `semsignal` のを。PおよびVはオランダ

```
struct sembuf p = { 0, -1, SEM_UNDO}; # semwait
struct sembuf v = { 0, +1, SEM_UNDO}; # semsignal
```

さて、あなたのIPCセマフォのIDをすることからめましょう。

```
int id;
// 2nd argument is number of semaphores
// 3rd argument is the mode (IPC_CREAT creates the semaphore set if needed)
if ((id = semget(KEY, 1, 0666 | IPC_CREAT) < 0) {
    /* error handling code */
}
```

では、カウンタが1になるようにセマフォをします。

```
union semun u;
u.val = 1;
if (semctl(id, 0, SETVAL, u) < 0) { // SETVAL is a macro to specify that you're setting the
value of the semaphore to that specified by the union u
    /* error handling code */
}
```

これで、にじてセマフォをデクリメントまたはインクリメントできます。クリティカルセクションのに、 `semop()` をしてカウンタをらします。

```
if (semop(id, &p, 1) < 0) {
    /* error handling code */
}
```

セマフォをインクリメントするには、 `&p` わりに `&p &v` をします。

```
if (semop(id, &v, 1) < 0) {
    /* error handling code */
}
```

すべてのは、すれば `0` をし、したは `-1` をします。これらのりをチェックしないと、ながするがあります。

1.1 スレッドをしたレース

のプログラムでは、プロセスを `fork` 、とのがせずにをにしようとします。

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int main()
{
    int pid;
    pid =  fork();
    srand(pid);
```

```
    if(pid < 0)
    {
        perror("fork"); exit(1);
    }
    else if(pid)
    {
        char *s = "abcdefgh";
        int l = strlen(s);
        for(int i = 0; i < l; ++i)
        {
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
        }
    }
    else
    {
        char *s = "ABCDEFGH";
        int l = strlen(s);
        for(int i = 0; i < l; ++i)
        {
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
        }
    }
}
```

1

```
aAABaBCbCbDDcEEcddeFFGGHHeffgghh
```

2

```
aabbccAABddBCeeCffgDDghEEhFFGGHH
```

このプログラムをコンパイルしてすると、なるがられるはずです。

1.2セマフォーによるレースの

セマフォーをするように*1.1*をすると、

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
#define KEY 0x1111

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};

struct sembuf p = { 0, -1, SEM_UNDO};
struct sembuf v = { 0, +1, SEM_UNDO};

int main()
{
    int id = semget(KEY, 1, 0666 | IPC_CREAT);
    if(id < 0)
    {
        perror("semget"); exit(11);
    }
    union semun u;
    u.val = 1;
    if(semctl(id, 0, SETVAL, u) < 0)
    {
        perror("semctl"); exit(12);
    }
    int pid;
    pid =  fork();
    srand(pid);
    if(pid < 0)
    {
        perror("fork"); exit(1);
    }
    else if(pid)
    {
        char *s = "abcdefgh";
        int l = strlen(s);
        for(int i = 0; i < l; ++i)
        {
            if(semop(id, &p, 1) < 0)
            {
                perror("semop p"); exit(13);
            }
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
            putchar(s[i]);
            fflush(stdout);
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(14);
            }

            sleep(rand() % 2);
        }
    }
    else
    {
        char *s = "ABCDEFGH";
        int l = strlen(s);
        for(int i = 0; i < l; ++i)
        {
            if(semop(id, &p, 1) < 0)
```

```
            {
                perror("semop p"); exit(15);
            }
            putchar(s[i]);
            fflush(stdout);
            sleep(rand() % 2);
            putchar(s[i]);
            fflush(stdout);
            if(semop(id, &v, 1) < 0)
            {
                perror("semop p"); exit(16);
            }

            sleep(rand() % 2);
        }
    }
}
```

```
aabbAABBCCccddeeDDffEEFFGGHHgghh
```

このプログラムをコンパイルしてすると、じがられます。

オンラインでプロセスIPCをむ https://riptutorial.com/ja/c/topic/10564/プロセス-ipc-

27: ヘッダファイルのとインクルード

き

のCでは、ヘッダ—ファイルがしくされ、しくされなければならないなツ—ルです。コンパイラは、プログラムのしてコンパイルされたをクロスチェックすることができます。

ヘッダは、ののがとするタイプ、、マクロなどをします。これらのをするすべてのコ—ドにはヘッダ—がまれています。これらのをするすべてのコ—ドにヘッダ—がまれています。これにより、コンパイラはとがするかどうかをできます。

Examples

き

Cプロジェクトでヘッダファイルをしてするには、いくつかのガイドラインがあります。

- ヘッダ—ファイルがTUにまれているは、ビルドをしてはなりません。
- じめ

  ヘッダ—ファイルでされたがなは、のヘッダ—をにめるはありません。
- ビルドにすることなくヘッダからをすることはできません。
- あなたがするものをめるIWYU

  CよりもC ++へのがいが、Cでもである。 TUのコ—ド `code.c`とふがヘッダ `"headerA.h"`とふによってされたをする、TUにまれていても`code.c`は`#include "headerA.h"``#include "headerA.h"`がありますでは`"headerA.h"`をむのヘッダ `"headerB.h"`とふを`"headerA.h"`。

によっては、これらのガイドラインの1つをるのにながあるかもしれませんが、ル—ルをっていて、それをるにそのをっていることにづくべきです。

のヘッダ—ファイルがTUにまれている、コンパイルのはないはずです。これは「」とばれます。あなたのヘッダ—はでなければなりません。 `#include <stdio.h>`がだけまれていることをしなければならない、どのくらいしいになるかえてみてください。

をするには、ヘッダ—ガ—ドと`#pragma once`という2つのがあります。

ヘッダ—ガ—ド

ヘッダ—ガ—ドはシンプルでがく、Cにしています。ヘッダ—ファイルののコメントは、のであるがあります。

```
#ifndef UNIQUE_ID_FOR_HEADER
#define UNIQUE_ID_FOR_HEADER
```

のコメントは、 `#endif`あります。

```
#endif /* UNIQUE_ID_FOR_HEADER */
```

の`#include`をむすべてのコードは、これらののにあるがあります。

それぞれのはでなければなりません。くの、 `HEADER_H_INCLUDED`などのがされます。いコードのには、ヘッダガードとしてされているシンボル `<stdio.h> #ifndef BUFSIZ` をしていますが、のほどがありません。

1つのオプションは、ヘッダガードにされたMD5またはのハッシュをすることです。にされているをにするシステムヘッダーでされているスキームをエミュレートしないでください。アンダースコアでまるにいて、のアンダースコアまたはがきます。

`#pragma once`

わりに、 `#pragma once`ディレクティブをサポートするコンパイラもあり`#pragma once`が、これはヘッダガードの3とじがあります。

```
#pragma once
```

`#pragma once`をサポートするコンパイラには、MS Visual StudioとGCCとClangがまれています。しかし、がされるは、ヘッダーガードをするか、をするがよいでしょう。のコンパイラC89をサポートするコンパイラは、らないプラグマをするがあります'によってされないプラグマはされます'が、GCCのいバージョンはそんなにやかされませんでした。

じめ

のヘッダーはでなければなりません。つまり、 `header.h`されたをするのあるプログラムには、そのヘッダー `#include "header.h"` をめることができ、のヘッダーをにめなければならないかどうかはしません。

ヘッダーファイルはでなければなりません。

ルール

に、これはにのです。

ATT Indian Hill Cのスタイルとコーディングでは、

> ヘッダーファイルはネストしないでください。したがって、ヘッダーファイルのプロ

ローグは、ヘッダーがするためにのヘッダーが`#include`するがあるものをするがあります。なには、のヘッダファイルをのなるソースファイルにめるには、すべての`#include`を1つのインクルードファイルにれることがされます。

これはじめのです。

のルール

しかし、それ、はのにいていた。ソースファイルがヘッダ`header.h`によってされたをするがある、プログラマはをくことができるはずです

```
#include "header.h"
```

コマンドにしいパスがされているのみ、ヘッダーがソースファイルにさ`header.h`ことなく`header.h`になヘッダーがすべてまれます。

これにより、ソースコードのモジュールがします。また、コードがされ、10または2にわたってハッキングされたにする「このヘッダーがされた」のからソースをします。

NASAのゴダードセンターGSFCは、Cのためのコーディングは、よりなのつですが、するのがしいです。ヘッダーはでなければならないとかれています。また、ヘッダーがであることをするなもされます。ヘッダーのファイルにはヘッダーがのヘッダーとしてめるがあります。でない、そのコードはコンパイルされません。

GSFCがしたはのとおりです。

> §2.1.1ヘッダーはをむ
>
> このでは、ユニットヘッダがとするのてのヘッダのための`#include`をユニットのヘッダにめるがあります。ユニットヘッダに`#include`をにすると、コンパイラはヘッダになすべての`#include`がまれていることをできます。
>
> このではされていないのでは、ヘッダーに`#include`をすることはできません。すべての`#includes`はファイルでわれます。ユニットヘッダーファイルには、なヘッダーがなでまれていることをする#ifdefステートメントがまれているがあります。
>
> のの1つは、ファイルの`#include`リストがmakefileになリストであり、このリストがコンパイラによってチェックされていることです。では、ツールをしてリストをするがあります。しかし、ブランチのはすべて、このようなツールをします。
>
> のなは、ユニットのヘッダーリストがされた、そのユニットをするファイルをして、`#include`ステートメントリストをするがあることです。また、コンパイラライブラリユニットになヘッダリストは、なるターゲットでなるがあります。
>
> ののもうつのは、コンパイラライブラリのヘッダファイルやのサードパーティのファイルをして、な`#ifdef`をするがあることです。

したがって、じめとは、

- ヘッダ―header.hしいネストされたヘッダのextra.h 、あなたがするすべてのソースファイルをチェックするはありませんheader.hするがあるかどうかをするためにextra.h。
- ヘッダ―header.hもはやのヘッダ―めるがnotneeded.h 、あなたがするすべてのソースファイルをチェックするはありませんheader.hあなたはにりすことができるかどうかするためにnotneeded.h ただし、、あなたがしているものをめるを 。
- のヘッダ―をめるためのしいをするはありませんジョブをにするためにトポロジカルなべえがです。

じめの

chkhdrとヘッダ―ファイルのをテストするためにできるスクリプトchkhdr ライブラリとのリンクをしてください。

ヘッダ―はなチェックメカニズムですが、なりさくするがあります。に、ファイルにはのヘッダ―がなため、ヘッダ―にのヘッダ―をめないでください。ヘッダ―には、されているサ―ビスのコンシュ―マ―にとってなヘッダ―のみがまれているがあります。

たとえば、ヘッダの1つがFILE *または<stdio.h>のみでされたのの1つをしないり、プロジェクトヘッダには<stdio.h>めないでください。インタフェ―スがsize_tする、なのヘッダ―は<stddef.h>です。らかに、 size_tをするのヘッダ―がまれている、 <stddef.h>もめるはありません。

ヘッダ―がの、コンパイルもにえられます。

のくのヘッダ―をめることをのとするヘッダ―をすることはです。すべてのヘッダ―でされているすべてのがになソースファイルはほとんどないため、これらはほとんどにえません。えば、ヘッダがにするとはらないので、Cヘッダをすべてむ<standard-ch>をすることができます。しかし、<locale.h>または<tgmath.h>のをにするプログラムはごくわずかです。

- Cでのファイルをリンクするもごください。

あなたがするものをめるIWYU

Googleのプロジェクトのインクルード IWYUは、ソースファイルにコードでされているすべてのヘッダ―がまれるようにします。

ソースファイルsource.cに、 freeloader.hにむヘッダarbitrary.hがまれているとしますが、ソースファイルもfreeloader.hのをかつしてします。すべてがうまくまります。その、あるarbitrary.hがされ、クライアントはもはやfreeloader.hのをとしなくなります。、 source.cコンパイルを-それはIWYUをたしていなかったので。のコードなのでsource.cにfreeloader.hがなされているがあります- 、それがするものがまれているがある#include "freeloader.h"あまりにもソースインチ はがないことをしたはずです。

IWYUのは、インタ―フェイスになをえてもコードがコンパイルされけるをにします。らかに、

あなたのコ―ドがされたインタ―フェイスからでされるをびす、のはがになるのをぐことができません。このため、なはAPIへのをけ、のリリ―スなどではサイクルがあるです。

これはなヘッダ―がいにめることができるので、C ++ののです。ソ―スファイル`file.cpp`は、あるプラットフォ―ムにのヘッダ`header2.h`む1つのヘッダ`header1.h`をむことができる。 `file.cpp`のがするかもしれない`header2.h`。これははではないでしょう - `header1.h`は`header2.h`まれているため、コ―ドはコンパイルされ`header2.h` 。のプラットフォ―ム、またはのプラットフォ―ムのアップグレ―ドでは、 `header1.h`それはもはやまれてすることはできなかった`header2.h`し、その`file.cpp` 、として、コンパイルをするでしょう。

IWYUはを`header2.h` 、 `file.cpp`に`header2.h`ことをします。これによりコンパイルがされます。 Cコ―ドにものがされます。

およびその

Cのでは、 `#include <header.h>`と`#include "header.h"`にはほとんどいがないとわれています。

> [`#include <header.h>`]は、 `<`と`>`りのでされたシ―ケンスによってにされるヘッダ―にして、ののシ―ケンスをし、ヘッダ―のでそのディレクティブをきえます。どのようにがされるか、またはヘッダ―がされるかはです。
>
> [`#include "header.h"`]は、されたシ―ケンスによってされるソ―スファイルのが"…"デリミタのできえられます。されたソ―スファイルは、のでされます。このがサポ―トされていない、またはがした、ディレクティブは[`#include <header.h>`]とじようにされます。

したがって、でまれたフォ―ムは、カッコでまれたフォ―ムよりもくのでされることがあります。では、をするはコンパイルがしますが、のヘッダ―をかっこにめるようにしています。に、POSIXなどのではきのをしています。プロジェクトでされたヘッダ―のヘッダ―をします。ヘッダ―プロジェクトがするのプロジェクトのヘッダ―をむでは、がです。

コンパイラはスペ―スをけけなくても、 `#include`とヘッダのにスペ―スがであることにしてください。スペ―スはいです。

くのプロジェクトでは、のようながされています。

```
#include <openssl/ssl.h>
#include <sys/stat.h>
#include <linux/kernel.h>
```

プロジェクトでそのコントロ―ルをするかどうかをするがありますおそらくよいえです。のプロジェクトでされているをにするがありますに、 `sys`と`linux`がいになります。

これをするは、コ―ドをくし、のをさせるがあります。

`#include "../include/header.h"`をしないでください。

ヘッダーファイルは、をすることはめったにありません。グローバルはにえますが、グローバルがなはヘッダーにし、なソースファイルでそのをします。ソースファイルにはとをクロスチェックするヘッダーがまれますをするすべてのソースファイルはヘッダをしてします。

ソースファイルにグローバルをしません。ソースファイルにはのみがまれます。

ヘッダーファイルは`static`をすべきではありません。 `static inline`は、のソースファイルでがなにヘッダーでされます。

- ソースファイルは、グローバルとグローバルをします。
- ソースファイルはグローバルやのをしません。またはをするヘッダーがまれます。
- ヘッダーファイルは、グローバルとおよびとそののサポートをします。
- ヘッダーファイルでは、または`static inline`のはされません。

クロスリファレンス

- Cでをするは
- CおよびC ++でのヘッダファイルのリスト
- C99で`static`または`extern`なしで`inline`でえますか
- `extern`をしてソースファイルでをするにはどうすればよいですか
- ヘッダに`"../include/header.h"`ようなパスのメリットはありますか
- ヘッダの
- すべてのヘッダーをめるがありますか

オンラインでヘッダファイルのとインクルードをむ https://riptutorial.com/ja/c/topic/6257/ヘッダファイルのとインクルード

28: ポインタ

き

ポインタとは、のオブジェクトやのアドレスをできるのです。

- <データ> * <>;
- int * ptrToInt;
- void * ptrToVoid; / * C89 + * /
- struct someStruct * ptrToStruct;
- int ** ptrToPtrToInt;
- int arr [length]; int * ptrToFirstElem = arr; / * <C99の、 'length'はコンパイルでなければなりません。> = C11のは1にするがあります。 * /
- int * arrayOfPtrsToInt [length]; / * <C99の、 'length'はコンパイルでなければなりません。> = C11のは1にするがあります。 * /

アスタリスクのは、のにはしません。

```
/* The * operator binds to right and therefore these are all equivalent. */
int *i;
int * i;
int* i;
```

ただし、にのポインタをするは、それぞれにのアスタリスクがです。

```
int *i, *j; /* i and j are both pointers */
int* i, j;  /* i is a pointer, but j is an int not a pointer variable */
```

ポインター—のもです。アスタリスクはののにいています

```
int *foo[2]; /* foo is a array of pointers, can be accessed as *foo[0] and *foo[1] */
```

Examples

なエラー—

ポインタのなは、くの、セグメンテーションのためにセキュリティバグやプログラムクラッシュをむがあるバグのとなります。

りてのをチェックしない

メモリりてはするとはされず、わりにNULLポインタをすことがありNULL。りをして、りてがしたかどうかをせずに、のをびします。これはクラッシュにつながりますが、クラッシュがするとい

うはなく、そのことにってもにつながるがあります。

たとえば、でない

```
struct SomeStruct *s = malloc(sizeof *s);
s->someValue = 0; /* UNSAFE, because s might be a null pointer */
```

な

```
struct SomeStruct *s = malloc(sizeof *s);
if (s)
{
    s->someValue = 0; /* This is safe, we have checked that s is valid */
}
```

メモリをするときに**sizeof**のわりにリテラルをう

えられたコンパイラ/マシンでは、のサイズはです。ただし、 `char`のされたのサイズは、すべてのコンパイラ/マシンでじになることをするはありません。コードがメモリりてに`sizeof(int)`わりに4をすると、のマシンでするがありますが、コードはずしものマシンやコンパイラにするはありません。のサイズは、 `sizeof(that_type)`または`sizeof(*var_ptr_to_that_type)`きえてください。

なりて

```
int *intPtr = malloc(4*1000);    /* allocating storage for 1000 int */
long *longPtr = malloc(8*1000);  /* allocating storage for 1000 long */
```

ポータブルりて

```
int *intPtr = malloc(sizeof(int)*1000);     /* allocating storage for 1000 int */
long *longPtr = malloc(sizeof(long)*1000);  /* allocating storage for 1000 long */
```

あるいは、よりいことには

```
int *intPtr = malloc(sizeof(*intPtr)*1000);     /* allocating storage for 1000 int */
long *longPtr = malloc(sizeof(*longPtr)*1000);  /* allocating storage for 1000 long */
```

メモリリーク

`free`をしてメモリのりてをしないと、プログラムではされなくなったなメモリがされます。これはメモリリークとばれます 。メモリリークはメモリリソースをし、りてのにつながります。

エラー

すべてのりてはじパターンにわなければなりません

1. `malloc` または`calloc` をしたりて
2. データをするための
3. `free`りて

`free` ダングリングポインター や`malloc` ワイルドポインター のびしのにメモリーをする、 `free` 2びす「ダブルフリー」など、このパターンをしないと、セグメンテーションがし、としてプログラムがクラッシュします。

これらのエラーはであり、デバッグがしいがあります。たとえば、されたメモリは、OSによってにされません。したがって、しばらくの、ポインタがになり、するようにえます。

それがするシステムでは、 Valgrindは、メモリがリークされたと、にりてられたをするためのなツールです。

スタックへのポインタの

ポインタをしても、しすのはびません。えば

```
int* myFunction()
{
    int x = 10;
    return &x;
}
```

ここで、 `x`は にスタックりてとばれます をちます 。これはスタックにりてられているため、`myFunction`がされているだけです。 `myFunction`がすると、`x`がされます。このは、 `x`のアドレスをし`&x`を、それをびしにし、びしにしないへのポインタをします。このにアクセスしようとすると、のがします。

ほとんどのコンパイラは、がしたににスタックフレームをクリアしないので、されたポインタのは、しばしばされるデータをえます。ただし、のがびされると、しているメモリがきされ、しされているデータがしているようにえます。

これをするには、されるのを`malloc`し、しくされたへのポインタをすか、をすのではなくになポインタをすがあります。

```
#include <stdlib.h>
#include <stdio.h>

int *solution1(void)
{
    int *x = malloc(sizeof *x);
    if (x == NULL)
    {
        /* Something went wrong */
        return NULL;
    }

    *x = 10;
```

```
    return x;
}

void solution2(int *x)
{
    /* NB: calling this function with an invalid or null pointer
       causes undefined behaviour. */

    *x = 10;
}

int main(void)
{
    {
        /* Use solution1() */

        int *foo = solution1();
        if (foo == NULL)
        {
            /* Something went wrong */
            return 1;
        }

        printf("The value set by solution1() is %i\n", *foo);
        /* Will output: "The value set by solution1() is 10" */

        free(foo);    /* Tidy up */
    }

    {
        /* Use solution2() */

        int bar;
        solution2(&bar);

        printf("The value set by solution2() is %i\n", bar);
        /* Will output: "The value set by solution2() is 10" */
    }

    return 0;
}
```

*と

あなたがく`*p++`によってされるものインクリメントする`p`、あなたはっています。

のに、ポストインクリメント/デクリメントがされます。したがって、このはポインタのインクリメントします`p`をしてによってされたものをす`p`、それをすることなく、インクリメントするに。

あなたはくべき`(*p)++`によってされたものをする`p`。

このルールは、デクリメントをする`*p--`ポインタのデクリメントします`p`によってされているもの、それではなく`p`。

ポインタをする

```
int a = 1;
int *a_pointer = &a;
```

a_pointerをし、aのをするには、のをします。

```
*a_pointer = 2;
```

これは、のprintをしてできます。

```
printf("%d\n", a); /* Prints 2 */
printf("%d\n", *a_pointer); /* Also prints 2 */
```

しかし、 NULLポインタまたはのでなポインタをするとされることになります。この

```
int *p1, *p2;

p1 = (int *) 0xbad;
p2 = NULL;

*p1 = 42;
*p2 = *p1 + 1;
```

はのです。 p1は、なアドレスではないのあるアドレス0xbadをしているため、されないことがあります。がそこにあるのをがっていますかオペレーティングシステムのメモリまたはのプログラムのメモリであるがあります。このようなのタイムコードは、ハードコードされたアドレスにのをするめみでされています。 p2はNULLであるため、することはできません。これはです。

へのポインタをする

たちはのをっているとしましょう

```
struct MY_STRUCT
{
    int my_int;
    float my_float;
};
```

たちはstructキーワードをするようにMY_STRUCTをすることができるので、するたびにstruct MY_STRUCTをタイプするはありません。ただしこれはオプションです。

```
typedef struct MY_STRUCT MY_STRUCT;
```

こののインスタンスへのポインタがあれば

```
MY_STRUCT *instance;
```

このがファイルスコープにれると、プログラムがするときにinstanceがヌルポインタでされます。このステートメントがにある、そのはです。は、されるに、なMY_STRUCTをすように、またはに

りられたスペ―スをすようにするがあります。えば

```
MY_STRUCT info = { 1, 3.141593F };
MY_STRUCT *instance = &info;
```

ポインタがな、2つのなるの1つをしてポインタをしてメンバ―にアクセスできます。

```
int a = (*instance).my_int;
float b = instance->my_float;
```

どちらのメソッドもしますが、かっこ、`*`、およびドットのみわせではなく、arrow `->`をするがい. なぜなら、にネストされたでは、みやすくしやすいからです。

もう1つのないをにします。

```
MY_STRUCT copy = *instance;
copy.my_int    = 2;
```

この、 `copy`には`instance`ののコピ―がまれ`instance`。 `copy` `my_int`をしても`instance`はされません。

```
MY_STRUCT *ref = instance;
ref->my_int    = 2;
```

この、 `ref`は`instance`へのです。 をして`my_int`をすると、 `instance`がされ`instance`。

ではなく、のパラメ―タとしてへのポインタをするのがです。 をパラメ―タとしてすると、がきいにスタックがオ―バ―フロ―するがあります。 へのポインタをすると、ポインタになスタックしかされませんが、がにされたをしたにをきこすがあります。

ポインタ

ポインター―をしてをポイントすることもできます。

なをりましょう

```
int my_function(int a, int b)
{
    return 2 * a + 3 * b;
}
```

さて、ののポインタをしましょう

```
int (*my_pointer)(int, int);
```

するには、このテンプレ―トをしてください

```
return_type_of_func (*my_func_pointer)(type_arg1, type_arg2, ...)
```

に、このポインタをにりてるがあります。

```
my_pointer = &my_function;
```

このポインタをしてをびすことができるようになりました

```
/* Calling the pointed function */
int result = (*my_pointer)(4, 2);

...

/* Using the function pointer as an argument to another function */
void another_function(int (*another_pointer)(int, int))
{
    int a = 4;
    int b = 2;
    int result = (*another_pointer)(a, b);

    printf("%d\n", result);
}
```

このは、とよりでしているようにえますが、とのポインタは&と*のをとしません。のスニペットもじようにです。

```
/* Attribution without the & operator */
my_pointer = my_function;

/* Dereferencing without the * operator */
int result = my_pointer(4, 2);
```

ポインタのをめるために、typedefをすることができます。

```
typedef void (*Callback)(int a);

void some_function(Callback callback)
{
    int a = 4;
    callback(a);
}
```

もう1つののトリックは、Cのでは、プロトタイプのようなものにのようなただしではないのポインタをすることができます。したがって、はとににできます。

```
void some_function(void callback(int))
{
    int a = 4;
    callback(a);
}
```

もしてください

ポインタ

ポインタの

ポインタのは、ワイルドポインタをけるためのいです。はであり、のとじです。

```
#include <stddef.h>

int main()
{
    int *p1 = NULL;
    char *p2 = NULL;
    float *p3 = NULL;

         /* NULL is a macro defined in stddef.h, stdio.h, stdlib.h, and string.h */

    ...
}
```

ほとんどのオペレーティングシステムでは、 NULLされたポインタをってすると、プログラムがすぐにクラッシュし、のをにできます。されていないポインタをすると、がなバグがしばしばします。

あぶない

NULLポインタののはであるため、プログラムがされているオペレーティングシステムのなであっても、 ずしもクラッシュをきこすとはりません 。コンパイラのは、クラッシュをマスクして、ソースコードのヌルポインタがしたポイントのにクラッシュをさせたり、ヌルポインタをむコードのをせずプログラムからしたりすることがあります。デバッグビルドではこれらのはわれませんが、ではされません。のしないおよび/またはましくないもされる。

NULLは、、りてられたメモリ、またはをしていないので、ガードとしてすることはです。

あぶない

、 NULLは(void *)0としてされNULL 。 しかし、これはりてられたメモリアドレスが0x0ことをするものではありません。よりにするために、NULLポインターのC-faqをしてください

NULLのをむようにポインタをすることもできます。

```
int i1;

int main()
{
   int *p1 = &i1;
   const char *p2 = "A constant string to point to";
   float *p3 = malloc(10 * sizeof(float));
}
```

オペレーターのアドレス

のオブジェクト、、、、ポインタまたはの、アドレスをしてそのオブジェクトのアドレスにアクセスできます。

```
int i = 1;
int *p = NULL;
```

したがって、 `p = &i;` `i`アドレスをポインタ`p`にコピーします。

それは`i`を`p` としてされます。

`printf("%d\n", *p);` `i`のである1をします。

ポインタ

ここをクリックしてください ポインタ

としての**void ***ポインタをにす

KR

`void*`は、オブジェクトへのポインタのすべてのをキャッチします。このは、 `malloc`をしています。これは、のようにされています。

```
void* malloc(size_t);
```

voidへのポインタりのは、 `malloc`からのりをのオブジェクトへのポインタにりてることができることをします

```
int* vector = malloc(10 * sizeof *vector);
```

には、をにvoidポインタのにれたりしたりしないことがいです。なキャストでは、 `stdlib.h`をインクルードするのをれてしまった、コンパイラは`malloc()`ったりのをるかもしれないがしないかもしれないので、 `malloc()`のです。 DRYをりさないでくださいのによくうためには、voidポインタのしいをするもあります。とをしてください。のコードには、タイプミスがをきこすのあるなのがいくつかまれています。

```
int* vector = (int*)malloc(10 * sizeof int*);
```

に、

```
void* memcpy(void *restrict target, void const *restrict source, size_t size);
```

にかかわらずオブジェクトのアドレスをすことができるので、 `void *`されたをつことができます

。ここでも、びしはキャスト

```
unsigned char buffer[sizeof(int)];
int b = 67;
memcpy(buffer, &b, sizeof buffer);
```

Constポインタ

ポインタ

- intへのポインタ

  ポインタはなるをすことができ、 intポインタですることができます。コードのこのサンプルは、をすようにBをりてるint b 、にのbへの100 。

  ```
  int b;
  int* p;
  p = &b;    /* OK */
  *p = 100;  /* OK */
  ```

- const intへのポインタ

  ポインタはなるをすことができますが、 intはポインタでできません。

  ```
  int b;
  const int* p;
  p = &b;    /* OK */
  *p = 100;  /* Compiler Error */
  ```

- intへのconstポインタ

  ポインタは1つのintをすことができますが、 intのはポインタをしてできます。

  ```
  int a, b;
  int* const p = &b; /* OK as initialisation, no assignment */
  *p = 100;  /* OK */
  p = &a;    /* Compiler Error */
  ```

- const const intへのconstポインタ

  ポインタは1つのintをすだけで、 intをポインタですることはできません。

  ```
  int a, b;
  const int* const p = &b; /* OK as initialisation, no assignment */
  p = &a;   /* Compiler Error */
  *p = 100; /* Compiler Error */
  ```

ポインター—へのポインター—

- `int`へのポインタへのポインタ

  このコードは、 `p1`のアドレスをdoubleポインタ`p`します `int* p1 int`をしますをします。

  に、 `p1`を`int a`にすように`int a`ます。に、aのを100にします。

```
void f1(void)
{
  int a, b;
  int *p1;
  int **p;
  p1 = &b; /* OK */
  p = &p1; /* OK */
  *p = &a; /* OK */
  **p = 100; /* OK */
}
```

- `const int`へのポインタへのポインタ

```
 void f2(void)
{
  int b;
  const int *p1;
  const int **p;
  p = &p1; /* OK */
  *p = &b; /* OK */
  **p = 100; /* error: assignment of read-only location ‘**p’ */
}
```

- `int`への`const`ポインタへのポインタ

```
void f3(void)
{
  int b;
  int *p1;
  int * const *p;
  p = &p1; /* OK */
  *p = &b; /* error: assignment of read-only location ‘*p’ */
  **p = 100; /* OK */
}
```

- `int`へのポインタへの`const`ポインタ

```
void f4(void)
{
  int b;
  int *p1;
  int ** const p = &p1; /* OK as initialisation, not assignment */
  p = &p1; /* error: assignment of read-only variable ‘p’ */
  *p = &b; /* OK */
  **p = 100; /* OK */
}
```

- `const int`への`const`ポインタへのポインタ

```
void f5(void)
{
  int b;
  const int *p1;
  const int * const *p;
  p = &p1; /* OK */
  *p = &b; /* error: assignment of read-only location ‘*p’ */
  **p = 100; /* error: assignment of read-only location ‘**p’ */
}
```

- `const const int`へのポインタへの`const int`ポインタ

```
void f6(void)
{
  int b;
  const int *p1;
  const int ** const p = &p1; /* OK as initialisation, not assignment */
  p = &p1; /* error: assignment of read-only variable ‘p’ */
  *p = &b; /* OK */
  **p = 100; /* error: assignment of read-only location ‘**p’ */
}
```

- `int`への`const`ポインタへの`const`ポインタ

```
void f7(void)
{
  int b;
  int *p1;
  int * const * const p = &p1; /* OK as initialisation, not assignment */
  p = &p1; /* error: assignment of read-only variable ‘p’  */
  *p = &b; /* error: assignment of read-only location ‘*p’ */
  **p = 100; /* OK */
}
```

じアスタリスク、なる

CとC ++のポインタをりくもしやすいことは、ポインタシンボルであるアスタリスク `*` がとともにされるときにには2つのなるがされるということです。

まず、 `*`をしてポインタをします。

```
int i = 5;
/* 'p' is a pointer to an integer, initialized as NULL */
int *p = NULL;
/* '&i' evaluates into address of 'i', which then assigned to 'p' */
p = &i;
/* 'p' is now holding the address of 'i' */
```

していないときまたはしていないとき、 `*`はポインタのをするためにわれます

```
*p = 123;
/* 'p' was pointing to 'i', so this changes value of 'i' to 123 */
```

のポインタにののアドレスをさせたい、 `*` はしませんが、のようにします

```
p = &another_variable;
```

Cプログラミングののでのは、ポインタをにしてするときにします。

```
int *p = &i;
```

`int i = 5;`そして`int i; i = 5;`じをえ、それらのうちのいくつかは`int *p = &i;`と`int *p; *p = &i;`じをえる。は、いいえ、 `int *p; *p = &i;` されていないポインタをしようとします。このポインタはUBになります。ポインタをしたりデリファレンスしていないときは、 `*`しないでください。

アスタリスク `*` は、ポインタがされているにじて、Cでポインタとので2つのなるをちます。 でする、ののは、メモリのアドレスにするポインタでなければなりません。にされているとにすると、アスタリスクはポインター—をし、ポインター—をメモリにし、そこにされているをまたはできるようにします。

りく

ポインター—をうときは、あなたのPとQをにすることがです。あなたがアスタリスクをしているとき、そしてあなたがそこでそれをするとき、それがをするのかしてください。このさなディテ—ルをとすと、にしたくないバグやのにつながるがあります。

ポインター—へのポインター—

Cでは、ポインタはのポインタをすることができます。

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  int A = 42;
  int* pA = &A;
  int** ppA = &pA;
  int*** pppA = &ppA;

  printf("%d", ***pppA); /* prints 42 */

  return EXIT_SUCCESS;
}
```

しかし、とはされません。

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main(void) {
  int A = 42;
  int* pA = &A;
  int** ppA = &&A; /* Compilation error here! */
  int*** pppA = &&&A;  /* Compilation error here! */

  ...
```

き

ポインタはのとのにアスタリスク * がかれていることをいて、のとにされます。

```
int *pointer; /* inside a function, pointer is uninitialized and doesn't point to any valid
object yet */
```

じの2つのポインタをするには、じで、のにアスタリスクをします。えば、

```
int *iptr1, *iptr2;
int *iptr3,  iptr4;  /* iptr3 is a pointer variable, whereas iptr4 is misnamed and is an int
*/
```

アンパサンド & でされるアドレスまたははは、なのポインタにできるのアドレスをえます。

```
int value = 1;
pointer = &value;
```

アスタリスク * でされるまたははは、ポインタがすオブジェクトのをします。

```
printf("Value of pointed to integer: %d\n", *pointer);
/* Value of pointed to integer: 1 */
```

ポインタがまたはをしている、それをして、 ->をしてそのメンバーにアクセスできます。

```
SomeStruct *s = &someObject;
s->someMember = 5; /* Equivalent to (*s).someMember = 5 */
```

Cでは、ポインタはりてなのであり、それのはそれのとしてわれます。たとえば、のは、ポインタのをします。

```
printf("Value of the pointer itself: %p\n", (void *)pointer);
/* Value of the pointer itself: 0x7ffcd41b06e4 */
/* This address will be different each time the program is executed */
```

ポインタはであるため、なオブジェクトをしていないがあります。これは、nullにされているか

```
pointer = 0;     /* or alternatively */
pointer = NULL;
```

またはになアドレスではないのビットパターンをむことにより、はにいです。なぜなら、ポインタがされるにテストすることはできないからです。ポインタがnullののテストしかありません。

```
if (!pointer) exit(EXIT_FAILURE);
```

ポインタは、 なオブジェクトをしているにのみすることができます。そうでない、ポインタはです。ののくは、 セグメンテーションフォールトやをさせるなどのエラーをさせてけてくれるかもしれませんが、のものはあなたのプログラムをなにしてしまうかもしれません。

によってされるは、のへのなであるため、のをしてすることができます。

```
*pointer += 1;
printf("Value of pointed to variable after change: %d\n", *pointer);
/* Value of pointed to variable after change: 2 */
```

ポインタもりてです。つまり、オブジェクトをすポインターをして、でじタイプののオブジェクトをすことができます。

```
int value2 = 10;
pointer = &value2;
printf("Value from pointer: %d\n", *pointer);
/* Value from pointer: 10 */
```

のとに、ポインタにはのがあります。 `short int`アドレスは`long int`へのポインタにすることはできません。このようなるいはとばれ、Cではされていますが、いくつかのがあります。

ポインターはのでなければなりませんが、ポインターのごとにりてられるメモリーは、しされるのサイズではなく、アドレスをするためににによってされるメモリーとじです。

```
#include <stdio.h>

int main(void) {
    printf("Size of int pointer: %zu\n", sizeof (int*));      /* size 4 bytes */
    printf("Size of int variable: %zu\n", sizeof (int));      /* size 4 bytes */
    printf("Size of char pointer: %zu\n", sizeof (char*));    /* size 4 bytes */
    printf("Size of char variable: %zu\n", sizeof (char));    /* size 1 bytes */
    printf("Size of short pointer: %zu\n", sizeof (short*));  /* size 4 bytes */
    printf("Size of short variable: %zu\n", sizeof (short));  /* size 2 bytes */
    return 0;
}
```

*C99*または*C11*をサポートしていない*Microsoft Visual Studio*をしているは、サンプルで`%zu`わりに`%zu` `%Iu` [1]をするがあります。

のはによってなるがありますが、すべてのがなるタイプのポインタにしてじサイズをすことにしてください。

カーディフのについた

ポインタと

ポインタとはCでにされています。Cのはにメモリのしたにされます。ポインタのはに、しされているのサイズによってスケーリングされます。したがって、3つのdoubleのとへのポインタをつ、`*ptr`はのdoubleをします。 `*(ptr + 1)`は2の`*(ptr + 2)`は3のdoubleです。よりなは、`[]`をうことです。

```
double point[3] = {0.0, 1.0, 2.0};
double *ptr = point;

/* prints x 0.0, y 1.0 z 2.0 */
printf("x %f y %f z %f\n", ptr[0], ptr[1], ptr[2]);
```

したがって、にptrとはがあります。このルールは、がサブルーチンにされるときにポインタにすることもします。

```
double point[3] = {0.0, 1.0, 2.0};

printf("length of point is %s\n", length(point));

/* get the distance of a 3D point from the origin */
double length(double *pt)
{
   return sqrt(pt[0] * pt[0] + pt[1] * pt[1] + pt[2] * pt[2])
}
```

ポインタは、のの、またはのをえたをすことがあります。ただし、ののをむのへのポインタをするのはエラーです。 なぜなら、セグメントされたアーキテクチャでは、ののののアドレスがセグメントをえるがあるため、コンパイラはのに1をえたものがしないことをします。

1Microsoftのは、 `printf()`とのでつけることができます 。

voidポインターによるのるい

`qsort()`ライブラリは、ボイドポインタをってのをなでさせるのいです。

```
void qsort (
    void *base,                                 /* Array to be sorted */
    size_t num,                                 /* Number of elements in array */
    size_t size,                                /* Size in bytes of each element */
    int (*compar)(const void *, const void *)); /* Comparison function for two elements */
```

ソートされるはvoidポインタとしてされるため、のタイプののをできます。の2つのは、 `qsort()`でされるのとのきさバイトをします。

のは、2つのvoidポインタをとるへのポインタです。びしがこのをするようにすることで、`qsort()`はあらゆるのをにソートできます。

をするためののをにします。 `qsort()`されるは、このシグネチャをつがあることにしてください。ポリモ―フィックにするは、voidポインタのを、したいののポインタにキャストすることです。

```
int compare_floats(const void *a, const void *b)
{
    float fa = *((float *)a);
    float fb = *((float *)b);
    if (fa < fb)
        return -1;
    if (fa > fb)
        return 1;
    return 0;
}
```

たちはqsortがこのをってをすることをっているので、するにvoidポインタのをfloatポインタにキャストします。

さて、さ "len"の ""のqsortのいはとてもです

```
qsort(array, len, sizeof(array[0]), compare_floats);
```

オンラインでポインタをむ https://riptutorial.com/ja/c/topic/1108/ポインタ

29: マルチスレッド

き

C11にはのスレッドライブラリ`<threads.h>`ありますが、まだされていないコンパイラはありません。したがって、Cでマルチスレッドをするには、 `pthread.h`ヘッダーをして、POSIXスレッドライブラリpthreadともばれますなどのプラットフォームのをするがあります。

- thrd_t //スレッドをするのなオブジェクト
- int thrd_createthrd_t * thr、thrd_start_t func、void * arg; //スレッドをする
- int thrd_equalthrd_t thr0、thrd_t thr1; //がじスレッドをしているかどうかをべる
- thr_t thrd_currentvoid; //それをびすスレッドのをします
- int thrd_sleepconsttimespec * duration、timespec * remaining; //なくともされた、びしスレッドのをする
- void thrd_yieldvoid; //それをびすスレッドのわりにのスレッドをできるようにする
- _Noreturn void thrd_exitint res; //スレッドをびすスレッドをさせる
- int thrd_detatchthrd_t thr; //のからのスレッドをりす
- int thrd_jointhrd_t thr、int * res; //されたスレッドがするまでのスレッドをブロックする

スレッドをすると、 http: //www.riptutorial.com/c/example/2622/data-raceなどののがされるがあります 。なるスレッドでされるへのレースフリーアクセスのために、C11は`mtx_lock()`ミューテックスまたはオプションの http://www.riptutorial.com/c/topic/4924/atomicsデータおよびするをします。 `stdatomic.h`。

Examples

C11スレッドのな

```
#include <threads.h>
#include <stdio.h>

int run(void *arg)
{
    printf("Hello world of C11 threads.");

    return 0;
}

int main(int argc, const char *argv[])
{
    thrd_t thread;
    int result;

    thrd_create(&thread, run, NULL);

    thrd_join(&thread, &result);
```

```
    printf("Thread return %d at the end\n", result);
}
```

オンラインでマルチスレッドをむ https://riptutorial.com/ja/c/topic/10489/マルチスレッド

30: メモリ

き

にりてられたメモリをするために、Cライブラリは`malloc()` 、 `calloc()` 、 `realloc()` 、 `realloc()` を`free()`ます。 C99では、 `aligned_alloc()`もあります。 システムによっては、 `alloca()`もされています。

- void * aligned_allocsize_t alignment、size_t size; / * C11のみ* /
- void * callocsize_t nelements、size_t size;
- void freevoid * ptr;
- void * mallocsize_t size;
- void * reallocvoid * ptr、size_t size;
- void * allocasize_t size; / *ではなく、ポータブルではない、なalloca.hから。 * /

パラメーター

サイズ `malloc` 、 `realloc` 、および`aligned_alloc`	メモリのサイズバイト。 `aligned_alloc` 、サイズはのでなければなりません。
サイズ `calloc`	のサイズ
ニレメンツ	の
ptr	に`malloc` 、 `calloc` 、 `realloc`または`aligned_alloc`によってされたりてみメモリへのポインタ
アライメント	りてられたメモリの

C11

`aligned_alloc()`は、C11でのみされています。

POSIXにづくシステムなどのシステムは、したメモリ `posix_memalign()` をりてるのをし、そののメモリオプション `mmap()` もえています。

Examples

メモリーの

freeをびしてにりてられたメモリをすることはです。

```
int *p = malloc(10 * sizeof *p); /* allocation of memory */
if (p == NULL)
{
    perror("malloc failed");
    return -1;
}

free(p); /* release of memory */
/* note that after free(p), even using the *value* of the pointer p
   has undefined behavior, until a new value is stored into it. */

/* reusing/re-purposing the pointer itself */
int i = 42;
p = &i; /* This is valid, has defined behaviour */
```

pがすメモリは、 free()びしにlibcのかとなるOSのどちらかによってされるため、されたメモリブロックにpアクセスするとのになります。されたメモリをするポインタは 、、 ダングリングポインタとばれ、セキュリティのリスクがあります。さらに、Cでは、ぶらがっているポインタのにアクセスすることさえものいがあるとべています。のように、ポインタpをすることができます。

あなただけびすことができることにしてください free()からされたポインタにmalloc() calloc() realloc()とaligned_alloc()、またはドキュメントがメモリがそのようりてられているかをします strdup ()ようなものはにするです。ポインタをすると、

- に&をしてられる
- りてられたブロックので、

されています。このようなエラーは、コンパイラによってされることはありませんが、プログラムのはのになります。

このようなされていないをぐには、2つののがあります。

とましいものはです - pがになったときにpがしなくなりました。えば

```
if (something_is_needed())
{

    int *p = malloc(10 * sizeof *p);
    if (p == NULL)
    {
        perror("malloc failed");
        return -1;
    }

    /* do whatever is needed with p */

    free(p);
}
```

ブロックつまり} のわりのでfree()びすことによって、 pはしなくなります。コンパイラは、そのにpをしようとすると、コンパイルエラーがします。

2のは、ポインターがしているメモリーをしたにポインターをにすることです。

```
free(p);
p = NULL;     // you may also use 0 instead of NULL
```

このアプロ―チの

- くのプラットフォ―ムで、ヌルポインタのをみると、クラッシュがします。ここでは、なくともされたにされたをすスタックトレ―スをします。

  ポインタを`NULL`しないと、ポインタがぶらがってしまいます。プログラムがクラッシュするはにいですが、でポインタがすメモリがってれてしまうためです。このようなバグは、のとはになコ―ルスタックにつながるがあるため、トレ―スすることはです。

  したがって、このアプロ―チはフェイル･ファ―ストのコンセプトにいます。

- ヌルポインタをすることはです。 Cのでは 、 `free(NULL)`はであることがされています。

  > freeは、ptrがしすスペ―スのりてをします。つまり、でりてできるようになります。 ptrがヌルポインタの、アクションはしません。それのは、がポインタとしないは、からされた`calloc` 、 `malloc` 、または`realloc`、またはスペ―スがびしによってりてされたは`free`または`realloc`はです。

- には、のアプロ―チをすることはできませんたとえば、メモリはあるにりてられ、まったくなるではでりてされます

メモリのりて

Cのメモリりては、 `<stdlib.h>`ヘッダでされています。オブジェクトのメモリをにりてるは、のコ―ドをできます。

```
int *p = malloc(10 * sizeof *p);
if (p == NULL)
{
    perror("malloc() failed");
    return -1;
}
```

これは、10バイトのを`int` 、その、くのバイトをすることをし、sがメモリにを`malloc`とは、をしすなわち、にしてされたメモリチャンクのアドレス`malloc` ポインタというに`p` 。

`sizeof`のはされているので `sizeof`がに`1`になるように`char` 、 `signed char`および`unsigned char`であるをく、 `sizeof`のをめるのに`sizeof`をすることをお`sizeof`します。

`malloc`はリクエストをできないがあるため、nullポインタをすがあります。でヌルポインタのをするために、これをチェックすることがです。

`malloc()`をしてにりてられたメモリは、 `realloc()`をしてサイズするか、になったときに`free()`をしてすることが`free()`ます。

わりに、 `int array[10];` し`int array[10];`じのメモリがりてられます。しかし、キーワード`static`たないのでされていると、されたとそれがびすでのみできますはスタックにりてられ、スペースはのためにされるためです。がります。あるいは、で`static`されて`static`、またはのでされている、そのはプログラムのです。ポインタもからすことができますが、Cのはをすことはできません。

ゼロメモリ

`malloc`されたメモリはなにされていないがあり、 `memset`してメモリをゼロにするか、なをすぐにコピーするようにするがあります。あるいは、 `calloc`は、すべてのビットが`0`される、ましいサイズのブロックをします。これは、ゼロまたはヌルポインタのとじであるはありません。

```
int *p = calloc(10, sizeof *p);
if (p == NULL)
{
    perror("calloc() failed");
    return -1;
}
```

*`calloc`*にするほとんどのにされるは、パフォーマンスのために`calloc()`をします。したがって、ネットエフェクトはじですが、 `malloc()` 、 `memset()`びすよりもくなります。

メモリの

C11

C11は、されたでスペースをりてるしい`aligned_alloc()`をしました。りてられるメモリが、`malloc()`または`calloc()`でたすことができないでするがあるにできます。 `malloc()`および`calloc()`は、 のオブジェクトにしてにされたメモリをりてますつまり、は`alignof(max_align_t)`。 しかし、`aligned_alloc()`よりきなをできます。

```
/* Allocates 1024 bytes with 256 bytes alignment. */
char *ptr = aligned_alloc(256, 1024);
if (ptr) {
    perror("aligned_alloc()");
    return -1;
}
free(ptr);
```

C11は、2つのをす1されたサイズ 2がアライメント ののでなければならず、2 アライメントのはによってサポートされるなわせであるべきです。いずれかをたさないと、 のがします。

メモリのりて

メモリをりてた、ポインタのをまたはするがあるかもしれません。 `void *realloc(void *ptr, size_t size)`はいオブジェクトは、によってさ`ptr`によりされたサイズをっているオブジェクトへのポインタをす`size`。 `ptr`は、にりてられた`malloc` 、 `calloc`または`realloc`またはヌルポインタでりてられたメモリブロックへのポインタです。のメモリののがされます。しいサイズがきい、い

サイズをえるのメモリはされません。しいサイズがい、のはわれます。 `ptr`がNULLの、しいブロックがりてられ、そのへのポインタがされます。

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *p = malloc(10 * sizeof *p);
    if (NULL == p)
    {
        perror("malloc() failed");
        return EXIT_FAILURE;
    }

    p[0] = 42;
    p[9] = 15;

    /* Reallocate array to a larger size, storing the result into a
     * temporary pointer in case realloc() fails. */
    {
        int *temporary = realloc(p, 1000000 * sizeof *temporary);

        /* realloc() failed, the original allocation was not free'd yet. */
        if (NULL == temporary)
        {
            perror("realloc() failed");
            free(p); /* Clean up. */
            return EXIT_FAILURE;
        }

        p = temporary;
    }

    /* From here on, array can be used with the new size it was
     * realloc'ed to, until it is free'd. */

    /* The values of p[0] to p[9] are preserved, so this will print:
       42 15
    */
    printf("%d %d\n", p[0], p[9]);

    free(p);

    return EXIT_SUCCESS;
}
```

りてされたオブジェクトは、 `*p`とじアドレスをっていてもいなくてもかまいません。したがって、びしがしたは、しいアドレスをむ`realloc`からのりをすることがです。

`realloc`のりをの`p`わりに`temporary`にりてるようにしてください。 `realloc`は、らかのがしたにnullをし、ポインタをきします。これにより、デ―タがわれ、メモリリ―クがします。

サイズの

C99

C99、CにはVLAがあり、にのみされるをつモデルとなります。きすぎるVLAスタックをすがありますをりてないようにするがありますが、*VLA*へのポインタをして`sizeof`でポインタをするとありません。

```
double sumAll(size_t n, size_t m, double A[n][m]) {
    double ret = 0.0;
    for (size_t i = 0; i < n; ++i)
       for (size_t j = 0; j < m; ++j)
          ret += A[i][j]
    return ret;
}

int main(int argc, char *argv[argc+1]) {
   size_t n = argc*10;
   size_t m = argc*8;
   double (*matrix)[m] = malloc(sizeof(double[n][m]));
   // initialize matrix somehow
   double res = sumAll(n, m, matrix);
   printf("result is %g\n", res);
   free(matrix);
}
```

ここで`matrix`は`double[m]`のへのポインタであり、 `double[n][m]`をつ`sizeof`は`n`のためのスペ―スをむことをする。

このスペ―スはすべてしてりてられ、したがって`free`への1のびしでりてをすることができます。

でのVLAのは、ヘッダ―のとポインター―のなにもします。さて、なはパラメ―タの`[]`でされます。どちらのでも、 `[]`のは、パラメ―タリストでにしたパラメ―タをします。 `sumAll`の、これらはユ―ザコ―ドがにしてするさです。 Cのすべてののパラメ―タにして、ものはポインタにきされるので、これはとじです

```
  double sumAll(size_t n, size_t m, double (*A)[m]);
```

つまり、 `n`はにはインタフェ―スのではありませんが、はにち、のアクセスについてするためにチェックコンパイラによってもできます。

に、 `main`、`argc+1`は、 `argv`にしてCがするのさです。

にVLAサポ―トはC11ではオプションですが、C11をしていないコンパイラがないことはわかっています。なら`__STDC_NO_VLA__`マクロでテストすることができます。

reallocptr、0は**freeptr**とではありません。

`realloc`はには、のポインタの`malloc + memcpy + free`とじです。

されたスペ―スのサイズがゼロの、 `realloc`のはです。これは、`0 size`パラメ―タをけるすべてのメモリりてでです。そのようなは、にはnullのポインタをすかもしれませんが、されることはありません。

したがって、 `realloc(ptr,0)`は`free(ptr)`とではありません。それはかもしれない

- 「な」で、ちょうど`ptr`す
- `free(ptr)` 、ダミ―をりてて
- `free(ptr)`して`0`をす
- のは`0`をし、それはもしません。

に、の2つのケ―スはアプリケ―ションコ―ドによってできません。

これは、 `realloc(ptr,0)`がにメモリを/しないがあることをします。したがって、 `free`わりにするべきではありません。

ユ―ザ―のメモリ

`malloc()`は、メモリのペ―ジをするために、なオペレ―ティングシステムのをびします。しかし、にはもなものはなく、きなをしてそこからりけることでストレ―トCですることができますには8バイトにするのがです。

なスキ―ムをするために、びしからされるポインタのにメモリのにブロックがされます。つまり、 `free()`されたポインタからすると、にはブロックサイズにえ、それがフリ―リストにされることをにするいくつかのである、オフみることによってされてもよい-りてのブロックのリンクリストを。

ユ―ザ―がりてをすると、されたとじかそれのサイズのブロックがつかるまできリストがされ、であればされます。これは、ユ―ザ―がなサイズとできないでくのりてをしてしているにメモリのにつながりますすべてののプログラムがそのようにするわけではありません。

```
/* typical control block */
struct block
{
   size_t size;         /* size of block */
   struct block *next;  /* next block in free list */
   struct block *prev;  /* back pointer to previous block in memory */
   void *padding;       /* need 16 bytes to make multiple of 8 */
}

static struct block arena[10000]; /* allocate from here */
static struct block *firstfree;
```

くのプログラムでは、じサイズのさなオブジェクトをりてるがあります。これはがにです。のポインタでブロックをするだけです。したがって、32バイトのブロックがな

```
union block
{
   union block * next;
   unsigned char payload[32];
}

static union block arena[100];
static union block * head;
```

```
void init(void)
{
    int i;
    for (i = 0; i < 100 - 1; i++)
        arena[i].next = &arena[i + 1];
    arena[i].next = 0; /* last one, null */
    head = &block[0];
}

void *block_alloc()
{
    void *answer = head;
    if (answer)
        head = head->next;
    return answer;
}

void block_free(void *ptr)
{
    union block *block = ptr;
    block->next = head;
    head - block;
}
```

このはにかつであり、のさをうことなくすることができます。

allocaスタックにメモリをりてる

allocaは、のためにここでされています。これはにポータブルではなくなではカバーされていません、にながあり、できないほどではありません。のCコードは、 VLAできえるべきです。

マニュアルページ

```
#include <alloca.h>
// glibc version of stdlib.h include alloca.h by default

void foo(int size) {
    char *data = alloca(size);
    /*
      function body;
    */
    // data is automatically freed
}
```

びしのスタックフレームにメモリをりてます。びしがすると、されたポインタによってされるはにされます。

このはメモリにですが、きなりてをするとスタックのオーバーフローがするがあり、 alloca りてられたメモリをfreeでできないことにしてくださいスタックオーバーフローのがするがあります。

これらのから、ループでもでもallocaをすることはされません。

また、のりにメモリがfreeれるため、のとしてポインタをすことはできません はです 。

- `malloc`とじものをびす
- りににフリーズされる
- `free realloc`とがありません の
- ポインタをのとしてすことはできません の
- りりサイズはスタックスペースによってされます。これはほとんどのマシンでは `malloc()`によってなヒープスペースよりもずっとさくなります。
- `alloca()`とVLAを1つのでしないようにする
- `alloca()`は`malloc()`らとじくらいがありません

- しいコードでは`alloca()`をわないでください

C99

の。

```
void foo(int size) {
    char data[size];
    /*
      function body;
    */
    // data is automatically freed
}
```

これは`alloca()`がするところでし、 `alloca()`がえばループので`alloca()`そうでないところでします。 `__STDC_NO_VLA__`していないC99またはC11のいずれかをしています。

オンラインでメモリをむ https://riptutorial.com/ja/c/topic/4726/メモリ

31: リンクされたリスト

Cはリンクリストデータをしていません。 Cをしていてリンクリストがなは、のライブラリGLibなどのリンクリストをするか、のリンクリストインターフェイスをするがあります。このトピックでは、リンクリストとリンクリストをするためのとしてできるリンクリストとリンクリストのをします。

リンクリスト

リストには、nextという1つのリンクでされるノードがまれています。

データ

```
struct singly_node
{
  struct singly_node * next;
};
```

リンクリスト

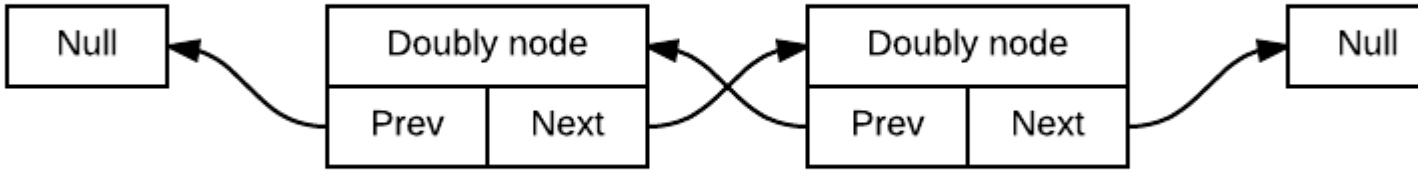
このリストには、との2つのリンクでされるノードがまれています。リンクは、、じをつノードをしています。

データ

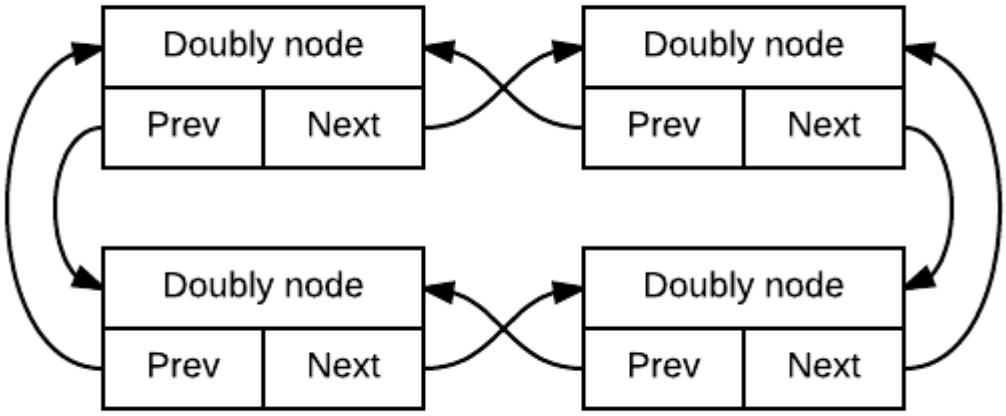
```
struct doubly_node
{
  struct doubly_node * prev;
  struct doubly_node * next;
};
```

Topoliges

リニアまたはオープン



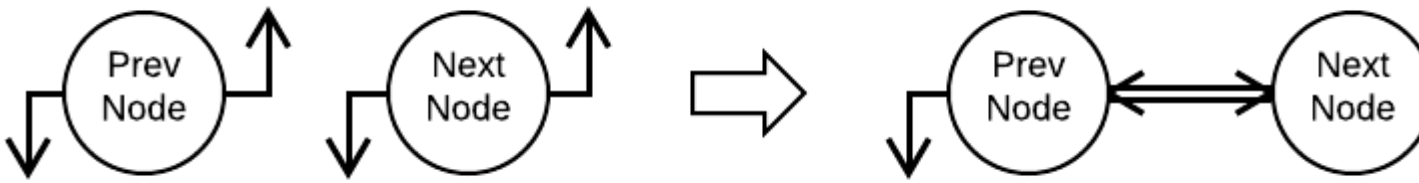
またはリング



き

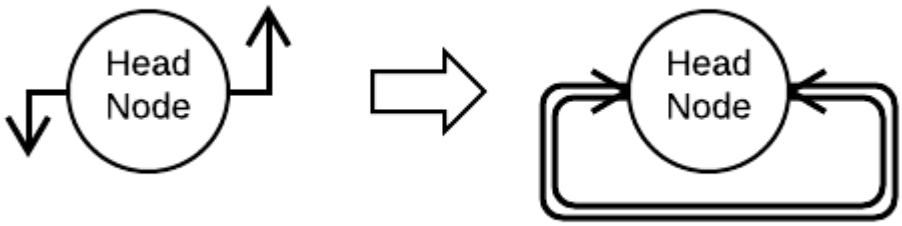
バインド

2つのノードをする。



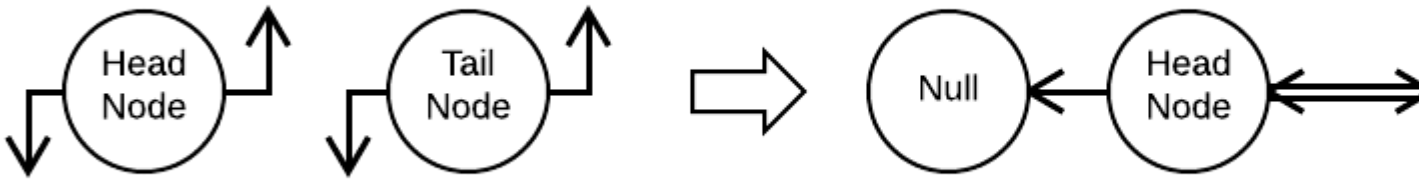

```
void doubly_node_bind (struct doubly_node * prev, struct doubly_node * next)
{
  prev->next = next;
  next->prev = prev;
}
```

リンクリストをる




```
void doubly_node_make_empty_circularly_list (struct doubly_node * head)
{
  doubly_node_bind (head, head);
}
```

リンクリストをる




```
void doubly_node_make_empty_linear_list (struct doubly_node * head, struct doubly_node * tail)
{
  head->prev = NULL;
  tail->next = NULL;
  doubly_node_bind (head, tail);
}
```

リストにはにNULLのわりに1つのノードがまれるとします。その、プロシージャはNULLをするはありません。

```
void doubly_node_insert_between
(struct doubly_node * prev, struct doubly_node * next, struct doubly_node * insertion)
{
  doubly_node_bind (prev, insertion);
  doubly_node_bind (insertion, next);
}

void doubly_node_insert_before
(struct doubly_node * tail, struct doubly_node * insertion)
{
  doubly_node_insert_between (tail->prev, tail, insertion);
}

void doubly_node_insert_after
(struct doubly_node * head, struct doubly_node * insertion)
{
  doubly_node_insert_between (head, head->next, insertion);
```

```
}
```

Examples

リンクリストのにノードをする

のコードはのをめ、リンクリストのにそれらをしけます。

```
/* This program will demonstrate inserting a node at the beginning of a linked list */

#include <stdio.h>
#include <stdlib.h>

struct Node {
  int data;
  struct Node* next;
};


void insert_node (struct Node **head, int nodeValue);
void print_list (struct Node *head);

int main(int argc, char *argv[]) {
  struct Node* headNode;
  headNode = NULL; /* Initialize our first node pointer to be NULL. */
  size_t listSize, i;
  do {
    printf("How many numbers would you like to input?\n");
  } while(1 != scanf("%zu", &listSize));

  for (i = 0; i < listSize; i++) {
    int numToAdd;
    do {
      printf("Enter a number:\n");
    } while (1 != scanf("%d", &numToAdd));

    insert_node (&headNode, numToAdd);
    printf("Current list after your inserted node: \n");
    print_list(headNode);
  }

  return 0;
}

void print_list (struct Node *head) {
  struct node* currentNode = head;

  /* Iterate through each link. */
  while (currentNode != NULL) {
      printf("Value: %d\n", currentNode->data);
      currentNode = currentNode -> next;
  }
}

void insert_node (struct Node **head, int nodeValue) {
  struct Node *currentNode = malloc(sizeof *currentNode);
  currentNode->data = nodeValue;
```

```
  currentNode->next = (*head);

  *head = currentNode;
}
```

ノ―ドのの

にノ―ドをするをするために、なシナリオをてみましょう

1. リストはですので、しいノ―ドをするがあります。この、 `HEAD`はのノ―ドへのポインタで、のようになります。

```
| HEAD | --> NULL
```

line `currentNode->next = *headNode;`がりてられます`currentNode->next`であることが`NULL`ため、`headNode`々のからまり`NULL`。

、のノ―ドのポインタをのノ―ドをすようにします。

```
 -----      -------------
|HEAD | --> |CURRENTNODE| --> NULL /* The head node points to the current node */
 -----      -------------
```

これは`*headNode = currentNode;`われ`*headNode = currentNode;`

2. リストにはすでにがされています。にしいノ―ドをするがあります。にするために、まず1つのノ―ドからめましょう

```
-----    -----------
HEAD --> FIRST NODE --> NULL
-----    -----------
```

`currentNode->next = *headNode`、データはのようになります。

```
---------        -----    ---------------------
currentNode -->  HEAD --> POINTER TO FIRST NODE --> NULL
---------        -----    ---------------------
```

これは、、らかにするがあります`*headNode`をしているがあり`currentNode`。

```
----    -----------    ---------------
HEAD -> currentNode -->     NODE       -> NULL
----    -----------    ---------------
```

これは`*headNode = currentNode;`われ`*headNode = currentNode;`

nのノ―ドにノ―ドをする

これまでは、リンクリストのにノードをするをしました 。しかし、ほとんどの、のにもノードをできるようにしたいとうでしょう。のコードは、リンクされたリストのどこにでもノードをするための`insert()`のをしています。

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
  int data;
  struct Node* next;
};

struct Node* insert(struct Node* head, int value, size_t position);
void print_list (struct Node* head);

int main(int argc, char *argv[]) {
  struct Node *head = NULL; /* Initialize the list to be empty */

  /* Insert nodes at positions with values: */
  head = insert(head, 1, 0);
  head = insert(head, 100, 1);
  head = insert(head, 21, 2);
  head = insert(head, 2, 3);
  head = insert(head, 5, 4);
  head = insert(head, 42, 2);

  print_list(head);
  return 0;
}

struct Node* insert(struct Node* head, int value, size_t position) {
  size_t i = 0;
  struct Node *currentNode;

  /* Create our node */
  currentNode = malloc(sizeof *currentNode);
  /* Check for success of malloc() here! */

  /* Assign data */
  currentNode->data = value;

  /* Holds a pointer to the 'next' field that we have to link to the new node.
     By initializing it to &head we handle the case of insertion at the beginning. */
  struct Node **nextForPosition = &head;
  /* Iterate to get the 'next' field we are looking for.
     Note: Insert at the end if position is larger than current number of elements. */
  for (i = 0; i < position && *nextForPosition != NULL; i++) {
      /* nextForPosition is pointing to the 'next' field of the node.
         So *nextForPosition is a pointer to the next node.
         Update it with a pointer to the 'next' field of the next node. */
      nextForPosition = &(*nextForPosition)->next;
  }

  /* Here, we are taking the link to the next node (the one our newly inserted node should
  point to) by dereferencing nextForPosition, which points to the 'next' field of the node
  that is in the position we want to insert our node at.
  We assign this link to our next value. */
  currentNode->next = *nextForPosition;

  /* Now, we want to correct the link of the node before the position of our
```

```
  new node: it will be changed to be a pointer to our new node. */
  *nextForPosition = currentNode;

  return head;
}

void print_list (struct Node* head) {
  /* Go through the list of nodes and print out the data in each node */
  struct Node* i = head;
  while (i != NULL) {
    printf("%d\n", i->data);
    i = i->next;
  }
}
```

リンクされたリストをする

このタスクをにすることもできますが、このではアプローチをすることをしました。このタスクは、リンクされたリストのにすべてのノードをするにです。にをします。

```
#include <stdio.h>
#include <stdlib.h>

#define NUM_ITEMS 10

struct Node {
  int data;
  struct Node *next;
};

void insert_node(struct Node **headNode, int nodeValue, int position);
void print_list(struct Node *headNode);
void reverse_list(struct Node **headNode);


int main(void) {
  int i;
  struct Node *head = NULL;

  for(i = 1; i <= NUM_ITEMS; i++) {
    insert_node(&head, i, i);
  }
  print_list(head);

  printf("I will now reverse the linked list\n");
  reverse_list(&head);
  print_list(head);
  return 0;
}

void print_list(struct Node *headNode) {
  struct Node *iterator;

  for(iterator = headNode; iterator != NULL; iterator = iterator->next) {
    printf("Value: %d\n", iterator->data);
  }
}
```

```
void insert_node(struct Node **headNode, int nodeValue, int position) {
  int i;
  struct Node *currentNode = (struct Node *)malloc(sizeof(struct Node));
  struct Node *nodeBeforePosition = *headNode;

  currentNode->data = nodeValue;

  if(position == 1) {
    currentNode->next = *headNode;
    *headNode = currentNode;
    return;
  }

  for (i = 0; i < position - 2; i++) {
    nodeBeforePosition = nodeBeforePosition->next;
  }

  currentNode->next = nodeBeforePosition->next;
  nodeBeforePosition->next = currentNode;
}

void reverse_list(struct Node **headNode) {
  struct Node *iterator = *headNode;
  struct Node *previousNode = NULL;
  struct Node *nextNode = NULL;

  while (iterator != NULL) {
    nextNode = iterator->next;
    iterator->next = previousNode;
    previousNode = iterator;
    iterator = nextNode;
  }

  /* Iterator will be NULL by the end, so the last node will be stored in
  previousNode.  We will set the last node to be the headNode */
  *headNode = previousNode;
}
```

リバ―スリストメソッドの

たちは、`previousNode`りに`NULL`々は、ループののにっているので、々はのヘッドノ―ドのにノ―ドをしているなら、それはなります`NULL`。のノ―ドはリストののノ―ドになり、のは`NULL`になり`NULL`。

に、ここでリンクされたリストをさせるというは、リンクをにさせることです。ノ―ドののメンバ―は、そののノ―ドになります。

```
Head -> 1 -> 2 -> 3 -> 4 -> 5
```

はノ―ドをします。このリストはのようになります。

```
1 <- 2 <- 3 <- 4 <- 5 <- Head
```

に、はわりに5のノ―ドをし、ノ―ドはそののノ―ドをしていなければなりません。

そのにもなかったので、ノード1は`NULL`をすべきです。ノード2はノード1をし、ノード3はノード2をすはずである。

しかし、このにはさなが*1*つあります。のノードへのリンクをしてそれをのノードにすると、リンクのノードがリストののノードにすることはできなくなります。

このをするには、リンクをするに `nextNode` にのをするだけです。

リンクリスト

リンクリストにノードをする、リストをににす、にするをすコードの

```
#include <stdio.h>
#include <stdlib.h>

/* This data is not always stored in a structure, but it is sometimes for ease of use */
struct Node {
  /* Sometimes a key is also stored and used in the functions */
  int data;
  struct Node* next;
  struct Node* previous;
};

void insert_at_beginning(struct Node **pheadNode, int value);
void insert_at_end(struct Node **pheadNode, int value);

void print_list(struct Node *headNode);
void print_list_backwards(struct Node *headNode);

void free_list(struct Node *headNode);

int main(void) {
  /* Sometimes in a doubly linked list the last node is also stored */
  struct Node *head = NULL;

  printf("Insert a node at the beginning of the list.\n");
  insert_at_beginning(&head, 5);
  print_list(head);

  printf("Insert a node at the beginning, and then print the list backwards\n");
  insert_at_beginning(&head, 10);
  print_list_backwards(head);

  printf("Insert a node at the end, and then print the list forwards.\n");

  insert_at_end(&head, 15);
  print_list(head);

  free_list(head);

  return 0;
}

void print_list_backwards(struct Node *headNode) {
  if (NULL == headNode)
  {
    return;
```

```
  }
  /*
  Iterate through the list, and once we get to the end, iterate backwards to print
  out the items in reverse order (this is done with the pointer to the previous node).
  This can be done even more easily if a pointer to the last node is stored.
  */
  struct Node *i = headNode;
  while (i->next != NULL) {
    i = i->next; /* Move to the end of the list */
  }

  while (i != NULL) {
    printf("Value: %d\n", i->data);
    i = i->previous;
  }
}

void print_list(struct Node *headNode) {
  /* Iterate through the list and print out the data member of each node */
  struct Node *i;
  for (i = headNode; i != NULL; i = i->next) {
    printf("Value: %d\n", i->data);
  }
}

void insert_at_beginning(struct Node **pheadNode, int value) {
  struct Node *currentNode;

  if (NULL == pheadNode)
  {
    return;
  }
  /*
  This is done similarly to how we insert a node at the beginning of a singly linked
  list, instead we set the previous member of the structure as well
  */
  currentNode = malloc(sizeof *currentNode);

  currentNode->next = NULL;
  currentNode->previous = NULL;
  currentNode->data = value;

  if (*pheadNode == NULL) { /* The list is empty */
    *pheadNode = currentNode;
    return;
  }

  currentNode->next = *pheadNode;
  (*pheadNode)->previous = currentNode;
  *pheadNode = currentNode;
}

void insert_at_end(struct Node **pheadNode, int value) {
  struct Node *currentNode;

  if (NULL == pheadNode)
  {
    return;
  }

  /*
```

```
  This can, again be done easily by being able to have the previous element.  It
  would also be even more useful to have a pointer to the last node, which is commonly
  used.
  */

  currentNode = malloc(sizeof *currentNode);
  struct Node *i = *pheadNode;

  currentNode->data = value;
  currentNode->next = NULL;
  currentNode->previous = NULL;

  if (*pheadNode == NULL) {
    *pheadNode = currentNode;
    return;
  }

  while (i->next != NULL) { /* Go to the end of the list */
    i = i->next;
  }

  i->next = currentNode;
  currentNode->previous = i;
}

void free_list(struct Node *node) {
  while (node != NULL) {
    struct Node *next = node->next;
    free(node);
    node = next;
  }
}
```

のノードへのポインタをするとながあることにしてくださいリストのまでにジャンプすることは、まですることもです。

```
struct Node *lastNode = NULL;
```

この、リストのにするがあります。

キーをしてをすることもあります。これはにNodeのメンバです。

```
struct Node {
  int data;
  int key;
  struct Node* next;
  struct Node* previous;
};
```

このキーは、のなど、のにしてタスクがされたときにされます。

オンラインでリンクされたリストをむ https://riptutorial.com/ja/c/topic/560/リンクされたリスト

32: なCプログラミングのイディオムとの

Examples

リテラルとの

あるとあるとをしているとします

```
if ( i  == 2) //Bad-way
{
    doSomething;
}
```

== =とっているとします。それから、それをするあなたのいがかかります。

```
if( 2 == i) //Good-way
{
    doSomething;
}
```

に、がってっていると、コンパイラは「リテラルへのりてをみました」とをいます。これは2つのをするときはされませんが、しずつちます。

はこちらをごください。

のパラメータリストをにしないでください - voidをしてください

びされたときにをとせず、プロトタイプとでパラメータリストをするのジレンマにしているをしているとします。

- プロトタイプとのにしてパラメータリストをにするがあります。これにより、なびしステートメントのようにえます。
- あなたは、キーワード***void***ののうちの1つそのうちのほんのしかありませんのどこかをんで、びしにをけれないのパラメータリストをすることです。したがって、これもです。

だから、しいですか

ANSWERキーワード***void***をする

ながののためにするのをしているは、コードでそのをするがよいでしょう。たとえば、 `#define`マクロのわりに`enum`をしますこれはのです。

C11セクション6.7.6.3「」10には、

> リストののとしてののののないパラメーターのなケースは、にパラメーターがないこと

をします。

そのじセクションのパラグラフ14はのいをしています

> ...そのののであるののリストは、そのにパラメ―タがないことをします。そののので はないののリストは、パラメ―タ―のまたはタイプにするがされないことをします。

のものについてKRpgs- 72-73によってされたな

> さらに、にがまれていない `double atof();`それはまた、 `atof`のについてはもされないことをすると`atof`。 すべてのパラメ―タチェックがオフになります。こののリストのなは、いCプログラムをしいコンパイラでコンパイルできるようにすることです。しかし、しいプログラムでそれをうのはいえです。がをるは、それらをします。がないは`void`し`void`。

これはあなたのプロトタイプがどのようにえるかです

```
int foo(void);
```

そして、これはのです

```
int foo(void)
{
    ...
    <statements>
    ...
    return 1;
}
```

の`int foo()`のつまり、キ―ワ―ド***void***をわずに`int foo()`をうの1つは、 `foo(42)`ようなったをってをびすとコンパイラがエラ―をできることです。こののびしでは、パラメ―タリストをのままにしてもエラ―はしません。エラ―はかにし、されず、コ―ドはされます。

これはまた、 `main()`をのようにするがあることをします

```
int main(void)
{
    ...
    <statements>
    ...
    return 0;
}
```

のパラメ―タリストでされたがをらなくても、のプロトタイプはされないので、ききをしてをびすと、コンパイラはをわないことにしてください。えば

```
#include <stdio.h>

static void parameterless()
{
```

```
    printf("%s called\n", __func__);
}

int main(void)
{
    parameterless(3, "arguments", "provided");
    return 0;
}
```

そのコードが`proto79.c`ファイルにされていれば、GCCをってUnixでコンパイルすることができますデモンストレーションにされるmacOS Sierra 10.12.5のバージョン7.1.0

```
$ gcc -O3 -g -std=c11 -Wall -Wextra -Werror -Wmissing-prototypes -pedantic proto79.c -o
proto79
$
```

よりしいオプションをしてコンパイルすると、エラーがします。

```
$ gcc -O3 -g -std=c11 -Wall -Wextra -Werror -Wmissing-prototypes -Wstrict-prototypes -Wold-
style-definition -pedantic proto79.c -o proto79
proto79.c:3:13: error: function declaration isn't a prototype [-Werror=strict-prototypes]
 static void parameterless()
             ^~~~~~~~~~~~
proto79.c: In function 'parameterless':
proto79.c:3:13: error: old-style function definition [-Werror=old-style-definition]
cc1: all warnings being treated as errors
$
```

になプロトタイプ`static void parameterless(void)`すると、コンパイルでエラーがします。

```
$ gcc -O3 -g -std=c11 -Wall -Wextra -Werror -Wmissing-prototypes -Wstrict-prototypes -Wold-
style-definition -pedantic proto79.c -o proto79
proto79.c: In function 'main':
proto79.c:10:5: error: too many arguments to function 'parameterless'
     parameterless(3, "arguments", "provided");
     ^~~~~~~~~~~~
proto79.c:3:13: note: declared here
 static void parameterless(void)
             ^~~~~~~~~~~~
$
```

- にプロトタイプがあることをし、ルールにわないときにコンパイラがあなたにするようにしてください。

オンラインでなCプログラミングのイディオムとのをむ https://riptutorial.com/ja/c/topic/10543/なcプログラミングのイディオムとの

33: な

- _Generic、generic-assoc-list

パラメーター

パラメータ	
ジェネリックアソーシエーションリスト	ジェネリックアソシエーションまたはジェネリックアソーシエーションリスト、ジェネリックアソシエーション
ジェネリック	タイプ**OR**デフォルト

1. すべてのは、 `_Generic`のにされます。
2. `_Generic`なは7でされる。したがって、のようなフェーズはにしています。

Examples

がのかどうかをチェックする

```
#include <stdio.h>

#define is_const_int(x) _Generic((&x),  \
        const int *: "a const int",     \
        int *:       "a non-const int", \
        default:     "of other type")

int main(void)
{
    const int i = 1;
    int j = 1;
    double k = 1.0;
    printf("i is %s\n", is_const_int(i));
    printf("j is %s\n", is_const_int(j));
    printf("k is %s\n", is_const_int(k));
}
```

```
i is a const int
j is a non-const int
k is of other type
```

しかし、タイプジェネリックマクロがのようにされているとします。

```
#define is_const_int(x) _Generic((x), \
        const int: "a const int",     \
        int:       "a non-const int", \
        default:   "of other type")
```

はのとおりです。

```
i is a non-const int
j is a non-const int
k is of other type
```

これは、 `_Generic`プライマリののののために、すべてのがされるためです。

タイプジェネリックマクロ

```
#include <stdio.h>

void print_int(int x) { printf("int: %d\n", x); }
void print_dbl(double x) { printf("double: %g\n", x); }
void print_default() { puts("unknown argument"); }

#define print(X) _Generic((X), \
        int: print_int, \
        double: print_dbl, \
        default: print_default)(X)

int main(void) {
    print(42);
    print(3.14);
    print("hello, world");
}
```

```
int: 42
double: 3.14
unknown argument
```

が`int`でも`double`でもないは、がされることにしてください。をするには、そのタイプを`print(X)`マクロにします。

のにづく

ジェネリックののののがで、のがすべてである、ネストされた`_Generic`をけるなは、のパラメータのをすることです。

```
int max_int(int, int);
unsigned max_unsigned(unsigned, unsigned);
double max_double(double, double);

#define MAX(X, Y) _Generic((X)+(Y),                \
                           int:      max_int,      \
                           unsigned: max_unsigned, \
                           default:  max_double)   \
                     ((X), (Y))
```

ここで、`(X)+(Y)`は、そのタイプによってのみされ、されない。されたタイプをするために、オペランドののがされます。

よりなでは、オペレータにするのについて、それらをにれにすることによってをうことができる。

このでは、2つのintおよび/またはstringのみわせをり、それらのをす4つのをします。

```
int AddIntInt(int a, int b);
int AddIntStr(int a, const char* b);
int AddStrInt(const char*  a, int b );
int AddStrStr(const char*  a, const char*  b);

#define AddStr(y)                               \
   _Generic((y),          int: AddStrInt,      \
                         char*: AddStrStr,     \
                  const char*: AddStrStr )

#define AddInt(y)                               \
   _Generic((y),           int: AddIntInt,     \
                         char*: AddIntStr,     \
                  const char*: AddIntStr )

#define Add(x, y)                               \
   _Generic((x) ,         int: AddInt(y) ,     \
                        char*: AddStr(y) ,     \
                 const char*: AddStr(y))       \
                          ((x), (y))

int main( void )
{
    int result = 0;
    result = Add( 100 , 999 );
    result = Add( 100 , "999" );
    result = Add( "100" , 999 );
    result = Add( "100" , "999" );

    const int a = -123;
    char b[] = "4321";
    result = Add( a , b );

    int c = 1;
    const char d[] = "0";
    result = Add( d , ++c );
}
```

yが2されたようにえますが、それは1ではありません。のは、のびしのとに、Add (x , y)マクロので1だけされます。

---

1 ISOIEC 9899201X 6.5.1.13
のはされません。

オンラインでなをむ https://riptutorial.com/ja/c/topic/571/な

34:

`rand()`のために、のくのデフォルトのがにわたりしてきました。それらのには

- `arc4random()` OS XおよびBSDで
- `random()` Linuxで
- `drand48()` POSIXで

Examples

`rand()`は、 `0`と`RAND_MAX` `0`と`RAND_MAX`むののランダムをするためにできます。

`srand(int)`は、ジェネレータのシードにされます。 `rand()`はじシードでシードされるたびに、じのシーケンスをするがあります。 `rand()`びすにだけしてください。それは、のしいバッチをするたびに、りししたり、すべきではありません。

なは、 `time(NULL)`をシードとしてすることです。ジェネレータがなシーケンスをとするは、プログラムのにじでジェネレータをシードすることができます。これはにリリースコードではありませんが、バグをのあるものにするためにはデバッグにです。

ジェネレータをシードしないは、に`srand(1)`シードされているかのようにるうようにしてください。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void) {
    int i;
    srand(time(NULL));
    i = rand();

    printf("Random value between [0, %d]: %d\n", RAND_MAX, i);
    return 0;
}
```

な

```
Random value between [0, 2147483647]: 823321433
```

ノート

Cは、されるランダムシーケンスのをしません。には、 `rand()`いくつかのでは、されたのとにながありました。 **`rand()`**のは、のようななのにはおめできません。

ジェネレータ

ここには、 `rand()`またはのライブラリにしないスタンドアロンジェネレーータがあります。

なぜそんなことがしいのおそらく、あなたのプラットフォームのみみジェネレーータをしていないかもしれません。あるいは、のライブラリとはしたランダムのなソースがなこともあります。

このコードはpcg-random.orgのPCG32であり、れたをつでなRNGです。それはではではないので、にしないでください。

```
#include <stdint.h>

/* *Really* minimal PCG32 code / (c) 2014 M.E. O'Neill / pcg-random.org
 * Licensed under Apache License 2.0 (NO WARRANTY, etc. see website) */

typedef struct { uint64_t state;  uint64_t inc; } pcg32_random_t;

uint32_t pcg32_random_r(pcg32_random_t* rng) {
    uint64_t oldstate = rng->state;
    /* Advance internal state */
    rng->state = oldstate * 6364136223846793005ULL + (rng->inc | 1);
    /* Calculate output function (XSH RR), uses old state for max ILP */
    uint32_t xorshifted = ((oldstate >> 18u) ^ oldstate) >> 27u;
    uint32_t rot = oldstate >> 59u;
    return (xorshifted >> rot) | (xorshifted << ((-rot) & 31));
}

void pcg32_srandom_r(pcg32_random_t* rng, uint64_t initstate, uint64_t initseq) {
    rng->state = 0U;
    rng->inc = (initseq << 1u) | 1u;
    pcg32_random_r(rng);
    rng->state += initstate;
    pcg32_random_r(rng);
}
```

それをびすはのとおりです。

```
#include <stdio.h>
int main(void) {
    pcg32_random_t rng; /* RNG state */
    int i;

    /* Seed the RNG */
    pcg32_srandom_r(&rng, 42u, 54u);

    /* Print some random 32-bit integers */
    for (i = 0; i < 6; i++)
        printf("0x%08x\n", pcg32_random_r(&rng));

    return 0;
}
```

されたにをする

、をする、の、または0.01.0のpをするとです。モジュラスをしてシードをいにらすことができますが、NがRAND_MAXにしてきくなると、さなサイクルがにし、その、がわずかにんでしまいます。

マクロ

```
#define uniform() (rand() / (RAND_MAX + 1.0))
```

0.01.0のεをするので、ε

```
i = (int)(uniform() * N)
```

0からN - 1ののに`i`をします。

ながら、RAND_MAXは`double`のがにすことができるというでながあります。これは、 `RAND_MAX + 1.0`がRAND_MAXにされ、が々1をすことをします。しかしこれはそうではありません。

Xorshift Generation

にいとに`rand()`きは、xorshift、によってされたのクラスであるジョージ·マルサグリア 。 xorshift ジェネレータは、されていないのジェネレータのでももです。よりくのとのは、 xorshift Wikipediaのページでできます。

```
#include <stdint.h>

/* These state variables must be initialised so that they are not all zero. */
uint32_t w, x, y, z;

uint32_t xorshift128(void)
{
    uint32_t t = x;
    t ^= t << 11U;
    t ^= t >> 8U;
    x = y; y = z; z = w;
    w ^= w >> 19U;
    w ^= t;
    return w;
}
```

オンラインでをむ https://riptutorial.com/ja/c/topic/365/

35:

- void* signalint sig、void* funcintint;

パラメーター

パラメータ	
sig	シグナルハンドラを`SIGABRT`、`SIGFPE`、`SIGILL`、`SIGTERM`、`SIGINT`、`SIGSEGV`またはらかののいずれかにする`SIGFPE`。
	のいずれかであるシグナルハンドラ`SIG_DFL`、デフォルトのハンドラ`SIG_IGN`シグナルをする、またはきポインタ`void foo(int sig);`。

Cからののみをうシグナルハンドラのは、ユーザのシグナルハンドラでができるか、できないかを々なをす。

- `SIGSEGV`、`SIGFPE`、`SIGILL`またはのインプリメンテーションのハードウェアりみをしているときにユーザーがると、そのはCではされません。これは、Cのインタフェースがえば、`0`したをするをえないため、プログラムをすときにハードウェアりみがするとまったくじエラーになるためです。
- `abort`、または`raise`びしのとしてユーザーがびされた、シグナル・ハンドラーは`raise`をびす`raise`はできません。
- はどんなのでもするがあるため、のはにされず、もでうまくしません。したがって、シグナルハンドラのデータにするすべてのは、
  - タイプ`sig_atomic_t`すべてのバージョンまたはロックフリーのアトミックタイプC11
  - `volatile`されたものです。
- Cライブラリののは、プログラムのグローバルなでをするがあるため、はこれらのをりません。 Cは`abort`、`_Exit` C99、`quick_exit` C11、`signal`じシグナル、およびいくつかのアトミックC11のみをします。

ののいずれかにした、Cによってがになります。プラットフォームにはのがまれているがありますが、にそのプラットフォームをえてすることはできません。

- 、システムは、シグナルハンドラからできるCライブラリであるシグナルセーフであるのリストをっています。、`printf`はこれらのの1つです。
- に、Cのでは、スレッドインタフェースC11やPOSIXスレッドなどのプラットフォームのスレッドライブラリとのやりとりについてはあまりしていません。そのようなプラットホーム

は、それでシグナルとそのようなスレッドライブラリのをしなければならない。

Examples

"signal"による

シグナルは、プログラムがしたり、 `SIGINT` - インタラクティブなのように、プログラムのからされたとき `Cntrl-C`キー—などにトリガされたときに、 `SIGSEGV`ようなすることができます。

`signal()`はISO Cのであり、のシグナルをするをりてるためにできます

```
#include <stdio.h>  /* printf() */
#include <stdlib.h> /* abort()  */
#include <signal.h> /* signal() */

void handler_nonportable(int sig)
{
    /* undefined behavior, maybe fine on specific platform */
    printf("Catched: %d\n", sig);

    /* abort is safe to call */
    abort();
}

sig_atomic_t volatile finished = 0;

void handler(int sig)
{
    switch (sig) {
    /* hardware interrupts should not return */
    case SIGSEGV:
    case SIGFPE:
    case SIGILL:
```

C11

```
      /* quick_exit is safe to call */
      quick_exit(EXIT_FAILURE);
```

C11

```
      /* use _Exit in pre-C11 */
      _Exit(EXIT_FAILURE);
```

```
    default:
       /* Reset the signal to the default handler,
          so we will not be called again if things go
          wrong on return. */
      signal(sig, SIG_DFL);
      /* let everybody know that we are finished */
      finished = sig;
      return;
   }
}
```

```
int main(void)
{

    /* Catch the SIGSEGV signal, raised on segmentation faults (i.e NULL ptr access */
    if (signal(SIGSEGV, &handler) == SIG_ERR) {
        perror("could not establish handler for SIGSEGV");
        return EXIT_FAILURE;
    }

    /* Catch the SIGTERM signal, termination request */
    if (signal(SIGTERM, &handler) == SIG_ERR) {
        perror("could not establish handler for SIGTERM");
        return EXIT_FAILURE;
    }

    /* Ignore the SIGINT signal, by setting the handler to `SIG_IGN`. */
    signal(SIGINT, SIG_IGN);


    /* Do something that takes some time here, and leaves
       the time to terminate the program from the keyboard. */

    /* Then: */

    if (finished) {
       fprintf(stderr, "we have been terminated by signal %d\n", (int)finished);
        return EXIT_FAILURE;
    }


    /* Try to force a segmentation fault, and raise a SIGSEGV */
    {
      char* ptr = 0;
      *ptr = 0;
    }

    /* This should never be executed */
    return EXIT_SUCCESS;
}
```

`signal()`を`signal()`、シグナルハンドラのでができるかがなをけます。しくは、をしてください。

POSIXはのをおめ`sigaction()`わりの`signal()`そののとしいに。 POSIXはまた、 `SIGUSR1`や`SIGUSR2`など、ISO Cよりもくのシグナルをしています 。これは、プログラマがにのでできます。

オンラインでをむ https://riptutorial.com/ja/c/topic/453/

36: のとし

き

このセクションでは、Cプログラマがすべきであり、をけるべきであるないのいくつかについてします。のとそののについては、 のをしてください。

Examples

でのきとなしの

、ミックスするのはいアイデアではありませんsignedとunsignedでを。えば、ののは

```
#include <stdio.h>

int main(void)
{
    unsigned int a = 1000;
    signed int b = -1;

    if (a > b) puts("a is more than b");
    else puts("a is less or equal than b");

    return 0;
}
```

1000が-1よりきいので、a is more than bとされますが、そうではありません。

なるのは、のとばれるのでされます6.3.1.8を。

この、「」はunsigned int。 のでしたように 、

> そうでない、なしをするオペランドがのオペランドののランクのランクをする、きのオペランドは、なしのオペランドのにされる。

つまり、 intオペランドbはのにunsigned intにされます。

-1にするとunsigned int、のであるunsigned intことをし、1000よりもきい、 a > b falseですが。

すると==のわりにってく=

=はにされます。

==はにされます。

1つは、2つをぜないようにするがあります。には1つはってきみます

```
/* assign y to x */
if (x = y) {
     /* logic */
}
```

にしかったのはのときです

```
/* compare if x is equal to y */
if (x == y) {
    /* logic */
}
```

はyのをxにし、をうわりにそのがゼロであるかどうかをチェックします。

```
if ((x = y) != 0) {
    /* logic */
}
```

のをテストすることがされ、にされることがあります。なぜなら、コードをするがなく、めてにわなければならないからです。

```
while ((c = getopt_long(argc, argv, short_options, long_options, &option_index)) != -1) {
        switch (c) {
        ...
        }
}
```

```
c = getopt_long(argc, argv, short_options, long_options, &option_index);
while (c != -1) {
        switch (c) {
        ...
        }
        c = getopt_long(argc, argv, short_options, long_options, &option_index);
}
```

のコンパイラはこのパターンをし、のようにのにがあるとしませんが、のについてすることがあります。えば

```
if (x = y)          /* warning */

if ((x = y))        /* no warning */
if ((x = y) != 0)   /* no warning; explicit */
```

いくつかのプログラマーは、をオペレーターのにくというをしますにYodaとばれます 。はrvaluesなので、ったがされた、コンパイラはエラーをスローします。

```
if (5 = y) /* Error */

if (5 == y) /* No error */
```

しかし、これはコーードのをしくさせ、プログラマーがれたCコーードにうならばではないとえられ、2つのをするにはにたないのでなではありません。さらに、のコンパイラのくは、Yodaのでコーードがされているときにをすことがあります。

セミコロンのい

セミコロンにしてください。の

```
if (x > a);
   a = x;
```

には

```
if (x > a) {}
a = x;
```

これはどんなでもxがaにされることをしています。

セミコロンをすると、たないがじることがあります。

```
if (i < 0)
    return
day = date[0];
hour = date[1];
minute = date[2];
```

りのセミコロンはしているので、day = date [0]がされます。

これとのをする1つのは、のとルーープでにをすることです。えば

```
if (x > a) {
    a = x;
}
```

\ 0にな1バイトをりてることをれる

をmallocされたバッファにコピーするときは、ずstrlenに1をえてください。

```
char *dest = malloc(strlen(src)); /* WRONG */
char *dest = malloc(strlen(src) + 1); /* RIGHT */

strcpy(dest, src);
```

これは、 strlenはに\0がまれていないためです。 WRONG のようにのアプローチをとってstrcpyをびすと、あなたのプログラムはのをびします。

また、 stdinやのソースからののをみんでいるにもされます。えば

```
#define MAX_INPUT_LEN 42

char buffer[MAX_INPUT_LEN]; /* WRONG */
char buffer[MAX_INPUT_LEN + 1]; /* RIGHT */

scanf("%42s", buffer);  /* Ensure that the buffer is not overflowed */
```

きメモリをれるメモリリ―ク

プログラミングのベストプラクティスは、のコードでりてられたメモリをするか、 `strdup()` などのライブラリAPIなどのまたはをにびすことです。メモリをできないとメモリリ―クがするがあります。このリ―クは、プログラムまたはシステムでできないなのメモリをし、クラッシュやのにつながるがあります。ルプまたはでりしリ―クがすると、がするがくなります。れたプログラムがされるほど、プログラムのリスクはくなります。ときどきがにれることがあります。ののは、もものではられません。メモリのは、によってはなものになります。

のループは、しいメモリをにりてるである`getline()`びすことによって、メモリをせずに、になメモリリ―クを`getline()`リ―クのです。

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    char *line = NULL;
    size_t size = 0;

    /* The loop below leaks memory as fast as it can */

    for(;;) {
        getline(&line, &size, stdin); /* New memory implicitly allocated */

        /* <do whatever> */

        line = NULL;
    }

    return 0;
}
```

に、のコードでは`getline()`もしていますが、はりてられたメモリがしくされ、リ―クがされます。

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    char *line = NULL;
    size_t size = 0;

    for(;;) {
        if (getline(&line, &size, stdin) < 0) {
            free(line);
```

```
            line = NULL;

            /* Handle failure such as setting flag, breaking out of loop and/or exiting */
        }

        /* <do whatever> */

        free(line);
        line = NULL;

    }

    return 0;
}
```

メモリリークは、ずしもなをもたらすとはらず、ずしもなではない。「ベストプラクティス」は、なポイントとでにメモリをするようしますが、メモリののをし、メモリののリスクをするためにはがあります。えば、プログラムがとスコープでされている、りてののはあまりにもさいとえられます。その、なりてをバイパスすることはとえられます。えば、のオペレーティングシステムのほとんどは、プログラムの、`exit()`へのシステムコール、プロセスの、または`main()`にして、プログラムがしたときにされるすべてのメモリをにします。したプログラムのでにメモリをすると、にはしたり、パフォーマンスのをくがあります。

なメモリがである、りてはし、ハンドリングはコールスタックのなレベルでされるべきです。の`getline()`は、びしにされたメモリをするだけでなく、いくつかのでするのあるライブラリであるため、いです。したがって、C APIをしてドキュメントマニュアルページをんで、エラーとメモリににをい、されたメモリをするがどのソフトウェアにあるかをることがです。

もう1つのなメモリのは、メモリポインタをするメモリがされたに、メモリポインタをNULLにしてすることです。そのため、されたメモリにアクセスするためNULL /ガベージ·データのみりやデータのきみやプログラム·クラッシュなどのなをきこすがあります。のオペレーティングシステムでは、メモリの0 `NULL` をすることは、Cでされるようにnopですつまり、です。ポインタをNULLにすることで、ポインタ`free()`されます。メモリをにすると、にがかかり、し、をするのがになることにしてください。

あまりにもくのコピー

```
char buf[8]; /* tiny buffer, easy to overflow */

printf("What is your name?\n");
scanf("%s", buf); /* WRONG */
scanf("%7s", buf); /* RIGHT */
```

ユーザーが7よりいをするとnullでは-1、バッファ`buf`ろのメモリはきされます。この、のがします。のあるハッカーは、リターンアドレスをきし、ハッカーのなコードのアドレスにするために、これをすることがよくあります。

reallocのりをにコピーすることをれる

realloc がすると、 NULL し NULL 。 realloc のりにのバッファのをし、 NULL すと、のバッファいポインタがわれ、 メモリリークがします。は、なポインタにコピーすることで、にそれがNULLでないは、 その、のバッファにコピーします。

```
char *buf, *tmp;

buf = malloc(...);
...

/* WRONG */
if ((buf = realloc(buf, 16)) == NULL)
    perror("realloc");

/* RIGHT */
if ((tmp = realloc(buf, 16)) != NULL)
    buf = tmp;
else
    perror("realloc");
```

の

float 、 double 、および long double は、がでバイナリのをすため、のをにすことはできません。1/3のような10のをりすのとじように、2でもではできないもあります1/3、さらになのは1/10など。をしないでください。わりにデルタをしてください。

```
#include <float.h> // for DBL_EPSILON and FLT_EPSILON
#include <math.h>  // for fabs()

int main(void)
{
    double a = 0.1; // imprecise: (binary) 0.000110...

    // may be false or true
    if (a + a + a + a + a + a + a + a + a + a == 1.0) {
        printf("10 * 0.1 is indeed 1.0. This is not guaranteed in the general case.\n");
    }

    // Using a small delta value.
    if (fabs(a + a + a + a + a + a + a + a + a + a - 1.0) < 0.000001) {
        // C99 5.2.4.2.2p8 guarantees at least 10 decimal digits
        // of precision for the double type.
        printf("10 * 0.1 is almost 1.0.\n");
    }

    return 0;
}
```

もうつの

```
gcc -O3   -g   -I./inc   -std=c11   -Wall -Wextra -Werror -Wmissing-prototypes -Wstrict-
prototypes  -Wold-style-definition       rd11.c -o rd11 -L./lib -lsoq
#include <stdio.h>
#include <math.h>

static inline double rel_diff(double a, double b)
```

```
{
    return fabs(a - b) / fmax(fabs(a), fabs(b));
}

int main(void)
{
    double d1 = 3.14159265358979;
    double d2 = 355.0 / 113.0;

    double epsilon = 1.0;
    for (int i = 0; i < 10; i++)
    {
        if (rel_diff(d1, d2) < epsilon)
            printf("%d:%.10f <=> %.10f within tolerance %.10f (rel diff %.4E)\n",
                   i, d1, d2, epsilon, rel_diff(d1, d2));
        else
            printf("%d:%.10f <=> %.10f out of tolerance %.10f (rel diff %.4E)\n",
                   i, d1, d2, epsilon, rel_diff(d1, d2));
        epsilon /= 10.0;
    }
    return 0;
}
```

```
0:3.1415926536 <=> 3.1415929204 within tolerance 1.0000000000 (rel diff 8.4914E-08)
1:3.1415926536 <=> 3.1415929204 within tolerance 0.1000000000 (rel diff 8.4914E-08)
2:3.1415926536 <=> 3.1415929204 within tolerance 0.0100000000 (rel diff 8.4914E-08)
3:3.1415926536 <=> 3.1415929204 within tolerance 0.0010000000 (rel diff 8.4914E-08)
4:3.1415926536 <=> 3.1415929204 within tolerance 0.0001000000 (rel diff 8.4914E-08)
5:3.1415926536 <=> 3.1415929204 within tolerance 0.0000100000 (rel diff 8.4914E-08)
6:3.1415926536 <=> 3.1415929204 within tolerance 0.0000010000 (rel diff 8.4914E-08)
7:3.1415926536 <=> 3.1415929204 within tolerance 0.0000001000 (rel diff 8.4914E-08)
8:3.1415926536 <=> 3.1415929204 out of tolerance 0.0000000100 (rel diff 8.4914E-08)
9:3.1415926536 <=> 3.1415929204 out of tolerance 0.0000000010 (rel diff 8.4914E-08)
```

ポインタでなスケーリングをう

ポインタでは、ポインタにまたはされるはアドレスのではなく、するのとしてされます。

```
#include <stdio.h>

int main(void) {
    int array[] = {1, 2, 3, 4, 5};
    int *ptr = &array[0];
    int *ptr2 = ptr + sizeof(int) * 2; /* wrong */
    printf("%d %d\n", *ptr, *ptr2);
    return 0;
}
```

このコードは、 `ptr2` りてられたポインタをするになスケーリングをいます。の32ビットでな `sizeof(int)` が4の、は `array[0]` 8つの `array[0]` ろをし、 のをびします。

`ptr2` `array[0]` に2つのをすようにするには、に2をするだけです。

```
#include <stdio.h>
```

```
int main(void) {
    int array[] = {1, 2, 3, 4, 5};
    int *ptr = &array[0];
    int *ptr2 = ptr + 2;
    printf("%d %d\n", *ptr, *ptr2); /* "1 3" will be printed */
    return 0;
}
```

をしたなポインタはするがあるため、のをするほうがいかもしれません。

```
#include <stdio.h>

int main(void) {
    int array[] = {1, 2, 3, 4, 5};
    int *ptr = &array[0];
    int *ptr2 = &ptr[2];
    printf("%d %d\n", *ptr, *ptr2); /* "1 3" will be printed */
    return 0;
}
```

`E1[E2]`である`(*((E1)+(E2)))` N1570 6.5.2.1、2、び`&(E1[E2])`とである`((E1)+(E2))` N1570 6.5.3.2、102。

あるいは、ポインタがましい、ポインタをキャストしてなるデータにアドレスすると、バイトアドレスがになります。ただし、 エンディアンがになることがあります。また、「へのポインタ」のにキャストするとなエイリアシングのがします。

```
#include <stdio.h>

int main(void) {
    int array[3] = {1,2,3};  // 4 bytes * 3 allocated
    unsigned char *ptr = (unsigned char *) array;  // unsigned chars only take 1 byte
    /*
     * Now any pointer arithmetic on ptr will match
     * bytes in memory.  ptr can be treated like it
     * was declared as: unsigned char ptr[12];
     */

    return 0;
}
```

マクロはなのです

マクロはなです。 には、のではなく、トークンでします。

```
#include <stdio.h>

#define SQUARE(x) x*x

int main(void) {
    printf("%d\n", SQUARE(1+2));
    return 0;
}
```

このコ―ドでは`9 3*3`をするとされますが、マクロは`1+2*1+2`されるためには`5`がされます。

このをするには、とのマクロをカッコでむがあります。

```
#include <stdio.h>

#define SQUARE(x) ((x)*(x))

int main(void) {
    printf("%d\n", SQUARE(1+2));
    return 0;
}
```

のは、マクロのがされることがされていないことです。それらはまったくされないかもしれないし、されるかもしれない。

```
#include <stdio.h>

#define MIN(x, y) ((x) <= (y) ? (x) : (y))

int main(void) {
    int a = 0;
    printf("%d\n", MIN(a++, 10));
    printf("a = %d\n", a);
    return 0;
}
```

このコ―ドでは、マクロは`((a++) <= (10) ? (a++) : (10))`ます。 `a++ 0`は`10`よりさいので`a++`は2され、それは`a`のをり、 `MIN`からされるものはあなたとはなるかもしれません。

これは、をすることでできますが、はでされますが、マクロはによってがあることにしてください。

```
#include <stdio.h>

int min(int x, int y) {
    return x <= y ? x : y;
}

int main(void) {
    int a = 0;
    printf("%d\n", min(a++, 10));
    printf("a = %d\n", a);
    return 0;
}
```

、のはされていますが、この`min`は`double`データをりてることなくすることはできません。

マクロ･ディレクティブには、の2つのタイプがあります。

```
#define OBJECT_LIKE_MACRO     followed by a "replacement list" of preprocessor tokens
#define FUNCTION_LIKE_MACRO(with, arguments) followed by a replacement list
```

これらの2つのタイプのマクロをするものは、 `#define`のにくです *lparen*の、それはのようなマクロです。そうでなければ、それはオブジェクトのようなマクロです。のようなマクロをくことがされているは、マクロののにをれてはいけません (なはこれをしてください。

C99

C99では、 `static inline int min(int x, int y) { … }`できます。

C11

C11では、 `min`にして 'ジェネリック'をくことができます。

```
#include <stdio.h>

#define min(x, y) _Generic((x), \
                        long double: min_ld, \
                        unsigned long long: min_ull, \
                        default: min_i \
                        )(x, y)

#define gen_min(suffix, type) \
    static inline type min_##suffix(type x, type y) { return (x < y) ? x : y; }

gen_min(ld, long double)
gen_min(ull, unsigned long long)
gen_min(i, int)

int main(void)
{
    unsigned long long ull1 = 50ULL;
    unsigned long long ull2 = 37ULL;
    printf("min(%llu, %llu) = %llu\n", ull1, ull2, min(ull1, ull2));
    long double ld1 = 3.141592653L;
    long double ld2 = 3.141592652L;
    printf("min(%.10Lf, %.10Lf) = %.10Lf\n", ld1, ld2, min(ld1, ld2));
    int i1 = 3141653;
    int i2 = 3141652;
    printf("min(%d, %d) = %d\n", i1, i2, min(i1, i2));
    return 0;
}
```

なは、 `double`、 `float`、 `long long`、 `unsigned long`、 `long`、 `unsigned`ようなよりくのとな`gen_min`マクロびしですることができます。

リンクののエラー

コンパイルのもなエラーの1つは、リンクでします。エラーはのようになります。

```
$ gcc undefined_reference.c
/tmp/ccoXhwF0.o: In function `main':
undefined_reference.c:(.text+0x15): undefined reference to `foo'
collect2: error: ld returned 1 exit status
$
```

このエラーをしたコードをてみましょう

```
int foo(void);

int main(int argc, char **argv)
{
    int foo_val;
    foo_val = foo();
    return foo_val;
}
```

foo `int foo();` のはここにありますが、ののはありません。コンパイラにヘッダをしましたが、どこにもそのようなはされていなかったので、コンパイルはわりますが、リンカは`Undefined reference`エラーでします。
さなプログラムでこのエラーをするには、fooのをするだけです。

```
/* Declaration of foo */
int foo(void);

/* Definition of foo */
int foo(void)
{
    return 5;
}

int main(int argc, char **argv)
{
    int foo_val;
    foo_val = foo();
    return foo_val;
}
```

はこのコードをコンパイルします。わりに、 `foo()` ソースがのソースファイル`foo.c`にあり、 `foo.c` と`undefined_reference.c`にまれる`foo()`をするためのヘッダ`foo.h`があります。に、 `foo.c`と`undefined_reference.c`のオブジェクトファイルをリンクするか、のソースファイルをコンパイルします。

```
$ gcc -c undefined_reference.c
$ gcc -c foo.c
$ gcc -o working_program undefined_reference.o foo.o
$
```

または

```
$ gcc -o working_program undefined_reference.c foo.c
$
```

もっとなケースは、コードのようにライブラリがわっているです

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
int main(int argc, char **argv)
{
    double first;
    double second;
    double power;

    if (argc != 3)
    {
        fprintf(stderr, "Usage: %s <denom> <nom>\n", argv[0]);
        return EXIT_FAILURE;
    }

    /* Translate user input to numbers, extra error checking
     * should be done here. */
    first = strtod(argv[1], NULL);
    second = strtod(argv[2], NULL);

    /* Use function pow() from libm - this will cause a linkage
     * error unless this code is compiled against libm! */
    power = pow(first, second);

    printf("%f to the power of %f = %f\n", first, second, power);

    return EXIT_SUCCESS;
}
```

コードはにしいですが、 `#include <math.h>`から`pow()`がするため、コンパイルとリンクをみますが、のようなエラーがします。

```
$ gcc no_library_in_link.c -o no_library_in_link
/tmp/ccduQQqA.o: In function `main':
no_library_in_link.c:(.text+0x8b): undefined reference to `pow'
collect2: error: ld returned 1 exit status
$
```

これは、リンクで`pow()` がつからなかったためにします。これをするには、 `-lm`フラグをして`libm`というライブラリにリンクしたいとするがあります。 `-lm`がないmacOSなどのプラットフォームがあることにしてください。しかし、のをすると、ライブラリがになります。

そこで、コンパイルのをもうします。は、ライブラリをしますソースファイルまたはオブジェクトファイルのろ。

```
$ gcc no_library_in_link.c -lm -o library_in_link_cmd
$ ./library_in_link_cmd 2 4
2.000000 to the power of 4.000000 = 16.000000
$
```

そしてそれはします

アレイのの

、ポインタのなどをするコードのなは、 `Type**`と`Type[M][N]`がになるであるというです。

```
#include <stdio.h>

void print_strings(char **strings, size_t n)
{
    size_t i;
    for (i = 0; i < n; i++)
        puts(strings[i]);
}

int main(void)
{
    char s[4][20] = {"Example 1", "Example 2", "Example 3", "Example 4"};
    print_strings(s, 4);
    return 0;
}
```

コンパイラの

```
file1.c: In function 'main':
file1.c:13:23: error: passing argument 1 of 'print_strings' from incompatible pointer type [-
Wincompatible-pointer-types]
         print_strings(strings, 4);
                       ^
file1.c:3:10: note: expected 'char **' but argument is of type 'char (*)[20]'
     void print_strings(char **strings, size_t n)
```

このエラーは、`main`が`print_strings`にされたことをしています。また、`print_strings`がするをす`print_strings`と、`main`からされたもまれます。

このはアレイとばれるものがです。がこる`s`そのと`char[4][20]` 20の4のアレイがにされると、それはあなたがいたかのように、そののへのポインタになりされる`&s[0]`し、`char (*)[20]` 20の1へのポインタ。これは、ポインターの、の3-Dの、およびへのポインターのをむ、どのでもします。は、がしたときのをすです。タイプのは、がこるかをすためにされています。

`char [20]`	**20**	`char *`	ポインタ**1**
`char [4][20]`	20のうちの**4**つの	`char (*)[20]`	20のうちの**1**つの へのポインタ
`char *[4]`	の 1つのへの**4**つのポインタ	`char **`	1つのcharへのポインタ**1** へのポインタ
`char [3][4][20]`	20の4の**3**	`char (*)[4][20]`	ポインタ 20の4の**1**
`char (*[4])[20]`	20の1にする**4**つのポインタ	`char (**)[20]`	**1**の20からなる1つのへのポインタへのポインタ

がポインタにするがある、ポインタはなくとも1のとなすことができます。はヌルポインタで、これはもさずにはではありません。

アレイのはだけこります。がポインタにした、ではなくポインタになります。へのポインタがあ

っても、ポインタがなくとも1つののとみなされるがあるので、のはにしていることにしてください。

つまり、 `char (*)[20]` へのポインタはしてポインタ `char **` へのポインタになりません。 `print_strings`をするには、にしいをけれるようにしてください

```
void print_strings(char (*strings)[20], size_t n)
/* OR */
void print_strings(char strings[][20], size_t n)
```

は、 `print_strings`をのにしてにしたいにします`print_strings`ではなく30のがあるはどうなりますかまたは50えはパラメ―タのにのパラメ―タをすることです

```
#include <stdio.h>

/*
 * Note the rearranged parameters and the change in the parameter name
 * from the previous definitions:
 *      n (number of strings)
 *   => scount (string count)
 *
 * Of course, you could also use one of the following highly recommended forms
 * for the `strings` parameter instead:
 *
 *    char strings[scount][ccount]
 *    char strings[][ccount]
 */
void print_strings(size_t scount, size_t ccount, char (*strings)[ccount])
{
    size_t i;
    for (i = 0; i < scount; i++)
        puts(strings[i]);
}

int main(void)
{
    char s[4][20] = {"Example 1", "Example 2", "Example 3", "Example 4"};
    print_strings(4, 20, s);
    return 0;
}
```

これをコンパイルすると、エラ―はせず、したがられます。

```
Example 1
Example 2
Example 3
Example 4
```

しないを"の"をするにす

`malloc` 、 `calloc` 、および`realloc`でをりてる、なパタ―ンはのびしでのをりてることですびしが1だけしてもル―プであっても。

```
/* Could also be `int **` with malloc used to allocate outer array. */
int *array[4];
int i;

/* Allocate 4 arrays of 16 ints. */
for (i = 0; i < 4; i++)
    array[i] = malloc(16 * sizeof(*array[i]));
```

の1つののとののののとののバイトのは、「の」たとえば、 `int array[4][16];`

```
/* 0x40003c, 0x402000 */
printf("%p, %p\n", (void *)(array[0] + 15), (void *)array[1]);
```

`int`のサイズをすると、8328バイト8132-4のがられます。これは2032の`int`です。これはです。「」にはにギャップがありません。

"の"をするでにりてられたをするがあるは、 `int *`のオブジェクトをりててをしてをうがあります。

```
void func(int M, int N, int *array);
...

/* Equivalent to declaring `int array[M][N] = {{0}};` and assigning to array4_16[i][j]. */
int *array;
int M = 4, N = 16;
array = calloc(M, N * sizeof(*array));
array[i * N + j] = 1;
func(M, N, array);
```

`N`がではなくマクロまたはリテラルである、コードはへのポインタをりてたに、よりな2をうことができます。

```
void func(int M, int N, int *array);
#define N 16
void func_N(int M, int (*array)[N]);
...

int M = 4;
int (*array)[N];
array = calloc(M, sizeof(*array));
array[i][j] = 1;

/* Cast to `int *` works here because `array` is a single block of M*N ints with no gaps,
   just like `int array2[M * N];` and `int array3[M][N];` would be. */
func(M, N, (int *)array);
func_N(M, array);
```

C99

`N`がマクロまたはリテラルでない、 `array`はVLAをします。これは、 `int *`キャストすることによって`func`とともにすることができます`int *`としい`func_vla`は`func_vla`をきえ`func_N`

```
void func(int M, int N, int *array);
```

```
void func_vla(int M, int N, int array[M][N]);
...

int M = 4, N = 16;
int (*array)[N];
array = calloc(M, sizeof(*array));
array[i][j] = 1;
func(M, N, (int *)array);
func_vla(M, N, array);
```

C11

VLAはC11ではオプションです。がC11をサポートしていて、 `__STDC_NO_VLA__` マクロを1にしているは、C99よりのにしています。

リテラルのわりにをするも

Cでは、とリテラルはなるものです。

`'a'`ようにでまれたはです。は、をすコードのをつです。 `'abc'`ようなのをつをどのようにするかは、です。

`"abc"`ようなでまれたゼロのはリテラルです。リテラルは、が`char`のなです。ダブルクォートとヌルをえるはなので、 `"abc"`は4つの `{'a', 'b', 'c', '\0'}` があります。

このでは、をするがされています。このは、のでポインタにされ、されたポインタがであるはほとんどないので、このではのがびされます 。

```
#include <stdio.h>

int main(void) {
    const char *hello = 'hello, world'; /* bad */
    puts(hello);
    return 0;
}
```

このでは、をするがあるはリテラルがされます。リテラルからされたポインタは、のでにされ、ので`char`にされます。 するをできないきのにをするはであり、 `char`がされているかどうかはです。はなものになります。

```
#include <stdio.h>

int main(void) {
    char c = "a"; /* bad */
    printf("%c\n", c);
    return 0;
}
```

ほとんどの、コンパイラはこれらのについてをうでしょう。そうでないは、よりくのコンパイラオプションをするか、よりいコンパイラをすることをおめします。

ライブラリのりをする

Cライブラリのほとんどすべてのは、したときにかをします。えば、 `malloc`は、したときににによってりてられたメモリブロックへのポインタと、されたメモリブロックのりてにしたにはヌルポインタをします。したがって、デバッグをにするためににりをチェックするがあります。

これはいです

```
char* x = malloc(100000000000UL * sizeof *x);
/* more code */
scanf("%s", x); /* This might invoke undefined behaviour and if lucky causes a segmentation
violation, unless your system has a lot of memory */
```

これはいい

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    char* x = malloc(100000000000UL * sizeof *x);
    if (x == NULL) {
        perror("malloc() failed");
        exit(EXIT_FAILURE);
    }

    if (scanf("%s", x) != 1) {
        fprintf(stderr, "could not read string\n");
        free(x);
        exit(EXIT_FAILURE);
    }

    /* Do stuff with x. */

    /* Clean up. */
    free(x);

    return EXIT_SUCCESS;
}
```

このでエラーのをすぐにることができます。そうしないと、ったでバグをしてをやすことがあります。

は、の**scanf**びしではされません

このプログラム

```
#include <stdio.h>
#include <string.h>

int main(void) {
    int num = 0;
    char str[128], *lf;
```

```
    scanf("%d", &num);
    fgets(str, sizeof(str), stdin);

    if ((lf = strchr(str, '\n')) != NULL) *lf = '\0';
    printf("%d \"%s\"\n", num, str);
    return 0;
}
```

このでされます

```
42
life
```

はのようになります`42 ""`ではなくの`42 "life"`。

これは、 `42`のは`scanf()`びしでされず、 `life`むに`fgets()`によってされるためです。それで`fgets()` `life`むにをめます。

このをするには、えば、オンラインシステムのをするときに、lineのがかっているときになの1つは、 `scanf()`せず、 `fgets()`してすべてのをむことです。 `sscanf()`をすると、みられたをできます。

```
#include <stdio.h>
#include <string.h>

int main(void) {
    int num = 0;
    char line_buffer[128] = "", str[128], *lf;

    fgets(line_buffer, sizeof(line_buffer), stdin);
    sscanf(line_buffer, "%d", &num);
    fgets(str, sizeof(str), stdin);

    if ((lf = strchr(str, '\n')) != NULL) *lf = '\0';
    printf("%d \"%s\"\n", num, str);
    return 0;
}
```

もうつのは、 `scanf()`をった、そして`fgets()`をうにをつまでむことです。

```
#include <stdio.h>
#include <string.h>

int main(void) {
    int num = 0;
    char str[128], *lf;
    int c;

    scanf("%d", &num);
    while ((c = getchar()) != '\n' && c != EOF);
    fgets(str, sizeof(str), stdin);

    if ((lf = strchr(str, '\n')) != NULL) *lf = '\0';
    printf("%d \"%s\"\n", num, str);
```

```
    return 0;
}
```

#defineにセミコロンをする

Cプリプロセッサでして Cのとしてうのはですが、プリプロセッサはなるテキストメカニズムなので、いです。たとえば、のようにします。

```
/* WRONG */
#define MAX 100;
int arr[MAX];
```

コ―ドはのようにされます。

```
int arr[100;];
```

これはエラ―です。は、セミコロンを`#define`からすることです。 `#define`をセミコロンでわらせるのはいなくいです。

コメントはれにできません

Cでは、のコメント/*と*/はネストしません。

このスタイルのコメントをしてコ―ドブロックまたはブロックにをけると、のようになります。

```
/*
 * max(): Finds the largest integer in an array and returns it.
 * If the array length is less than 1, the result is undefined.
 * arr: The array of integers to search.
 * num: The number of integers in arr.
 */
int max(int arr[], int num)
{
    int max = arr[0];
    for (int i = 0; i < num; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}
```

あなたはそれをにコメントにすることはできません

```
//Trying to comment out the block...
/*

/*
 * max(): Finds the largest integer in an array and returns it.
 * If the array length is less than 1, the result is undefined.
 * arr: The array of integers to search.
 * num: The number of integers in arr.
 */
int max(int arr[], int num)
```

```
{
    int max = arr[0];
    for (int i = 0; i < num; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

//Causes an error on the line below...
*/
```

1つのは、C99スタイルのコメントをすることです。

```
// max(): Finds the largest integer in an array and returns it.
// If the array length is less than 1, the result is undefined.
// arr: The array of integers to search.
// num: The number of integers in arr.
int max(int arr[], int num)
{
    int max = arr[0];
    for (int i = 0; i < num; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}
```

これで、ブロックをにコメントにすることができます。

```
/*

// max(): Finds the largest integer in an array and returns it.
// If the array length is less than 1, the result is undefined.
// arr: The array of integers to search.
// num: The number of integers in arr.
int max(int arr[], int num)
{
    int max = arr[0];
    for (int i = 0; i < num; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

*/
```

のは、 `#ifdef`または`#ifndef`プリプロセッサディレクティブをわりにして、コメントをしてコードをにすることをけることです。これらのディレクティブはネストして、みのスタイルでコードにコメントすることができます。

```
#define DISABLE_MAX /* Remove or comment this line to enable max() code block */

#ifdef DISABLE_MAX
/*
 * max(): Finds the largest integer in an array and returns it.
 * If the array length is less than 1, the result is undefined.
 * arr: The array of integers to search.
```

```
 * num: The number of integers in arr.
 */
int max(int arr[], int num)
{
    int max = arr[0];
    for (int i = 0; i < num; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}
#endif
```

いくつかのガイドは、コードセクションにしてコメントをつけてはならないことをしており、コードをににするは`#if 0`ディレクティブをすることができます。

コードセクションをブロックするには、 *#if 0*をしてください。

のオーバーステッピング

は0からまります。つまり、インデックスはに0からまり、インデックスのさから1をいたでします。したがって、のコードはののをせず、するにガベージをします。

```
#include <stdio.h>

int main(void)
{
    int x = 0;
    int myArray[5] = {1, 2, 3, 4, 5}; //Declaring 5 elements

    for(x = 1; x <= 5; x++) //Looping from 1 till 5.
       printf("%d\t", myArray[x]);

    printf("\n");
    return 0;
}
```

```
2 3 4 5 GarbageValue
```

は、ましいをるためのしいをしています。

```
#include <stdio.h>

int main(void)
{
    int x = 0;
    int myArray[5] = {1, 2, 3, 4, 5}; //Declaring 5 elements

    for(x = 0; x < 5; x++) //Looping from 0 till 4.
       printf("%d\t", myArray[x]);

    printf("\n");
    return 0;
}
```

```
1 2 3 4 5
```

アレイのさをることがです。そうしないと、バッファがれたり、のメモリにアクセスしてセグメンテーションがするがあります。

- がけている

のをすることは、のなです。

があありません

```
#include <stdio.h>

int factorial(int n)
{
       return n * factorial(n - 1);
}

int main()
{
    printf("Factorial %d = %d\n", 3, factorial(3));
    return 0;
}
```

な `Segmentation fault: 11`

こののは、ループがにループし、セグメントエラーがすることです。をするにはがです。

ベース

```
#include <stdio.h>

int factorial(int n)
{
    if (n == 1) // Base Condition, very crucial in designing the recursive functions.
    {
       return 1;
    }
    else
    {
       return n * factorial(n - 1);
    }
}

int main()
{
    printf("Factorial %d = %d\n", 3, factorial(3));
    return 0;
}
```

サンプル

```
Factorial 3 = 6
```

このは、$_n$が1になるとすぐにします $_n$のがさい - $_{int}$が32ビットの、は$_{12}$です。

うべき

1. アルゴリズムをします。プログラムでは、しばしばまるシーードがです。これは、にされるパラメーターをするか、であるがのためのシーードをするゲートウェイをすることによってされます。
2. されているのがベースケースとするかどうかをします。そうであれば、をしてします。
3. またはよりなサブまたはサブのからえをします。
4. サブにしてアルゴリズムをします。
5. えをするにをする。
6. をします。

ソース

の "のチェック

オリジナルのCにはのブールがなかったので、 `bool` 、 `true` 、 `false`はのをたず、しばしばプログラマによってされました。`true`は1とされ、 `false`は0とされます。

C99

C99にはみ`_Bool`と、 `bool` `_Bool`、 `false`および`true`をするヘッダー`<stdbool.h>`がされてい`true`。また、 `bool` 、 `true`および`false`をすることもできますが、これはされたであることにしてください。

さらになのは、は、するものがすべてとみなし、0のをとみなします。えば

```
/* Return 'true' if the most significant bit is set */
bool isUpperBitSet(uint8_t bitField)
{
    if ((bitField & 0x80) == true)  /* Comparison only succeeds if true is 0x80 and bitField
has that bit set */
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

のでは、はビットがセットされているかどうかをチェックし、 `true`あれば`true`をし`true`。しかし、 `true`にチェックする`true`により、 `if`は、 `(bitfield & 0x80)` `true`とされているとされるにのみします。は`1`あり、ほとんど`0x80`ありません。あなたがするケースにしてにチェックするか、

```
/* Return 'true' if the most significant bit is set */
bool isUpperBitSet(uint8_t bitField)
{
    if ((bitField & 0x80) == 0x80) /* Explicitly test for the case we expect */
    {
        return true;
```

```
    }
    else
    {
        return false;
    }
}
```

または、ゼロのをtrueとしてします。

```
/* Return 'true' if the most significant bit is set */
bool isUpperBitSet(uint8_t bitField)
{
    /* If upper bit is set, result is 0x80 which the if will evaluate as true */
    if (bitField & 0x80)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

リテラルはデフォルトで**double**です

`float`のをリテラルにするときや、 `0.1`などののリテラルが`double`であるため、それらをリテラルとするときはがです。これはきにつながるがあります

```
#include <stdio.h>
int main() {
    float  n;
    n = 0.1;
    if (n > 0.1) printf("Wierd\n");
    return 0;
}
// Prints "Wierd" when n is float
```

ここで、 `n`はされ、にめられ、その0.100000001490116121というになります。に、 `n`はにされて`0.1`リテラル0.10000000000000001にしいとされ、がじます。

めにえて、 `float`を`double`リテラルとさ`float`と、ハードウェアでをサポートしていないプラットフォームではパフォーマンスがします。

オンラインでのとしをむ https://riptutorial.com/ja/c/topic/2006/のとし

37:

は`enum`キーワードとオプションのでされ、そのにでまれたリストがきます。

は`int`です。

*enumerator-list*にはなくとも1つのがあります。

は、オプションで`int`のに "りて"られることがあります。

はであり、 `char` 、きまたはなしのいずれかとがあります。これまでにされているのはです。いずれのでも、されるは、のにしてされたすべてのをすことができなければなりません。

にがりてられておらず、それがリストの1のエントリであれば、それは`0`をとり、そうでなければ、 リストののエントリのに1をえたをとる。

の「」をすると、じのなるがじをつことがあります。

Examples

な

は、ユーザーのデータであり、でされ、にはがけられます。キーワード`enum`は、をするためにされます。

`int`または`string/ char*`かわりに`enum`をすると、コンパイルのチェックがえ、ながされるのをけることができます。

1

```
enum color{ RED, GREEN, BLUE };

void printColor(enum color chosenColor)
{
    const char *color_name = "Invalid color";
    switch (chosenColor)
    {
       case RED:
         color_name = "RED";
         break;

       case GREEN:
        color_name = "GREEN";
        break;

       case BLUE:
        color_name = "BLUE";
        break;
    }
```

```
    printf("%s\n", color_name);
}
```

mainがのようにされているたとえば

```
int main(){
    enum color chosenColor;
    printf("Enter a number between 0 and 2");
    scanf("%d", (int*)&chosenColor);
    printColor(chosenColor);
    return 0;
}
```

C99

2

このでは、C99にされているされたをしています。

```
enum week{ MON, TUE, WED, THU, FRI, SAT, SUN };

static const char* const dow[] = {
  [MON] = "Mon", [TUE] = "Tue", [WED] = "Wed",
  [THU] = "Thu", [FRI] = "Fri", [SAT] = "Sat", [SUN] = "Sun" };

void printDayOfWeek(enum week day)
{
   printf("%s\n", dow[day]);
}
```

チェックをしたじ

```
enum week{ DOW_INVALID = -1,
  MON, TUE, WED, THU, FRI, SAT, SUN,
  DOW_MAX };

static const char* const dow[] = {
  [MON] = "Mon", [TUE] = "Tue", [WED] = "Wed",
  [THU] = "Thu", [FRI] = "Fri", [SAT] = "Sat", [SUN] = "Sun" };

void printDayOfWeek(enum week day)
{
   assert(day > DOW_INVALID && day < DOW_MAX);
   printf("%s\n", dow[day]);
}
```

Typedef enum

にをけるいくつかのとがあります。 1つは、 `enum`キーワードのにタグをすることです。

```
enum color
{
    RED,
```

```
    GREEN,
    BLUE
};
```

このは、キーワードとタグでにのようにするがあります。

```
enum color chosenColor = RED;
```

enumするときにtypedefすると、タグをしてenumキーワードなしでtypeをできます。

```
typedef enum
{
    RED,
    GREEN,
    BLUE
} color;

color chosenColor = RED;
```

しかし、このの、タグをにしていないため、 enum colorとしてすることはできません。なの1つは、 enumキーワードのにかかわらずじをできるように、をすることです。これは、 C ++とがあるというながありま す

```
enum color                /* as in the first example */
{
    RED,
    GREEN,
    BLUE
};
typedef enum color color; /* also a typedef of same identifier */

color chosenColor  = RED;
enum color defaultColor = BLUE;
```

```
void printColor()
{
    if (chosenColor == RED)
    {
        printf("RED\n");
    }
    else if (chosenColor == GREEN)
    {
        printf("GREEN\n");
    }
    else if (chosenColor == BLUE)
    {
        printf("BLUE\n");
    }
}
```

typedefについてはTypedefをしてください。

をつ

はであるはありません。

```
#include <stdlib.h> /* for EXIT_SUCCESS */
#include <stdio.h> /* for printf() */


enum Dupes
{
   Base, /* Takes 0 */
   One, /* Takes Base + 1 */
   Two, /* Takes One + 1 */
   Negative = -1,
   AnotherZero /* Takes Negative + 1 == 0, sigh */
};

int main(void)
{
  printf("Base = %d\n", Base);
  printf("One = %d\n", One);
  printf("Two = %d\n", Two);
  printf("Negative = %d\n", Negative);
  printf("AnotherZero = %d\n", AnotherZero);

  return EXIT_SUCCESS;
}
```

サンプルはをします

```
Base = 0
One = 1
Two = 2
Negative = -1
AnotherZero = 0
```

typenameのない

は、をけずにすることもできます

```
  enum { buffersize = 256, };
  static unsigned char buffer [buffersize] = { 0 };
```

これにより、こののようにとしてできる`int`のコンパイルをすることができます。

オンラインでをむ https://riptutorial.com/ja/c/topic/5460/

38:

Examples

Cでのの

ながない、と`static`はゼロにされることがされます。 `register`をむは、 な 1 つまりゴミをちます。

スカラーは、のにとをけてすると、されます。

```
int x = 1;
char squota = '\'';
long day = 1000L * 60L * 60L * 24L; /* milliseconds/day */
```

および`static`の、は 2でなければなりません。プログラムのがされるににがされます。

と`register`の、イニシャライザはにされず、びしであってもにされたをむであってもいません。

たとえば、のコードスニペットをしてください

```
int binsearch(int x, int v[], int n)
{
   int low = 0;
   int high = n - 1;
   int mid;
   ...
}
```

のわりに

```
   int low, high, mid;

   low = 0;
   high = n - 1;
```

、のはのにぎません。どちらのをむかはにのです。のはするからえにくく、くれているため、にはなりてをします。、はなりされるにされるべきです。

の

は、そのにいて、カッコでまれたコンマでられたイニシャライザのリストですることができます。

たとえば、のをのでするには、のようにします。

```
int days_of_month[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }
```

1は、このでがゼロとしてエンコードされることにしてください。

のサイズをすると、コンパイラはイニシャライザをカウントすることでさをします。この、12あります。

のがされたサイズよりない、のタイプはすべてのタイプのでゼロになります。

がすぎるとエラーになります。イニシャライザのりしをするなはありませんが、GCCにはそのようながあります。

C99

C89 / C90のバージョンのCでは、のでをするはありませんでした。

C99

C99では、 されたをしてののをし、されていないはゼロにします。

の

はのなケースです。かっこののわりにをうことができます

```
char chr_array[] = "hello";
```

よりいがのもののである

```
char chr_array[] = { 'h', 'e', 'l', 'l', 'o', '\0' };
```

この、のサイズは6です5+の'\0'。

1 Cでされた、されていないはどうなりますかそれはがありますか

2 は、コンパイルにできるものとしてされています。だから、 `int global_var = f();`です。もうつのよくあるは、 `const`されたをとしてえることです。 Cでは、 `const`は「コンパイル」ではなく「みり」をします。だから、 `const int SIZE = 10; int global_arr[SIZE]; const int SIZE = 10; int global_var = SIZE;` Cではではない

のとの

とのは、のメンバごとに1つのを、でんだのでできます。

```
struct Date
{
    int year;
    int month;
    int day;
};

struct Date us_independence_day = { 1776, 7, 4 };
```

```
struct Date uk_battles[] =
{
    { 1066, 10, 14 },    // Battle of Hastings
    { 1815,  6, 18 },    // Battle of Waterloo
    { 1805, 10, 21 },    // Battle of Trafalgar
};
```

のはなしですることができ、の1990はうこともあるにはしばしばそれらがなくてもかれていたことにしてください。

```
struct Date uk_battles[] =
{
    1066, 10, 14,    // Battle of Hastings
    1815,  6, 18,    // Battle of Waterloo
    1805, 10, 21,    // Battle of Trafalgar
};
```

これはしますが、ではありません。しいコードでこのをしようとするべきではなく、はそれがもたらすコンパイラのをするがあります。

イニシャライザもしてください。

されたの

C99

C99では、されたイニシャライザのがされました。これらをすると、、、またはのどのをのでするかをできます。

のためのされた

プレーンな`int`ようななの

```
int array[] = { [4] = 29, [5] = 31, [17] = 101, [18] = 103, [19] = 107, [20] = 109 };
```

でまれたは、のであり、のどのを=のののでするかをします。されていないはデフォルトでされているため、ゼロがされています。このでは、されたイニシャライザをにしています。らはあるものであるはありません。このはギャップをしています。それらはです。このでは、じにして2つのなるをしていません。 ISO / IEC 98992011、§6.7.9、¶19 は、じサブオブジェクトのにリストされたイニシャライザをオーバーライドするのサブオブジェクトにしてされるリストのでわれる 。

このでは、のサイズがにされていないため、されたイニシャライザでされたインデックスがのサイズをします。このでは21になります。サイズがされている、のわりをえてエントリをすると、どおりエラーになります。

のの

をしてのどのをするかをすることができます。*element*

```
struct Date
{
    int year;
    int month;
    int day;
};

struct Date us_independence_day = { .day = 4, .month = 7, .year = 1776 };
```

がリストにない、それらはデフォルトでゼロされます。

のためのされた

したをして、のどのをするかをできます。

C89

Cは、 `union`をするがありませんでした。 C89 / C90では、 `union`のメンバーをすることができます。そのため、どのメンバーがにリストされているかをできます。

```
struct discriminated_union
{
    enum { DU_INT, DU_DOUBLE } discriminant;
    union
    {
        int     du_int;
        double  du_double;
    } du;
};

struct discriminated_union du1 = { .discriminant = DU_INT, .du = { .du_int = 1 } };
struct discriminated_union du2 = { .discriminant = DU_DOUBLE, .du = { .du_double = 3.14159 }
};
```

C11

C11では、でのメンバをできるので、のでは`du`はありません。

```
struct discriminated_union
{
    enum { DU_INT, DU_DOUBLE } discriminant;
    union
    {
        int     du_int;
        double  du_double;
    };
};

struct discriminated_union du1 = { .discriminant = DU_INT, .du_int = 1 };
struct discriminated_union du2 = { .discriminant = DU_DOUBLE, .du_double = 3.14159 };
```

などのためのされた

これらのコンストラクトは、などのをむのにしてみわせることができます。なセットをすると、そのがににになります。

```
typedef struct Date Date;  // See earlier in this example

struct date_range
{
    Date    dr_from;
    Date    dr_to;
    char    dr_what[80];
};

struct date_range ranges[] =
{
    [3] = { .dr_from = { .year = 1066, .month = 10, .day = 14 },
            .dr_to   = { .year = 1066, .month = 12, .day = 25 },
            .dr_what = "Battle of Hastings to Coronation of William the Conqueror",
          },
    [2] = { .dr_from = { .month = 7, .day =  4, .year = 1776 },
            .dr_to   = { .month = 5, .day = 14, .year = 1787 },
            .dr_what = "US Declaration of Independence to Constitutional Convention",
          }
 };
```

のの

GCCは、じイニシャライザをえるべきののをすることをにするをします

```
int array[] = { [3 ... 7] = 29, 19 = 107 };
```

トリプルドットは、ドットの1つがのとしてされないようにするがあります のマンスルール。

オンラインでをむ https://riptutorial.com/ja/c/topic/4547/

39:

は、すべてののCのISO-IEC 9899-2011でされているです。それらは、の6にされているの3つのの1つですサイドとセマンティクスにって。

ISO-IEC 9899-2011は、をのようにしています。

　　のをするためのまたはな

また、Cにして、「」はのではなく、になるをっていることにもしてください。

いえれば、はなプログラムをにするのをする。このでは、はのとていますが、それにわないプログラムはCでされていません。

、はビヘイビアとにきないがあります。すなわち、がられた、フェーズコンパイルのにメッセージをするがです。このメッセージはであるか、コンパイルをするがあります。

Examples

じスコープのする

Cでされているのとして、スコープ[1]でされたじの2つのがあります。たとえば、のようになります。

```
void foo(int bar)
{
    int var;
    double var;
}
```

このコードはをり、コンパイルにメッセージをするがあります。これは、のビヘイビアとしてにです。なぜなら、プログラムをするににがされ、かをするがあるからです。

したがって、は、このようなコンパイルににになエラーであるがあり、のをもたらすが、コンパイルにすることがまたはなはではない。

1ない

C99

にリンケージがない、6.7.2.3でされたタグをいて、じスコープでじにまたはののがしてはなりません。

+と-はでのみです。したがって、えばでそれらをしようとすると、プログラムはをします

```
struct foo
{
    bool bar;
};

void baz(void)
{
   struct foo testStruct;
   -testStruct; /* This breaks the constraint so must produce a diagnostic */
}
```

オンラインでをむ https://riptutorial.com/ja/c/topic/7397/

40:

Examples

プリインクリメント/ポストインクリメント/デクリメント

Cでは、になのである「++」と「 - 」の2つのがあります。は++ インクリメントととばれている--デクリメントとばれています。どちらもまたは でできます。 ++ののは++operandあり、のは operand++です。でされる、オペランドはに1インクリメントされ、のオペランドのはのにされます。のをえてみましょう。

```
int n, x = 5;
n = ++x; /* x is incremented  by 1(x=6), and result is assigned to n(6) */
         /* this is a short form for two statements: */
         /* x = x + 1; */
         /* n = x ; */
```

でると、のでオペランドののがされ、オペランドのが1インクリメントされます。のをえてみましょう。

```
int n, x = 5;
n = x++; /* value of x(5) is assigned first to n(5), and then x is incremented by 1; x(6) */
         /* this is a short form for two statements: */
         /* n = x; */
         /* x = x + 1; */
```

デクリメントのは、 --にすることができます。

のコードは、それぞれがをしているかをしています

```
    int main()
    {
        int a, b, x = 42;
        a = ++x; /* a and x are 43 */
        b = x++; /* b is 43, x is 44 */
        a = x--; /* a is is 44, x is 43 */
        b = --x; /* b and x are 42 */

        return 0;
    }
```

そのからポストオペレータはののをし、それをするが、めオペレータがをし、されたをすことはらかです。

Cのすべてのバージョンでは、とののはされていないため、のコードはしないをすがあります。

```
    int main()
    {
        int a, x = 42;
```

```
        a = x++ + x; /* wrong */
        a = x + x; /* right */
        ++x;

        int ar[10];
        x = 0;
        ar[x] = x++; /* wrong */
        ar[x++] = x; /* wrong */
        ar[x] = x; /* right */
        ++x;
        return 0;
    }
```

のでできれるは、プレオ―バをすることもいです。のコ―ドをてください。

また、がびされると、のにのすべてのがするがあることにもしてください。

```
    int foo(int x)
    {
        return x;
    }

    int main()
    {
        int a = 42;
        int b = foo(a++);  /* This returns 43, even if it seems like it should return 42 */
        return 0;
    }
```

オンラインでをむ https://riptutorial.com/ja/c/topic/7094/

41: ステートメント/ループfor、while、do-while

- / *すべてのバージョン* /
- for[expression]; [expression]; [expression]one_statement
- for[expression]; [expression]; [expression]{ゼロまたはの}
- 、one_statement
- while{0の}
- whileexpressionwhile one_statementをします。
- whileexpressionwhile {1つの}をします。
- //のフォームにえてC99
- for; []; []one_statement;
- for; []; []{ゼロまたはの}

ステートメント/ループは2つのカテゴリにされます。

- head-controlled iteration/ループ
- フットコントロールステートメント/ループ

ヘッドステートメント/ループ

```
for ([<expression>]; [<expression>]; [<expression>]) <statement>
while (<expression>) <statement>
```

C99

```
for ([declaration expression]; [expression] [; [expression]]) statement
```

フットコントロールのイテレーションステートメント/ループ

```
do <statement> while (<expression>);
```

Examples

ループの

もりしてコードブロックをするには、ループがにります。 `for`ループは、コードブロックがされるときにされます。えば、サイズnをユーザでたすには、 `scanf()`をnするがあります。

C99

```
#include <stddef.h>          // for size_t

int array[10];               // array of 10 int
```

```
for (size_t i = 0; i < 10; i++)    // i starts at 0 and finishes with 9
{
   scanf("%d", &array[i]);
}
```

このように、 scanf()びしはnこのでは10されますが、1だけきまれます。

ここで、iはループインデックスであり、されたものとしてもよくされる。 size_t のは、データオブジェクトをえたりループしたりするすべてのものにするがあります。

forでをするこのは、C99にされたコンパイラでのみできます。なんらかのでいコンパイラにまされているは、 forループのにループインデックスをすることができます

C99

```
#include <stddef.h>         /* for size_t */
size_t i;
int array[10];              /* array of 10 int */

for (i = 0; i < 10; i++)       /* i starts at 0 and finishes at 9 */
{
   scanf("%d", &array[i]);
}
```

whileループ

whileループは、がであるにコードをするためにされます。 whileループは、コードのブロックがされるときにされます。たとえば、されているコードは、ユーザーが0をしているり、ユーザーをします。ユーザーが0した、whileはではなくなり、はループをし、のコードにきます。

```
int num = 1;

while (num != 0)
{
    scanf("%d", &num);
}
```

Do-Whileループ

forループとwhileループとはなり、 do-whileループはループのにのをチェックします。つまり、doブロックはされ、ブロックののwhileのをチェックします。つまり、 do-whileループはになくとも1はされます。

たとえば、このdo-whileループは、これらののが50になるまで、ユーザーからのをします。

```
int num, sum;
num = sum = 0;

do
{
```

```
  scanf("%d", &num);
  sum += num;

} while (sum < 50);
```

do-whileループは、ほとんどのプログラミングスタイルでまれです。

forループにおけるのとれ

```
for ([declaration-or-expression]; [expression2]; [expression3])
{
    /* body of the loop */
}
```

forループでは、ループに3つのがあり、すべてオプションです。

- のdeclaration-or-expressionは、ループをします。ループのに1だけされます。

C99

これは、ループのと、またはなのいずれかです。の、されたのスコープはforステートメントによってされます。

C99

Cのなバージョンではここでのみがされ、ループのはforにくがありfor。

- 2のexpression2がテストです。ににされます。がtrue、コントロールはループのにります。そうでないは、ループのにループののにします。その、bodyとupdateステートメントがされるたびにこのがチェックされます。 true、コントロールはループののにります。このは、ループのがされるをチェックすることをしています。これはループをさせるなです。もう1つはジャンプをするです。
- 3のexpression3は、 ステートメントです。ループのがされるたびにされます。これは、ループがされたをするをインクリメントするためによくされ、このはイテレータとばれます 。

ループのインスタンスはとばれます 。

C99

```
for(int i = 0; i < 10 ; i++)
{
    printf("%d", i);
}
```

はのとおりです。

```
0123456789
```

のでは、にi = 0がされ、 iされます。に、i < 10がチェックされ、これはtrueとされる。コントロ

—ルはループのにり、 `i`のがされます。その、へし`i++`の、 `i`いて0から1に、がチェックされ、がされます。これは、 `i`のが10になるまできます。に、`i < 10`は`false`し、その、コントロールはループからします。

ループ

ループがループとばれるのは、がされてもループのかられることはないということです。これは、ループのテストが`false`されないにし`false`。

C99

```
for (int i = 0; i >= 0; )
{
    /* body of the loop where i is not changed*/
}
```

のでは、イテレータである`i`は0にされてい`true`。テストははは`true`です。しかし、 `i`はのどこでもされず、はです。したがって、 `i`は0のままであり、テストはして`false`にされることはなく、ループにつながります。

ジャンプがないとすると、ループがされるのは、にをにつことです

```
while (true)
{
    /* body of the loop */
}
```

`for`ループでは、はオプションです。この、がに`true`ループにつながる、をなさない。

```
for (;;)
{
    /* body of the loop */
}
```

しかし、によっては、 `break`などのジャンプをしてループをするで、をに`true`につことができます。

```
while (true)
{
    /* statements */
    if (condition)
    {
         /* more statements */
         break;
    }
}
```

ループアンロールとダフのデバイス

には、ストレートフォワードループをループににめることはできません。これは、ループがいく

つかのステートメント***B***によってプライムされるがあるからです。に、りしがいくつかのステートメント***A***でまり、ループのにび***B***がきます。

```
do_B();
while (condition) {
    do_A();
    do_B();
}
```

コードで***B***をりすことでなカット/ペーストのをするには、 ダフのデバイスがのからループをするためにされる`while`して、switchをし、をってします。

```
switch (true) while (condition) {
case false: do_A(); /* FALL THROUGH */
default:    do_B(); /* FALL THROUGH */
}
```

ダフのデバイスは、にはループアンローリングをするためにされました。メモリブロックにマスクをするとしてください`n`はのをつきです。

```
do {
    *ptr++ ^= mask;
} while (--n > 0);
```

`n`がに4でりれるは、にこれをすることができます

```
do {
    *ptr++ ^= mask;
    *ptr++ ^= mask;
    *ptr++ ^= mask;
    *ptr++ ^= mask;
} while ((n -= 4) > 0);
```

しかし、Duffのデバイスでは、 `n`が4でりれない、コードはループのでしいにジャンプするこのアンローリングイディオムにうことができます。

```
switch (n % 4) do {
case 0: *ptr++ ^= mask; /* FALL THROUGH */
case 3: *ptr++ ^= mask; /* FALL THROUGH */
case 2: *ptr++ ^= mask; /* FALL THROUGH */
case 1: *ptr++ ^= mask; /* FALL THROUGH */
} while ((n -= 4) > 0);
```

コンパイラのエンジンはプログラマのためにループをすることができるので、こののアンロールはのコンパイラではほとんどありません。

オンラインでステートメント/ループfor、while、do-whileをむ
https://riptutorial.com/ja/c/topic/5151/ステートメント-ループ-for-while-do-while

42: の

き

のはprintfファミリの `printf` 、 `fprintf`などとのでされ、なるのでをびすことができるため、*varargs*というがされます 。

をしてをするには、 `#include <stdarg.h>`します。

のをるをびすには、スコープにのをむなプロトタイプがあることをして`void err_exit(const char *format, ...);`えば。

- void va_startva_list ap、 *last* ; / *のをします。 *last*は、 "..."ののののパラメータです。* /
- のva_argva_listのAPを、 タイプ ; / *リストののをします。しいプロモートタイプをすようにしてください* /
- void va_endva_list ap; / *の* /
- void va_copyva_list dst、va_list src; / * C99のリスト、つまりのにおけるのをのリストにコピーするえば、をす* /

パラメータ

パラメータ	
`va_list` ap	ポインタ、のリストのの
	ののです。したがって、のをするしいがコンパイラによってされます。 `register` 、、またはとしてすることはできません
タイプ	えばみむためののされた`int`のために`short int`の
`va_list` src	コピーするのポインタ
`va_list` dst	されるしいリスト

`va_start` 、 `va_arg` 、 `va_end` 、および`va_copy`はにはマクロです。

ず `va_start`にびし、だけびすようにしてください。に`va_end`をびし、 `va_end`すべてので1だけびすようにしてください。そうしないと、 あなたのシステムではうまくいくかもしれませんが、かにがないので、バグをきます。

プロトタイプをってをしくし、 のではなく`register`でもでもないのにしてください。のをするにはなくとも1つのがであるため、のみをるをすることはできません。

`va_arg`びすときは、 されたをするがあります。

- `short`するようにされ`int` および`unsigned short`もされ、 `int`がないり`sizeof(unsigned short) == sizeof(int)`それがされるには、 `unsigned int`。
- `float`は`double`します。
- `signed char`は`int`します。 `unsigned char`も`sizeof(unsigned char) == sizeof(int)`でなければ`int`されます。
- `char`は`int`します。
- `uint8_t`や`int16_t`ようなC99もにプロモートされます。

なすなわちKRは`<varargs.h>`されていますが、されたものとしてはしないでください。なここでされ、 `<stdarg.h>`されたものがC89でされました。 `va_copy`マクロはC99でされましたが、そのにくのコンパイラによってされていました。

Examples

なカウントをしてva_listのさをする

のでは、はリストのをっていなければなりません。 `printf()`または`scanf()`では、はにをするかをします。

もなは、のはすべてじのなをすことです。これはのコードのvariadicでされます。このはののをします。のはですが、そのはリストのにとしてされます。

```
#include <stdio.h>
#include <stdarg.h>

/* first arg is the number of following int args to sum. */
int sum(int n, ...) {
    int sum = 0;
    va_list it; /* hold information about the variadic argument list. */

    va_start(it, n); /* start variadic argument processing */
    while (n--)
      sum += va_arg(it, int); /* get and sum the next variadic argument */
    va_end(it); /* end variadic argument processing */

    return sum;
}

int main(void)
{
    printf("%d\n", sum(5, 1, 2, 3, 4, 5)); /* prints 15 */
    printf("%d\n", sum(10, 5, 9, 2, 5, 111, 6666, 42, 1, 43, -6218)); /* prints 666 */
    return 0;
}
```

ターミネーターをしてva_listのわりをする

のでは、はリストのをっていなければなりません。 "の"アプローチ `printf`は、のをにすることです。しかし、これはにいえではありません

```
/* First argument specifies the number of parameters; the remainder are also int */
extern int sum(int n, ...);

/* But it's far from obvious from the code. */
sum(5, 2, 1, 4, 3, 6)

/* What happens if i.e. one argument is removed later on? */
sum(5, 2, 1, 3, 6)  /* Disaster */
```

POSIXの`execlp()`のようななターミネータをするほうがになることがあります。ここでは、のをするためにのの`double`は

```
#include <stdarg.h>
#include <stdio.h>
#include <math.h>

/* Sums args up until the terminator NAN */
double sum (double x, ...) {
  double sum = 0;
  va_list va;

  va_start(va, x);
  for (; !isnan(x); x = va_arg(va, double)) {
    sum += x;
  }
  va_end(va);

  return sum;
}

int main (void) {
  printf("%g\n", sum(5., 2., 1., 4., 3., 6., NAN));
  printf("%g\n", sum(1, 0.5, 0.25, 0.125, 0.0625, 0.03125, NAN));
}
```

なターミネーター

- すべてまたは - `0`または`-1`
- - `NAN`
- ポインタ - `NULL`
- の - らかのな

`printf`のようなインタフェースをってをする

リストのないの1つは、 `printf()`ファミリののりにいラッパーであるをすることです。 1つのそのようなは、エラーのセットである。

errmsg.h

```
#ifndef ERRMSG_H_INCLUDED
#define ERRMSG_H_INCLUDED

#include <stdarg.h>
#include <stdnoreturn.h>    // C11

void verrmsg(int errnum, const char *fmt, va_list ap);
noreturn void errmsg(int exitcode, int errnum, const char *fmt, ...);
void warnmsg(int errnum, const char *fmt, ...);

#endif
```

これはのです。そのようなパッケージはにであるがあります。、プログラマーは`errmsg()`または`warnmsg()`をします。 `verrmsg()`にします。しかし、かがもっとくのことをするがじた、された`verrmsg()`がになります。 あなたがそれをとしているまで、あなたはそれをしないが-あなたはそれをとするつもりはないYAGNIを - YAGNIそれをとするあなたがつもりです が、はにじることになります。

errmsg.c

このコードは、エラーにするために、を`vfprintf()`にするだけです。また、にされたシステム・エラー `errno` にするシステム・エラー・メッセージもされます。

```
#include "errmsg.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void
verrmsg(int errnum, const char *fmt, va_list ap)
{
    if (fmt)
        vfprintf(stderr, fmt, ap);
    if (errnum != 0)
        fprintf(stderr, ": %s", strerror(errnum));
    putc('\n', stderr);
}

void
errmsg(int exitcode, int errnum, const char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    verrmsg(errnum, fmt, ap);
    va_end(ap);
    exit(exitcode);
}

void
warnmsg(int errnum, const char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    verrmsg(errnum, fmt, ap);
    va_end(ap);
}
```

errmsg.h

これらのは、のようにできます。

```
#include "errmsg.h"
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    char buffer[BUFSIZ];
    int fd;
    if (argc != 2)
    {
        fprintf(stderr, "Usage: %s filename\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    const char *filename = argv[1];

    if ((fd = open(filename, O_RDONLY)) == -1)
        errmsg(EXIT_FAILURE, errno, "cannot open %s", filename);
    if (read(fd, buffer, sizeof(buffer)) != sizeof(buffer))
        errmsg(EXIT_FAILURE, errno, "cannot read %zu bytes from %s", sizeof(buffer),
filename);
    if (close(fd) == -1)
        warnmsg(errno, "cannot close %s", filename);
    /* continue the program */
    return 0;
}
```

open()またはread()システムコールのいずれかがすると、エラーがエラーにきまれ、プログラムはコード1でします。close close()システムコールがした、エラーはにメッセージとしてされ、プログラムはされます。

printf()のしいのチェック

GCCGNU Compiler Collectionのであ GNU Cコンパイラをしている、またはClangをしている、エラーメッセージにすが printf()しているものとすることをコンパイラにさせることができます。すべてのコンパイラがをサポートしているわけではないので、きでコンパイルするがあります。しかし、それがえるはのがある。

まず、コンパイラがGCCまたはGCCをエミュレートするClangであることをするをるがあります。えはGCCがそのことをすために__GNUC__をしていることです。

には format については、 のをしてください。

きえられた**errmsg.h**

```
#ifndef ERRMSG_H_INCLUDED
#define ERRMSG_H_INCLUDED
```

```
#include <stdarg.h>
#include <stdnoreturn.h>    // C11

#if !defined(PRINTFLIKE)
#if defined(__GNUC__)
#define PRINTFLIKE(n,m) __attribute__((format(printf,n,m)))
#else
#define PRINTFLIKE(n,m) /* If only */
#endif /* __GNUC__ */
#endif /* PRINTFLIKE */

void verrmsg(int errnum, const char *fmt, va_list ap);
void noreturn errmsg(int exitcode, int errnum, const char *fmt, ...)
        PRINTFLIKE(3, 4);
void warnmsg(int errnum, const char *fmt, ...)
        PRINTFLIKE(2, 3);

#endif
```

さて、もしあなたがのようないをしたら

```
errmsg(EXIT_FAILURE, errno, "Failed to open file '%d' for reading", filename);
```

`%d`は`%s`なければなりません、コンパイラはのようにをいます

```
$ gcc -O3 -g -std=c11 -Wall -Wextra -Werror -Wmissing-prototypes -Wstrict-prototypes \
>     -Wold-style-definition -c erruse.c
erruse.c: In function ‘main’:
erruse.c:20:64: error: format ‘%d’ expects argument of type ‘int’, but argument 4 has type
‘const char *’ [-Werror=format=]
         errmsg(EXIT_FAILURE, errno, "Failed to open file '%d' for reading", filename);
                                                                ~^
                                                                %s
cc1: all warnings being treated as errors
$
```

をする

をすると、ののとにするがなcountまたはのをけるようにされます。

のは、の`printf()`をラップし、 `char` 、 `int` 、 `double`10のをできるようにするをしています。 `printf()`とに、ラッピングののはです。がされるとき、はされるのがあるかどうか、そしてそのはどうあるべきかをすることができます。

```
#include <stdio.h>
#include <stdarg.h>

int simple_printf(const char *format, ...)
{
    va_list ap; /* hold information about the variadic argument list. */
    int printed = 0; /* count of printed characters */

    va_start(ap, format); /* start variadic argument processing */

    while (*format != '\0') /* read format string until string terminator */
```

```
    {
        int f = 0;

        if (*format == '%')
        {
            ++format;
            switch(*format)
            {
                case 'c' :
                    f = printf("%d", va_arg(ap, int)); /* print next variadic argument, note
type promotion from char to int */
                    break;
                case 'd' :
                    f = printf("%d", va_arg(ap, int)); /* print next variadic argument */
                    break;

                case 'f' :
                    f = printf("%f", va_arg(ap, double)); /* print next variadic argument */
                    break;
                default :
                    f = -1; /* invalid format specifier */
                    break;
            }
        }
        else
        {
            f = printf("%c", *format); /* print any other characters */
        }

        if (f < 0) /* check for errors */
        {
            printed = f;
            break;
        }
        else
        {
            printed += f;
        }
        ++format; /* move on to next character in string */
    }

    va_end(ap); /* end variadic argument processing */

    return printed;
}

int main (int argc, char *argv[])
{
    int x = 40;
    int y = 0;

    y = simple_printf("There are %d characters in this sentence", x);
    simple_printf("\n%d were printed\n", y);
}
```

オンラインでのをむ https://riptutorial.com/ja/c/topic/455/の

43:

は、にするのセマンティクスをするキーワードです。それらはシグニチャのなです。のレベルにをぼすまたはサブレベルポインターのみにし、ポイントのにするのにれることがあります。

キーワード	
`const`	されたオブジェクトのにれるをぎます。またはポインターきののをしますポインタのサブタイプのにれます。
`volatile`	プログラムフローのだけでなく、されたオブジェクトレベルまたはポインティングされたポインタサブタイプがのとしてをするがあることをコンパイラにします。
`restrict`	ポインターのみにするヒント。ポインターのに、じポインティングオブジェクトにアクセスするためにのポインターがされないというをします。

クラス `static` 、 `extern` 、 `auto` 、 `register` 、 `signed` 、 `unsigned` 、 `short` 、 `long` と `int` 、 `char` 、 `double`などにするのけはされません。いは、のでそれらをくことです

```
static const volatile unsigned long int a = 5; /* good practice */
unsigned volatile long static int const b = 5; /* bad practice */
```

トップレベル

```
/* "a" cannot be mutated by the program but can change as a result of external conditions */
const volatile int a = 5;

/* the const applies to array elements, i.e. "a[0]" cannot be mutated */
const int arr[] = { 1, 2, 3 };

/* for the lifetime of "ptr", no other pointer could point to the same "int" object */
int *restrict ptr;
```

ポインタのサブタイプの

```
/* "s1" can be mutated, but "*s1" cannot */
const char *s1 = "Hello";

/* neither "s2" (because of top-level const) nor "*s2" can be mutated */
const char *const s2 = "World";

/* "*p" may change its value as a result of external conditions, "**p" and "p" cannot */
```

```
char *volatile *p;

/* "q", "*q" and "**q" may change their values as a result of external conditions */
volatile char *volatile *volatile q;
```

Examples

な

```
const int a = 0; /* This variable is "unmodifiable", the compiler
                    should throw an error when this variable is changed */
int b = 0; /* This variable is modifiable */

b += 10; /* Changes the value of 'b' */
a += 10; /* Throws a compiler error */
```

`const`は、デ―タをするがないことをします。がたちのでわることはできないということではありません。

```
_Bool doIt(double const* a) {
   double rememberA = *a;
   // do something long and complicated that calls other functions

   return rememberA == *a;
}
```

`*a`のびしのに`*a`がされ`*a`がありますので、このは`false`または`true`すことがあり`true`。

`const`つは、ポインタをしてすることもできます。

```
const int a = 0;

int *a_ptr = (int*)&a; /* This conversion must be explicitly done with a cast */
*a_ptr += 10;          /* This has undefined behavior */

printf("a = %d\n", a); /* May print: "a = 10" */
```

しかし、そうすることで、のにつながるエラ―がします。ここでのしさは、なではこれがりにするかもしれないが、コ―ドがきくなるとってしまうことです。

`volatile`キ―ワ―ドは、プログラムフロ―のだけでなく、のがのとしていつでもされるがあることをコンパイラにします。

コンパイラは、volatileとのあるものはしません。

```
volatile int foo; /* Different ways to declare a volatile variable */
int volatile foo;

volatile uint8_t * pReg; /* Pointers to volatile variable */
```

```
uint8_t volatile * pReg;
```

volatileをするなは2つあります。

- メモリマップされたI / Oレジスタをつハードウェアとのインターフェイス。
- プログラムフローのでされたをするえば、りみサービスルーチンで

このをてみましょう

```
int quit = false;

void main()
{
    ...
    while (!quit) {
      // Do something that does not modify the quit variable
    }
    ...
}

void interrupt_handler(void)
{
  quit = true;
}
```

コンパイラは、whileループが`quit`をしてループを`while (true)`ループにしないことにくことができます。 `SIGINT`と`SIGTERM`ハンドラで`quit`がされていても、コンパイラはそれをしません。

`volatile`として`quit`をすると、コンパイラはループをしないようにし、はされます。

こののように、ハードウェアにアクセスするときにもじがします。

```
uint8_t * pReg = (uint8_t *) 0x1717;

// Wait for register to become non-zero
while (*pReg == 0) { } // Do something else
```

オプティマイザのは、がにじであるため、のをみることです。みすはありません。だから々はループにわる。コンパイラになをするために、をのようにします。

```
uint8_t volatile * pReg = (uint8_t volatile *) 0x1717;
```

オンラインでをむ https://riptutorial.com/ja/c/topic/2588/

44: のるい

Cは、、ライブラリによってされる、およびするCプロセッサまかにえばコンパイラとするCプログラムのをする。ビヘイビアにしては、のがプログラムやプロセッサののをしています。、のには、またはなのがあります。そのようなは、そのについてはもることができないため、にけるべきです。そのには、さまざまなのるいがあります。これらのは、Cプロセッサ、ランタイム、およびライブラリして によってなるがありますが、いずれのでもとがあり、するではこれらののでのをします。

プログラムがのるいにすることははにはです。たとえば、プログラムがのオペレーティングにのものであれば、そののプロセッサになインプリメンテーションのにすることはになりにくいでしょう。あるいは、きコンパイルをして、のにしたのるいをすることもできます。どのようなでも、どのにのるいがあるのかをることがです。そのるいをするか、するかどうか、どのようにするかについてにづいたをすことがです。

これらののバランスは、C2011でされているとのリストであり、へのとなります。それらのくはのをしています 。のたちは、ソースコードをプログラムにする8や、ホストされたとしてされているのいなど、のにもっとにっています。にくべきことやすべきものがでされています。されたすべてのがのCでサポートされているわけではありませんが、にえば、それらをサポートするのすべてのバージョンでのがあります。

プログラムとプロセッサ

- **1**バイト 3.6 / 3 のビット 。なくとも$_8$で、のはマクロ`CHAR_BIT`できます。
- どのメッセージが「メッセージ」 3.10 / 1 となされるか

ソース

- ソースファイルのマルチバイトがソースキャラクタセット 5.1.1.2/1 にマッピングされる。
- フェーズ3 5.1.1.2/1 のにのでないのシーケンスがのできえられるかどうかは、
- するがない 5.1.1.2/1 、のリテラルおよびがフェーズ5でされるセット。
- されるメッセージがされる 5.1.1.3/1 。
- フリースタンディング 5.1.2.1/1 でにびされるのと。
- されたセット 5.1.2.1/1 をえてなでなライブラリ。
- フリースタンディング 5.1.2.1/2 におけるプログラムの。
- ホストされたでは、 `int main(int argc, char *arg[])`および`int main(void)` 5.1.2.2.1 / 1 の

`main()`でされたシグニチャ。

- ホストされたが`main()` 5.1.2.2.1 / 2 の2がすをする。
- セクション5.1.2.3 プログラムと7.21.3 ファイル 5.1.2.3/7 ののための「デバイス」をするもの
- 5.1.2.3 / 10 でりみハンドラルーチンがするオブジェクトの。
- フリースタンディングのでは、のスレッドがサポートされているかどうか 5.1.2.4/1 。
- セット 5.2.1 / 1 のメンバーの。
- されたアルファベットのエスケープシーケンス 5.2.2 / 3 にする`char`。
- およびのと 5.2.4.2/1 。
- のと、ライブラリのからへの 5.2.4.2.2 / 6 。
- デフォルトのめモード 5.2.4.2.2 / 8 をエンコードするマクロ`FLT_ROUNDS`の。
- めは、サポートされている`FLT_ROUNDS`が3または-1 5.2.4.2.2 / 8 であることをとします。
- 5.2.4.2.2 / 9 をけるマクロ`FLT_EVAL_METHOD`の。
- サポートされている`FLT_EVAL_METHOD`が-1よりさい 5.2.4.2.2 / 9 ことをとする。
- マクロの`FLT_HAS_SUBNORM`、 `DBL_HAS_SUBNORM`、および`LDBL_HAS_SUBNORM`は、フォーマットが 5.2.4.2.2 / 10 をサポートするかどうかを`LDBL_HAS_SUBNORM`、

タイプ

- オブジェクトがけられているスレッドのスレッドからスレッドをつオブジェクトににアクセスしようとした 6.2.4 / 4
- セットのがりてられている`char`の 6.2.5 / 3 。
- サポートされているきする 6.2.5 / 4 、それらをするためのキーワード。
- **`char`**が**`signed char`**または**`unsigned char`** 6.2.5 / 15 とじとをするかどうか。 `char`がそれぞれなしまたはきのは、 `0`または`SCHAR_MIN`を`CHAR_MIN`せできます。
- オブジェクトのにおけるバイトの、、およびエンコーディング。ただし、でにされているをく 6.2.6.1/2 。
- の**3**つのされたのどれがえられたにされるか、およびオブジェクトののビットパターンがトラップ 6.2.6.2/2 であるかどうか。
- タイプの 6.2.8 / 1 。

- のされたアライメントがサポートされているかどうか 6.2.8 / 3 。
- サポートされているアラインメントのセット 6.2.8 / 4 。
- は、いにされたき 6.3.1.1/1 をにしてランクけされます。
- なし 6.3.1.3/3 にのをりてる 。
- だができないがオブジェクトにりてられている、オブジェクトにされているなは、もい2つのな 6.3.1.4/2 ; 6.3.1.5/1 ; 6.4.4.2 / 3 。
- 0 6.3.2.3/5 のをいて、をポインタにした 。

ソースフォーム

- ヘッダトークンがされる`#pragma`ディレクティブの 6.4 / 4 。
- マルチバイト、アンダースコアの、アクセントのないラテン、ユニバーサルキャラクタ、および 6.4.2.1/1 にれることがあるをむ。
- の 6.4.2.1/5 。
- いくつかのをいて、のソースがセット 6.4.4.4/2 ; 6.4.4.4/10 にマップされる。
- ワイドのをするためにされるのロケール、およびくのそのような 6.4.4.4/11 ののの。
- なるのワイドリテラルトークンをすることができるかどうか、もしそうであれば、としてじるマルチバイトシーケンス 6.4.5 / 5 のは、
- フェーズ7でワイドリテラルをマルチバイトシーケンスにするためにされるロケールと、がセット 6.4.5 / 6 でできないの。
- ヘッダがファイル 6.4.7 / 2 にマッピングされる。
- `FP_CONTRACT`がされていないときにがするかどうか 6.5 / 8
- **`sizeof`**と**`_Alignof`**のの 6.5.3.4/5 。
- ポインタののののサイズ 6.5.6 / 9 。
- きをの 6.5.7 / 5 でシフトした。

の

- `register`キーワードの 6.7.1 / 6 。
- `int`としてされたビットフィールドのが`unsigned int`とじか`signed int` 6.7.2 / 5 であるかどうか。

- のされた`_Bool` 、 `signed int` 、および`unsigned int`の、ビットフィールドのビットフィールドがタイプ 6.7.2.1/5 をつかどうか。
- がどのようにビットフィールドのをレイアウトするかについての 6.7.2.1 / 11 。
- とのビットフィールドメンバの 6.7.2.1/14 。
- 6.7.2.2/4 のとなる。
- `volatile` -qualifed 6.7.3 / 7 のオブジェクトへの「アクセス」をするもの
- `inline`の 6.7.4 / 6 。

プリプロセッサ

- がにされるかどうかは、のとじようにプリプロセッサのとじか、1のがの 6.10.1 / 4 をつかどうかです。
- は、 `#include` 6.10.2 / 2-3 でされたファイルをします。
- ヘッダーがマルチトークン`#include`ディレクティブ 6.10.2 / 4 のトークンからされる。
- `#include`ネスティングの 6.10.2 / 6 。
- かどうか`\`がにされた`\`プリプロセッサのにユニバーサルを`#` 6.10.3.2/2 。
- `STDC` 6.10.6 / 1 のプラグマの`#pragma` preprocessingディレクティブの。
- がないは、 `__DATE__` `__TIME__`マクロと`__TIME__`マクロのがそれぞれです 6.10.8.1/1 。
- マクロ`__STDC_ISO_10646__`がされていないは、 `wchar_t`されるエンコーディング 6.10.8.2/1 。
- マクロ`__STDC_UTF_32__`がされていないに`char32_t`されるエンコーディング 6.10.8.2/1 。

ライブラリ

- アサーションがしたときにされるメッセージの 7.2.1.1/2 。
- でされているものをえるの 7.6 / 6 。
- でされているのめモード 7.6 / 8 。
- によってされたものをえるの 7.6 / 10 。
- アクセススイッチ 7.6.1 / 2 のデフォルト。
- `fegetexceptflag()` 7.6.2.2/1 によってされたステータスフラグの。
- `feraiseexcept()`

が、 "オーバーフロー"または "アンダーフロー" 7.6.2.3/2 をさせるたびに、「な」をさらにさせるかどうか。

ロケールの

- `setlocale()` 7.11.1.1/3 でサポートされる`"C"`のロケール。
- でされるタイプ`float_t`と`double_t` `FLT_EVAL_METHOD`マクロとはなるをする`0` 、 `1` 、び`2` 7.12 / 2 。
- でされているをえるの 7.12 / 6 。
- `math.h`がすは、ドメインエラー 7.12.1 / 2 のにします。
- エラー 7.12.1 / 3 のイベントで`math.h`がす。
- がアンダーフローしたときの`math.h`のり、 `errno`が`ERANGE`さ`ERANGE`ているかどうか、およびこれらの 7.12.1 / 6 でがするかどうかの。
- FPスイッチのデフォルト 7.12.2 / 2 。
- `fmod()`が0をすか、2のが0 7.12.10.1/3 のときにドメインエラーをさせるかどうか。
- `remainder()`が0をすか、2のが0 7.12.10.2/3 のときにドメインエラーをさせるかどうか。
- `remquo()` 7.12.10.3 / 2 によってされたモジュラスのビット。
- `remquo()`が0をすか、2のが0 7.12.10.3/3 のときにドメインエラーをさせるかどうか。
- サポートされているシグナルのなセット、そのセマンティクス、およびデフォルト 7.14 / 4 。
- シグナルがされ、そのシグナルにけられたカスタムハンドラがする、ハンドラ 7.14.1.1/3 のにシグナルがブロックされているは、それがブロックされます。
- `SIGFPE` 、 `SIGILL` 、および`SIGSEGV`の`SIGFPE`は、カスタムシグナルハンドラからったときのをにします 7.14.1.1/3 。
- にされるようにされているデフォルトになく、 7.14.1.1 / 6 。
- マクロ`NULL`するのNULLポインター 7.19 / 3 。

ファイル

- テキストストリームののに 7.21.2 / 2 がかどうか。
- バイナリストリーム 7.21.2 / 3 ににされるヌルの。
-

モード 7.21.3 / 1 でいたファイルの。

- テキストストリームへのきみによってストリームがりてられるかどうか 7.21.3 / 2
- ストリームバッファリング 7.21.3 / 3 のサポート
- さがゼロのファイルがにするかどうか 7.21.3 / 4 。
- なファイルをするための 7.21.3 / 8 。
- じファイルをにくことができるかどうか 7.21.3 / 8 。
- マルチバイト 7.21.3 / 10 のエンコーディングのと。
- ターゲットファイルがいているときの`remove()`の 7.21.4.1/2 。
- ターゲットファイルがすでにするの`rename()`の 7.21.4.2/2 。
- プログラムがしたに、 `tmpfile()`によってされた`tmpfile()`がされるかどうか 7.21.4.3/2 。
- どので`freopen()` 7.21.5.4/3 によってどのモードができるのか

I / O

- またはのFPののどれがprintf - ファミリ 7.21.6.1/8 によってされるか。
- ポインターが`printf()` family 7.21.6.1/8 によってフォーマットされる。
- `-`が`[`フィールド 7.21.6.2/12 のスキャンリストのにされたときの`scanf()` `-` familyの。
- `scanf()` familyのほとんどのは`p`フィールドをします 7.21.6.2/12 。
- に`fgetpos()`によってされた`errno` 7.21.9.1/2 。
- に`fsetpos()`によってされた`errno` 7.21.9.3/2 。
- に`ftell()`によってされた`errno` 7.21.9.4/3 。
- `strtod()`のは、NaNフォーマット 7.22.1.3p4 のいくつかのサポートされているのです。
- がアンダーフローしたときに`strtod()` familyが`errno`を`ERANGE`に`errno`するかどうか 7.22.1.3 / 10 。

メモリりて

- されたバイトが0 7.22.3 / 1 ののメモリりての。

システム

- `abort()`がびされたときにクリーンアップがあればされ、どのがホストOSにされるか 7.22.4.1/2 。
- `exit()`がびされると、どのようなステータスがホストにされますか 7.22.4.4/5 。
- `_Exit()`がびされたときにオープンストリームのとホストにされるステータス 7.22.4.5/2 。
- `getenv()`とをするためのメソッド 7.22.4.6/2 でアクセスなの。
- `system()`のり 7.22.4.8/3 。
- と 7.27.1 / 1 。
- `clock_t`と`time_t` 7.27.1 / 4 でなのと。
- `clock()` 7.27.2.1/3 によってされたのとしてするのまり。
- `timespec_get()`によってされたのとなるエポックのまりタイムベースが`TIME_UTC`、 `TIME_UTC` / 3 。
- "C"ロケール 7.27.3.5/7 の`%Z`にする`strftime()`きえ。

ワイド

- またはのFPののどれが`wprintf()` family 7.29.2.1/8 によってされるか。
- ポインターが`wprintf()` family 7.29.2.1/8 によってフォーマットされる。
- `wscanf()` - familyのは、 `-`が`[`フィールド 7.29.2.2/12 のスキャンリストのにされるときにします。
- `wscanf()` familyのほとんどのは、 `p`フィールドをします 7.29.2.2/12 。
- NaNフォーマット 7.29.4.1.1 / 4 のいくつかのサポートされているの`wstrtod()`ファミリにする。
- がアンダーフローしたときに`wstrtod()` familyが`errno`を`ERANGE`に`errno`するかどうか 7.29.4.1.1 / 10 。

Examples

ののシフト

```
int signed_integer = -1;

// The right shift operation exhibits implementation-defined behavior:
int result = signed_integer >> 1;
```

のをにする

```
// Supposing SCHAR_MAX, the maximum value that can be represented by a signed char, is
// 127, the behavior of this assignment is implementation-defined:
signed char integer;
integer = 128;
```

0バイトのりて

```
// The allocation functions have implementation-defined behavior when the requested size
// of the allocation is zero.
void *p = malloc(0);
```

きの

きは、3つののいずれかでできます。どちらがされるかはされています。えられたきの`int`のは、にそのの`-1`のの2ビットからのようにすることができます。

```
enum { sign_magnitude = 1, ones_compl = 2, twos_compl = 3, };
#define SIGN_REP(T) ((T)-1 & (T)3)

switch (SIGN_REP(long)) {
   case sign_magnitude: { /* do something */ break; }
   case ones_compl:     { /* do otherwise */ break; }
   case twos_compl:     { /* do yet else  */ break; }
   case 0:  { _Static_assert(SIGN_REP(long), "bogus sign representation"); }
}
```

よりいのにもじパターンがてはまりますが、`&`のオペランドはがされるに「の」のとなるため、このではテストできません。

オンラインでのるいをむ https://riptutorial.com/ja/c/topic/4832/のるい

45:

オブジェクトまたはファンクションをするのは、にオブジェクトまたはファンクションのともばれます。

Examples

のCファイルからをびす

foo.h

```
#ifndef FOO_DOT_H    /* This is an "include guard" */
#define FOO_DOT_H    /* prevents the file from being included twice. */
                     /* Including a header file twice causes all kinds */
                     /* of interesting problems.*/

/**
 * This is a function declaration.
 * It tells the compiler that the function exists somewhere.
 */
void foo(int id, char *name);

#endif /* FOO_DOT_H */
```

foo.c

```
#include "foo.h"    /* Always include the header file that declares something
                     * in the C file that defines it. This makes sure that the
                     * declaration and definition are always in-sync.  Put this
                     * header first in foo.c to ensure the header is self-contained.
                     */
#include <stdio.h>

/**
 * This is the function definition.
 * It is the actual body of the function which was declared elsewhere.
 */
void foo(int id, char *name)
{
    fprintf(stderr, "foo(%d, \"%s\");\n", id, name);
    /* This will print how foo was called to stderr - standard error.
     * e.g., foo(42, "Hi!") will print `foo(42, "Hi!")`
     */
}
```

main.c

```
#include "foo.h"

int main(void)
{
    foo(42, "bar");
```

```
    return 0;
}
```

コンパイルとリンク

に、 `foo.c`と`main.c`をオブジェクトファイルに コンパイルします 。ここでは`gcc`コンパイラをしますが、コンパイラのがなり、のオプションがです。

```
$ gcc -Wall -c foo.c
$ gcc -Wall -c main.c
```

ここでそれらをリンクしてなファイルをします

```
$ gcc -o testprogram foo.o main.o
```

グロ―バルの

グロ―バルのはにおめできません。これにより、プログラムのがしくなり、デバッグがしくなります。しかし、にはグロ―バルをすることもされます。

global.h

```
#ifndef GLOBAL_DOT_H    /* This is an "include guard" */
#define GLOBAL_DOT_H

/**
 * This tells the compiler that g_myglobal exists somewhere.
 * Without "extern", this would create a new variable named
 * g_myglobal in _every file_ that included it. Don't miss this!
 */
extern int g_myglobal; /* _Declare_ g_myglobal, that is promise it will be _defined_ by
                        * some module. */

#endif /* GLOBAL_DOT_H */
```

global.c

```
#include "global.h" /* Always include the header file that declares something
                     * in the C file that defines it. This makes sure that the
                     * declaration and definition are always in-sync.
                     */

int g_myglobal;     /* _Define_ my_global. As living in global scope it gets initialised to 0
                     * on program start-up. */
```

main.c

```
#include "global.h"

int main(void)
{
    g_myglobal = 42;
```

```
    return 0;
}
```

externをしてソースファイルでをするにはどうすればよいですかをしてください。

グローバルの

ヘッダーは、えばテーブルのように、グローバルにされるみりリソースをするためにできます。

それらをしたいファイル「 」にまれるのヘッダーにそれらをしてください。じヘッダーをして、するをしてすべてのリソースをするとです

resources.h

```
#ifndef RESOURCES_H
#define RESOURCES_H

typedef enum { /* Define a type describing the possible valid resource IDs. */
  RESOURCE_UNDEFINED = -1, /* To be used to initialise any EnumResourceID typed variable to be

                              marked as "not in use", "not in list", "undefined", wtf.
                              Will say un-initialised on application level, not on language
level. Initialised uninitialised, so to say ;-)
                              Its like NULL for pointers ;-)*/
  RESOURCE_UNKNOWN = 0,    /* To be used if the application uses some resource ID,
                              for which we do not have a table entry defined, a fall back in
                              case we _need_ to display something, but do not find anything
                              appropriate. */

  /* The following identify the resources we have defined: */
  RESOURCE_OK,
  RESOURCE_CANCEL,
  RESOURCE_ABORT,
  /* Insert more here. */

  RESOURCE_MAX /* The maximum number of resources defined. */
} EnumResourceID;


extern const char * const resources[RESOURCE_MAX]; /* Declare, promise to anybody who includes

                                      this, that at linkage-time this symbol will be around.
                                      The 1st const guarantees the strings will not change,
                                      the 2nd const guarantees the string-table entries
                                      will never suddenly point somewhere else as set during
                                      initialisation. */
#endif
```

にする.cファイルをしたリソースをするには、するヘッダー.hファイルでされていたもののインスタンスをするのがあります。

resources.c

```
#include "resources.h" /* To make sure clashes between declaration and definition are
                          recognised by the compiler include the declaring header into
```

```
                          the implementing, defining translation unit (.c file).

/* Define the resources. Keep the promise made in resources.h. */
const char * const resources[RESOURCE_MAX] = {
  "<unknown>",
  "OK",
  "Cancel",
  "Abort"
};
```

これをするプログラムはのようになります。

main.c

```
#include <stdlib.h> /* for EXIT_SUCCESS */
#include <stdio.h>

#include "resources.h"


int main(void)
{
  EnumResourceID resource_id = RESOURCE_UNDEFINED;

  while ((++resource_id) < RESOURCE_MAX)
  {
    printf("resource ID: %d, resource: '%s'\n", resource_id, resources[resource_id]);
  }

  return EXIT_SUCCESS;
}
```

の3つのファイルをGCCをってコンパイルし、それらをリンクして`main`のプログラムファイルにします。

```
gcc -Wall -Wextra -pedantic -Wconversion -g  main.c resources.c -o main
```

これらの`-Wall -Wextra -pedantic -Wconversion`すると、コンパイラがに`-Wall -Wextra -pedantic -Wconversion`になるので、コーードをするにかをしてはいけません。、あるいはプロダクションにするがあります

したプログラムをする

```
$ ./main
```

そして、

```
resource ID: 0, resource: '<unknown>'
resource ID: 1, resource: 'OK'
resource ID: 2, resource: 'Cancel'
resource ID: 3, resource: 'Abort'
```

き

のはのとおりです。

```
int a; /* declaring single identifier of type int */
```

のでは、 intのオブジェクトをするaというのがされaいます。

```
int a1, b1; /* declaring 2 identifiers of type int */
```

2のでは、 a1とb1というの2つのがされていますが、じintでものオブジェクトをしています。

には、このようにします。まず、あるをれて、コンマ , でられた1つまたはのをきます このではされません。このコンテキスト 。そのようなをくには、 * 、びし ( ) またはまたはインデックス - [ ] のみをいくつかのにすることができますはまったくできません。されるは、のスコープでするはありません。いくつかの

```
/* 1 */ int /* 2 */ (*z) /* 3 */ , /* 4 */ *x , /* 5 */ **c /* 6 */ ;
```

1	の。
2	あるzへのをするされない。
3	じのでもう1つのがくことをすカンマがあります。
4	のxをするされていない。
5	のにをして、されていない(*c) 。
6	のわり。

ののどれもがこののにえなかったので、されるはそのにではないでしょう。

そのようなのに、それでされるがのスコープにされます。 がそれにリンケージをりてているは、のがじオブジェクトまたはをするようにじタイプのリンケージですることもできます

さらに、イコール = をにすることもできます。ののに=がのにく、されるもされているといます。=のにもうをれることができますが、はされ、そのはされたオブジェクトのとしてされます。

```
int l = 90; /* the same as: */

int l; l = 90; /* if it the declaration of l was in block scope */

int c = 2, b[c]; /* ok, equivalent to: */

int c = 2; int b[c];
```

コードのでは、しくされたのからまったくじをくことができ、すべてのアクセスにながりてられているとして、のにされたのオブジェクトをえますのオブジェクト。

```
void f()
{
    int b2; /* you should be able to write later in your code b2
            which will directly refer to the integer object
            that b2 identifies */

    b2 = 2; /* assign a value to b2 */

    printf("%d", b2); /*ok - should print 2*/

    int *b3; /* you should be able to write later in your code *b3 */

    b3 = &b2; /* assign valid pointer value to b3 */

    printf("%d", *b3); /* ok - should print 2 */

    int **b4; /* you should be able to write later in your code **b4 */

    b4 = &b3;

    printf("%d", **b4); /* ok - should print 2 */

    void (*p)(); /* you should be able to write later in your code (*p)() */

    p = &f; /* assign a valid pointer value */

    (*p)(); /* ok - calls function f by retrieving the
            pointer value inside p -    p
            and dereferencing it -      *p
            resulting in a function
            which is then called -      (*p)() -

            it is not *p() because else first the () operator is
            applied to p and then the resulting void object is
            dereferenced which is not what we want here */
}
```

`b3`のは、オブジェクトにアクセスするためのとして`b3`をするがあることをしています。

もちろん、 `b3 *` をするには、なもしておくがありますはポインタを。また、オブジェクトをするに、まずオブジェクトにをするがありますこのについては、 ここでしくることができます 。のではすべてこれをっています。

```
int a3(); /* you should be able to call a3 */
```

これはコンパイラに`a3`をびそうとしていることをえます。この、 `a3`はオブジェクトではなくをします。オブジェクトとの1つのいは、にはにらかのがあることです。

```
void f1()
{
    {
        int f2(); /* 1 refers to some function f2 */
```

```
    }

    {
        int f2(); /* refers to the exact same function f2 as (1) */
    }
}
```

のでは、2つのはじ`f2`していますが、2つのなるブロックスコープをつこのコンテキストでオブジェクトをしていた、2つのなるオブジェクトになります。

```
int (*a3)(); /* you should be able to apply indirection to `a3` and then call it */
```

はになっているようですが、のをっていれば、のをむのにはありません。 `*`は`( )`よりがいため、かっこがです。

えをする、としてられるはにはのでではありません。なぜなら、そのでされるインデックス`[`および`]`のはにこのオブジェクト/のよりも1です。

```
int a4[5]; /* here a4 shouldn't be accessed using the index 5 later on */
```

しかし、5よりさいのすべてのインデックスでアクセスでなければなりません。

```
a4[0], a4[1]; a4[4];
```

`a4[5]`はUBとなる。のについては、 ここをしてください 。

```
int (*a5)[5](); /* here a4 could be applied indirection
                   indexed up to (but not including) 5
                   and called */
```

なことに、たちの、にはですが、 `a5`のはのではされています。

Typedef

`typedef`は`typedef`というキーワードをに、のにあるです。えば

```
typedef int (*(*t0)())[5];
```

*`int typedef (*(*t0)())[5];`*ようにのにくこともでき*`int typedef (*(*t0)())[5];`*が、これはおめしません

のは、typedefのをします。でこのようにうことができます

```
t0 pf;
```

いているのとじがあります

```
int (*(*pf)())[5];
```

てかるように、typedefは、でのにするとしてを「」します。このので、キーストロークをできます。また、 `typedef`をったはまだされているので、のだけではされません

```
t0 (*pf1);
```

じです

```
int (*(**pf1)())[5];
```

またはのルールをしてCのをする

「」ルールは、Cをするためのなです。また、にもちます。

でしたシンボルをむ...

```
*   as "pointer to"          - always on the left side
[]  as "array of"            - always on the right side
()  as "function returning"  - always on the right side
```

ルールの

ステップ**1**

をします。これがあなたのです。それからあなたにう。「は」。あなたはをめました。

ステップ**2**

のにあるシンボルをてください。 、たとえば、あなたがつけた`()`そこに、あなたは、こののであることをっています。だからあなたは"がをす"ことがあります。または、 `[]`がつかったは、 "はです"といます。がりなくなるか、かっこをすまでにみます`)`。 をつと`(`のにめがあっても、それは`()`のまりです。

ステップ**3**

のにあるをてください。のシンボルのいずれかでないたとえば、「int」のようなもの、にそれをいます。それのは、のをしてにしてください。がりなくなるか、`(`。

ここでをするまで2と3をりします。

ここではいくつかのをします。

```
int *p[];
```

まず、をつける

```
int *p[];
     ^
```

*"p*は"

は、やがれるまでにしてください。

```
int *p[];
      ^^
```

*"p*は*"*

もはやにくことができませんののので、にしてつけてください

```
int *p[];
    ^
```

*"p*は"

にくとつける

```
int *p[];
^^^
```

*"p*は*int*へのポインタのです"。

または*"p*はが*int*へのポインタのです"

もうつの

```
int *(*func())();
```

をします。

```
int *(*func())();
       ^^^^
```

*"func*は"

にく。

```
int *(*func())();
           ^^
```

*"func*はをす"

のためににできないので、にします。

```
int *(*func())();
      ^
```

*"func*は、ポインタを"

のためににすることはできませんので、にんでください。

```
int *(*func())();
             ^^
```

*"func*はへのポインタをすです"

たちはシンボルかられているのでもうできませんので、にってください。

```
int *(*func())();
    ^
```

*"func*はポインタをすへのポインタをすです"

そしてに、にってください。にはもっていないからです。

```
int *(*func())();
^^^
```

*"func*は*int*へのポインタをすへのポインタをすです"。

ごのとおり、このルールはにです。また、をしているにをにチェックし、のシンボルのとかっこがかどうかについてのヒントをえることもできます。

いくつかのは、プロトタイプのサイズとリストのために、よりにえます。 `[3]`がされているは、*"arraysize 3of ..."*とみえられます 。 `(char *,int)`が "function expectingchar 、int*and returning ..."* としてみまれているのをた。

ここはしいものです

```
int (*(*fun_one)(char *,double))[9][20];
```

はこのステップをするためのステップをずにめません。

* "fun_oneは、 char 、doubleをし、intの*size 20*の*size 9*へのポインタをすへのポインタです。"

ごのように、のサイズとのリストをりくと、それほどではありません

```
int (*(*fun_one)())[][];
```

そのようにし、でのサイズとのリストをれることができます。

いくつかのな

このルーをってなをすることはかなりです。したがって、Cのでなものについてのがです。たとえば、の

```
int *((*fun_one)())[][];
```

*"fun_one*は、*int*へのポインタののをすへのポインタです"とみまれます 。はをすことはできず、へのポインタだけをすので、そのはです。

なみわせにはのものがあります

```
[]() - cannot have an array of functions
()() - cannot have a function that returns a function
()[] - cannot have a function that returns an array
```

のすべてのケースでは、があるためには、これらの`()`と`[]`のののの`*`をバインドするためのがです。

ここにいくつかのがあります

```
int i;              an int
int *p;             an int pointer (ptr to an int)
int a[];            an array of ints
int f();            a function returning an int
int **pp;           a pointer to an int pointer (ptr to a ptr to an int)
int (*pa)[];        a pointer to an array of ints
int (*pf)();        a pointer to a function returning an int
int *ap[];          an array of int pointers (array of ptrs to ints)
int aa[][];         an array of arrays of ints
int *fp();          a function returning an int pointer
int ***ppp;         a pointer to a pointer to an int pointer
int (**ppa)[];      a pointer to a pointer to an array of ints
int (**ppf)();      a pointer to a pointer to a function returning an int
int *(*pap)[];      a pointer to an array of int pointers
int (*paa)[][];     a pointer to an array of arrays of ints
int *(*pfp)();      a pointer to a function returning an int pointer
int **app[];        an array of pointers to int pointers
int (*apa[])[];     an array of pointers to arrays of ints
int (*apf[])();     an array of pointers to functions returning an int
int *aap[][];       an array of arrays of int pointers
int aaa[][][];      an array of arrays of arrays of int
int **fpp();        a function returning a pointer to an int pointer
int (*fpa())[];     a function returning a pointer to an array of ints
int (*fpf())();     a function returning a pointer to a function returning an int
```

```
int af[]();         an array of functions returning an int
int fa()[];         a function returning an array of ints
int ff()();         a function returning a function returning an int
int (*pfa)()[];     a pointer to a function returning an array of ints
int aaf[][]();      an array of arrays of functions returning an int
int (*paf)[]();     a pointer to a an array of functions returning an int
int (*pff)()();     a pointer to a function returning a function returning an int
int *afp[]();       an array of functions returning int pointers
int afa[]()[];      an array of functions returning an array of ints
```

```
int aff[]();        an array of functions returning functions returning an int
int *fap()[];       a function returning an array of int pointers
int faa()[][];      a function returning an array of arrays of ints
int faf()[]();      a function returning an array of functions returning an int
int *ffp()();       a function returning a function returning an int pointer
```

http : //ieng9.ucsd.edu/~cs30x/rt_lt.rule.html

オンラインでをむ https://riptutorial.com/ja/c/topic/3729/

46: と

とのいはですか

ソースくいシンボルの https : //www.amazon.com/Computer-Systems-Programmers-Perspective-2nd/dp/0136108040/

Examples

との

はをし、、オブジェクト、などのをします。は、コンパイラがそのへのをけれるがあるものです。これらはです

```
extern int bar;
extern int g(int, int);
double f(int, double); /* extern can be omitted for function declarations */
double h1();           /* declaration without prototype */
double h2();           /* ditto                         */
```

はにこのをインスタンス/します。これらのエンティティへのをリンクするためにリンカがとするものです。これらはのにするです。

```
int bar;
int g(int lhs, int rhs) {return lhs*rhs;}
double f(int i, double d) {return i+d;}
double h1(int a, int b) {return -1.5;}
double h2() {}  /* prototype is implied in definition, same as double h2(void) */
```

のわりにをすることができます。

ただし、これはに1するがあります。され、どこかでされているものをすることをれた、リンカ―はをするかをらず、シンボルのについてをいます。もかをすると、リンカ―はをしてしているシンボルにリンクするかどうかをりません。

```
extern int i = 0;  /* defines i */
extern int j;  /* declares j */
```

このは、「いシンボルといシンボル」リンカのからというをしてできます。しいはこちら スライド22をごください。

```
/* All are definitions. */
struct S { int a; int b; };            /* defines S */
struct X {                             /* defines X */
    int x;                             /* defines non-static data member x */
};
struct X anX;                                 /* defines anX */
```

オンラインでとをむ https://riptutorial.com/ja/c/topic/3104/と

47: 、、のリテラル

リテラルというは、えば0 やえば"c" などのをするCコードのをするためによくされます。にえば、では、リテラルの「リテラル」をし、、、とのをしますが、これはないではありません。

リテラルは、デフォルトのまたはそのをするリテラルをまたはできるなであるまたは ではないをつことができます。

Examples

リテラル

リテラルはをするためにされます。でされる3つのベースがサポートされています。

ベース		
	し	5
オクタル	0	0345
16	0xまたは0X	0x12AB 、 0X12AB 、 0x12ab 、 0x12Ab

このにはがまれていないので、リテラルはにであることにしてください。 -1ようなものは、1つのリテラル 1 をつとしてわれ、a -

10リテラルのは、 intおよびlongにするのデータです。 C99、にいリテラルでもlong longがサポートされています。

8または16リテラルのは、 int 、 unsigned 、 longおよびunsigned longのにするのデータです。C99、にきなリテラルではlong longとunsigned long longもサポートされています。

さまざまなをして、リテラルのデフォルトのをすることができます。

サフィックス	
L 、 l	long int
LL 、 ll C99	long long int
U 、 u	unsigned

UとL / LLは、のでみわせることができます。なるケースがあっても、をにすることはりですえば、2つのUをけるなど。

リテラル

のリテラルは、のをするためにされます。それらはでまれたのびですえば、 `"abcd"` 、 `char*`。

`L`は、リテラルを`wchar_t*`のワイドにします。たとえば、 `L"abcd"`ます。

C11、 `L`にたのがあります。

プレフィックス	ベースタイプ	エンコーディング
し	`char`	プラットフォームにする
`L`	`wchar_t`	プラットフォームにする
`u8`	`char`	UTF-8
`u`	`char16_t`	はUTF-16
`U`	`char32_t`	はUTF-32

の2つについては、エンコーディングがにするUTFエンコーディングである、テストマクロをしてクエリをできます。

リテラル

リテラルは、きをすためにされます。リテラルのをするには、のをできます。

サフィックス	タイプ	
し	`double`	`3.1415926 -3E6`
`f` 、 `F`	`float`	`3.1415926f 2.1E-6F`
`l` 、 `L`	`long double`	`3.1415926L 1E126L`

これらのをするには、リテラルはリテラルでなければなりません。えば、 `3f`ので、エラーである`3`ながら、リテラルである`3.f`は`3.0f`しいです。 `long double`、のためににの`L`をすることをおめします。

リテラル

リテラルは、1をすためにされるなのリテラルです。それらはでまれています `'a'`。 `int`です。リテラルのは、マシンのセットにじたです。らはをさない。

リテラルのにある`L`は、 `wchar_t`のワイドにします。に、C11の`u`と`U`は`char16_t`と`char32_t`のワイドになります。

のえば、をしようとする、エスケープシーケンスがされます。エスケープシーケンスは、のにされたのをします。すべてのエスケープシーケンスは2つのでされ、はバックスラッシュ`\`です。バックスラッシュのにあるは、シーケンスがどのリテラルとしてされるかをします。

エスケープシーケンス	された
`\b`	バックスペース
`\f`	フォームフィード
`\n`	
`\r`	キャリッジリターン
`\t`	タブ
`\v`	タブ
`\\`	バックスラッシュ
`\'`	
`\"`	
`\?`	
`\nnn`	8
`\xnn` ...	16

C89

エスケープシーケンス	された
`\a`	アラートビープ、ベル

C99

エスケープシーケンス	された
`\unnnn`	ユニバーサルキャラクターネーム
`\Unnnnnnnn`	ユニバーサルキャラクターネーム

ユニバーサルは、Unicodeコードポイントです。なのは、のにマップすることができます。`n`は16としてされます。のUTFエンコーディングにじて、ユニバーサルのシーケンスは、のの`char`ではなく、のでされるコードポイントになります。

エスケープシーケンスをテキストモードですると、OSのバイトまたはバイトシーケンスにされます。

のエスケープシーケンスは、 をけるためにされます 。たとえば、 `??/`はバックスラッシュ`'\'`すとしてコンパイルされますが、 `?\?/`をすると `"??/"`ます。

8のエスケープシーケンスには、1つ、2つまたは3つの8_nがあります。

オンラインで、、のリテラルをむ https://riptutorial.com/ja/c/topic/3455/--のリテラル

48:

き

Cでは、はみみではありません。 Cは、 `'\0'`によってヌルでわるの1をつです。

これは、が`"abc"` Cが4の`'a'` 、 `'b'` 、 `'c'` 、 `'\0'`をつことをします。

のなのをしてください。

- char str1 [] = "こんにちは、"; / ** /
- char str2 [14] = "こんにちは、"; / ** /
- char * str3 = "こんにちは、"; / ** /

Examples

さをするstrlen

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
    /* Exit if no second argument is found. */
    if (argc != 2)
    {
        puts("Argument missing.");
        return EXIT_FAILURE;
    }

    size_t len = strlen(argv[1]);
    printf("The length of the second argument is %zu.\n", len);

    return EXIT_SUCCESS;
}
```

このプログラムは、2ののさをし、を`len`します。に、そのさをにします。たとえば、パラメータ`program_name "Hello, world!"`してすると、プログラムはします`The length of the second argument is 13.`なぜなら、`Hello, world!` 13です。

`strlen`は、のからNUL`'\0'`までのすべてのバイトをカウントします。したがって、がNULでることがされているにのみできます。

また、にUnicodeがまれている、 `strlen`はにいくつのがあるかをえませんいくつかのはバイトのさかもしれないので。このようなには、 すなわち 、コードをでカウントするがあります。ののをえてみましょう。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    char asciiString[50] = "Hello world!";
    char utf8String[50] = "Γειά σου Κόσμε!"; /* "Hello World!" in Greek */

    printf("asciiString has %zu bytes in the array\n", sizeof(asciiString));
    printf("utf8String has %zu bytes in the array\n", sizeof(utf8String));
    printf("\"%s\" is %zu bytes\n", asciiString, strlen(asciiString));
    printf("\"%s\" is %zu bytes\n", utf8String, strlen(utf8String));
}
```

```
asciiString has 50 bytes in the array
utf8String has 50 bytes in the array
"Hello world!" is 12 bytes
"Γειά σου Κόσμε!" is 27 bytes
```

コピーと**strcpy**、strcat

```
#include <stdio.h>
#include <string.h>

int main(void)
{
  /* Always ensure that your string is large enough to contain the characters
   * and a terminating NUL character ('\0')!
   */
  char mystring[10];

  /* Copy "foo" into `mystring`, until a NUL character is encountered. */
  strcpy(mystring, "foo");
  printf("%s\n", mystring);

  /* At this point, we used 4 chars of `mystring`, the 3 characters of "foo",
   * and the NUL terminating byte.
   */

  /* Append "bar" to `mystring`. */
  strcat(mystring, "bar");
  printf("%s\n", mystring);

  /* We now use 7 characters of `mystring`: "foo" requires 3, "bar" requires 3
   * and there is a terminating NUL character ('\0') at the end.
   */

  /* Copy "bar" into `mystring`, overwriting the former contents. */
  strcpy(mystring, "bar");
  printf("%s\n", mystring);

  return 0;
}
```

```
foo
foobar
```

```
bar
```

のにまたはのからコピーするは、NULですることをしてください。

リテラル `"foo"` は、コンパイラによってNULされます。

strcmp、strncmp、strcasecmp、strncasecmp

`strcase*`は、Cではなく、POSIXです。

`strcmp`は、2つのNULLをにします。は、のがで2のにれるはのをし、しいは0を、ののでのがれたはをします。

```
#include <stdio.h>
#include <string.h>

void compare(char const *lhs, char const *rhs)
{
    int result = strcmp(lhs, rhs); // compute comparison once
    if (result < 0) {
        printf("%s comes before %s\n", lhs, rhs);
    } else if (result == 0) {
        printf("%s equals %s\n", lhs, rhs);
    } else { // last case: result > 0
        printf("%s comes after %s\n", lhs, rhs);
    }
}

int main(void)
{
    compare("BBB", "BBB");
    compare("BBB", "CCCCC");
    compare("BBB", "AAAAAA");
    return 0;
}
```

```
BBB equals BBB
BBB comes before CCCCC
BBB comes after AAAAAA
```

`strcmp`として、 `strcasecmp`は、をのにしたで、にそのをします。

```
#include <stdio.h>
#include <string.h>

void compare(char const *lhs, char const *rhs)
{
    int result = strcasecmp(lhs, rhs); // compute case-insensitive comparison once
    if (result < 0) {
        printf("%s comes before %s\n", lhs, rhs);
    } else if (result == 0) {
        printf("%s equals %s\n", lhs, rhs);
    } else { // last case: result > 0
        printf("%s comes after %s\n", lhs, rhs);
    }
```

```
}

int main(void)
{
    compare("BBB", "bBB");
    compare("BBB", "ccCCC");
    compare("BBB", "aaaaaa");
    return 0;
}
```

```
BBB equals bBB
BBB comes before ccCCC
BBB comes after aaaaaa
```

`strncmp`と`strncasecmp`はでnをします

```
#include <stdio.h>
#include <string.h>

void compare(char const *lhs, char const *rhs, int n)
{
    int result = strncmp(lhs, rhs, n); // compute comparison once
    if (result < 0) {
        printf("%s comes before %s\n", lhs, rhs);
    } else if (result == 0) {
        printf("%s equals %s\n", lhs, rhs);
    } else { // last case: result > 0
        printf("%s comes after %s\n", lhs, rhs);
    }
}

int main(void)
{
    compare("BBB", "Bb", 1);
    compare("BBB", "Bb", 2);
    compare("BBB", "Bb", 3);
    return 0;
}
```

```
BBB equals Bb
BBB comes before Bb
BBB comes before Bb
```

トークン**strtok**、strtok_r、strtok_s

`strtok`は、りのセットをして、をさなまたはトークンにします。

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    int toknum = 0;
    char src[] = "Hello,, world!";
    const char delimiters[] = ", !";
```

```
    char *token = strtok(src, delimiters);
    while (token != NULL)
    {
        printf("%d: [%s]\n", ++toknum, token);
        token = strtok(NULL, delimiters);
    }
    /* source is now "Hello\0, world\0\0" */
}
```

```
1: [Hello]
2: [world]
```

デリミタのには1つのデリミタがまれ、 strtokびすたびになるデリミタをできます。

じソースをトークンしける strtokびしは、のをもうすべきではなく、そのわりにのとして NULLをし NULL。 じソースがされた、わりにのトークンがトークンされます。つまり、じりをすると、 strtokはにのトークンをします。

strtokはトークンにしいメモリをりてないので、ソースをすることにしてください。つまり、のでは、 srcは、 strtokへのびしによってされたポインタによってされるトークンをするようにされます。つまり、ソースを constすることはできませんリテラルにすることはできません。また、りバイトのアイデンティティがわれていることもしますでは、 "、"と ""がソースからにされ、どのりがしたかはわかりません。

また、ソースののりは1つとしてわれることにしてください。このでは、2のコンマはされます。

strtokは、にバッファをするため、スレッドセーフでもリエントラントでもありません。これは、びしたということ strtok 、しているにびしていない strtokすることもでき strtok 、それがしてをあるのによってびすことはできません strtok。

strtokがではないというによってきこされるをすはのとおりです。

```
char src[] = "1.2,3.5,4.2";
char *first = strtok(src, ",");

do
{
    char *part;
    /* Nested calls to strtok do not work as desired */
    printf("[%s]\n", first);
    part = strtok(first, ".");
    while (part != NULL)
    {
        printf(" [%s]\n", part);
        part = strtok(NULL, ".");
    }
} while ((first = strtok(NULL, ",")) != NULL);
```

```
[1.2]
 [1]
 [2]
```

されるは、のである`do while`ループからなる3つのトークンをするが`"1.2"`、`"3.5"`、`"4.2"`これらの々のために、`strtok`ループがにしなければならないためにびし`"1"`、`"2"`、`"3"`、`"5"`、`"4"`、`"2"`をむ。

しかし、`strtok`はリエントラントではないので、これはしません。わりに、の`strtok`しく"1.2\0"トークンをし、ループはトークン`"1"`と`"2"`しくします。しかし、ループの`strtok`は、ループによってされるのにあり、すぐにNULLをします。`src`の2と3のはまったくされません。

C11

のCライブラリには、スレッドセーフなバージョンやリエントラントなバージョンはまれていませんが、POSIXの`strtok_r`などののバージョンもあります。 MSVCでは`strtok`との`strtok_s`がスレッドセーフであることにしてください。

C11

C11には、`strtok_s`というのスレッドセーフでリエントラントなバージョンをするオプションパートAnnex Kがあります。`__STDC_LIB_EXT1__`をしてをテストできます。このオプションはくサポートされていません。

`strtok_s`はPOSIX `strtok_s`とはなり、トークンされているのへのやのチェックを`strtok_r`ます。しかし、しくかれたプログラムでは、`strtok_s`と`strtok_r`はじようにします。

こので`strtok_s`をすると、しいがられるようになります。

```
/* you have to announce that you want to use Annex K */
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>

#ifndef __STDC_LIB_EXT1__
# error "we need strtok_s from Annex K"
#endif

char src[] = "1.2,3.5,4.2";
char *next = NULL;
char *first = strtok_s(src, ",", &next);

do
{
    char *part;
    char *posn;

    printf("[%s]\n", first);
    part = strtok_s(first, ".", &posn);
    while (part != NULL)
    {
        printf(" [%s]\n", part);
        part = strtok_s(NULL, ".", &posn);
    }
}
while ((first = strtok_s(NULL, ",", &next)) != NULL);
```

はのようになります。

```
[1.2]
 [1]
 [2]
[3.5]
 [3]
 [5]
[4.2]
 [4]
 [2]
```

ののかのオカレンスをします**strchr**、strrchr

strchrとstrrchrは、NULでするののをします。 strchrはのオカレンスへのポインタをし、 strrchrはのオカレンスへのポインタをします。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    char toSearchFor = 'A';

    /* Exit if no second argument is found. */
    if (argc != 2)
    {
        printf("Argument missing.\n");
        return EXIT_FAILURE;
    }

    {
        char *firstOcc = strchr(argv[1], toSearchFor);
        if (firstOcc != NULL)
        {
            printf("First position of %c in %s is %td.\n",
              toSearchFor, argv[1], firstOcc-argv[1]); /* A pointer difference's result
                                is a signed integer and uses the length modifier 't'. */
        }
        else
        {
            printf("%c is not in %s.\n", toSearchFor, argv[1]);
        }
    }

    {
        char *lastOcc = strrchr(argv[1], toSearchFor);
        if (lastOcc != NULL)
        {
            printf("Last position of %c in %s is %td.\n",
              toSearchFor, argv[1], lastOcc-argv[1]);
        }
    }

    return EXIT_SUCCESS;
}
```

posというのファイルをした

```
$ ./pos AAAAAAA
First position of A in AAAAAAA is 0.
Last position of A in AAAAAAA is 6.
$ ./pos BAbbbbbAccccAAAAzzz
First position of A in BAbbbbbAccccAAAAzzz is 1.
Last position of A in BAbbbbbAccccAAAAzzz is 15.
$  ./pos qwerty
A is not in qwerty.
```

`strrchr`ないの1つは、パスからファイルをすることです。たとえば、 `C:\Users\eak\myfile.txt`から`myfile.txt`をするには

```
char *getFileName(const char *path)
{
    char *pend;

    if ((pend = strrchr(path, '\')) != NULL)
        return pend + 1;

    return NULL;
}
```

ののりし

のさがわかっているは、forルーープをしてそのをりしできます。

```
char * string = "hello world"; /* This 11 chars long, excluding the 0-terminator. */
size_t i = 0;
for (; i < 11; i++) {
    printf("%c\n", string[i]);    /* Print each character of the string. */
}
```

わりに、の`strlen()`をして、のがわからないにのさをすることができます。

```
size_t length = strlen(string);
size_t i = 0;
for (; i < length; i++) {
    printf("%c\n", string[i]);    /* Print each character of the string. */
}
```

に、Cのがヌルですることがされていることをすることができますのの`strlen()`にしたときにすでにっていました;-)。そのサイズになくをりしし、ヌルにするとをできます

```
size_t i = 0;
while (string[i] != '\0') {       /* Stop looping when we reach the null-character. */
    printf("%c\n", string[i]);    /* Print each character of the string. */
    i++;
}
```

のな

Cでは、 はヌル '\ 0'でわるです。

リテラルをしてをすることができます。リテラルはでまれたです。えば、リテラル`"hello world"`る。リテラルはにヌルでします。

いくつかのをってをすることができます。えば、 `char *`をしてののをすようにすることができます

```
char * string = "hello world";
```

のように`char *`をにする、、はみりデータにりてられます。 `string`はののへのポインタで、`'h'`です。

リテラルはみりメモリにりてられているため、です[1]。それをしようとするとのにつながるので、 `const`をしてこのようなコンパイルエラーをするがい

```
char const * string = "hello world";
```

これは、の[2]をっています

```
char const string_arr[] = "hello world";
```

なをするには、をし、リテラルをしてそのをすることができます。

```
char modifiable_string[] = "hello world";
```

これはとじです

```
char modifiable_string[] = {'h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\0'};
```

2のバージョンはでまれたイニシャライザをしているため、はに`'\0'`がにまれていないり、はにNULLでしません。

1とは、リテラルのをできないことをしますが、ポインタ`string`をすることができますのをすか、またはすることができます。

2のは、ののをできないというでのがあります。 `string`は`char`へのポインターであり、 なIであるため、`string_arr`がなIであるにインクリメントしたり、のをポイントしたりすることはできません。

ストリングのの

のは、いくつかのことをします

1. が`char *` sの
2. が`char`のである

のようにポインタのをできます

```
char * string_array[] = {
    "foo",
    "bar",
    "baz"
};
```

リテラルを`char *`すると、がみみメモリにりてられます。ただし、`string_array`はみり/きみメモリにりてられます。つまり、のポインタをできますが、ポインタがすをすることはできません。

Cでは、main `argv` プログラムのにされたコマンドラインのへのパラメ―タは、 `char * char * argv[]`です。

のをすることもできます。はのなので、のはなるのであるです。

```
char modifiable_string_array_literals[][4] = {
    "foo",
    "bar",
    "baz"
};
```

これはのようになります。

```
char modifiable_string_array[][4] = {
    {'f', 'o', 'o', '\0'},
    {'b', 'a', 'r', '\0'},
    {'b', 'a', 'z', '\0'}
};
```

の2ののサイズとして`4`をすることにしてください。たちののはには4バイトです。ヌルをめるがあるからです。

strstr

```
/* finds the next instance of needle in haystack
   zbpos: the zero-based position to begin searching from
   haystack: the string to search in
   needle: the string that must be found
   returns the next match of `needle` in `haystack`, or -1 if not found
*/
int findnext(int zbpos, const char *haystack, const char *needle)
{
    char *p;

    if (((p = strstr(haystack + zbpos, needle)) != NULL)
        return p - haystack;

    return -1;
}
```

`strstr`は`haystack` の`needle`すをします。つかった、 `strstr`はオカレンスのアドレスをします。`needle`がつからないは、NULLをします。 `zbpos`をってじをももしけることはありません。のインスタンスをスキップするために、 `zbpos`オフセットをします。メモのクロ―ンは、のようなダイア

ログをするために`findnext`をこのようにびすことができます。

```
/*
    Called when the user clicks "Find Next"
    doc: The text of the document to search
    findwhat: The string to find
*/
void onfindnext(const char *doc, const char *findwhat)
{
    static int i;

    if ((i = findnext(i, doc, findwhat)) != -1)
        /* select the text starting from i and ending at i + strlen(findwhat) */
    else
        /* display a message box saying "end of search" */
}
```

リテラル

リテラルは、ヌルしたの`char`をします。それらはなをつため、リテラルまたはじとなるへのポインタは、へのポインタができないいくつかのでにできます。たとえば、からリテラルをすと、にされたがられます。

```
const char *get_hello() {
    return "Hello, World!";  /* safe */
}
```

なから、リテラルにするのはには`const`はありません。それにもかかわらず、それらをしようとするとのがします。、リテラルにするをしようとするプログラムは、クラッシュするか、そうでなければします。

```
char *foo = "hello";
foo[0] = 'y';  /* Undefined behavior - BAD! */
```

ポインター—がリテラルをしている、またはポインター—の`const`をして、このようなのるいをってうことをけることがされます。

```
const char *foo = "hello";
/* GOOD: can't modify the string pointed to by foo */
```

、リテラルのとなるへのポインタは、になものではありません。そのはのものをすようににすることができます

```
char *foo = "hello";
foo = "World!"; /* OK - we're just changing what foo points to */
```

さらに、 `char`のイニシャライザは`char`リテラルとじをつことができますが、このようなイニシャライザのは、されたのリテラルのをえません。イニシャライザは、にのさとをするだけです。に、にされていなければはです`const`

```
char foo[] = "hello";
foo[0] = 'y';  /* OK! */
```

をゼロにする

またはのメモリブロックをゼロにするために`memset`をびすことができます。

`str`はゼロにする、 `n`はのバイトです。

```
#include <stdlib.h> /* For EXIT_SUCCESS */
#include <stdio.h>
#include <string.h>


int main(void)
{
  char str[42] = "fortytwo";
  size_t n = sizeof str; /* Take the size not the length. */

  printf("'%s'\n", str);

  memset(str, '\0', n);

  printf("'%s'\n", str);

  return EXIT_SUCCESS;
}
```

```
'fortytwo'
''
```

もうつの

```
#include <stdlib.h> /* For EXIT_SUCCESS */
#include <stdio.h>
#include <string.h>


#define FORTY_STR "forty"
#define TWO_STR "two"

int main(void)
{
  char str[42] = FORTY_STR TWO_STR;
  size_t n = sizeof str; /* Take the size not the length. */
  char * point_to_two = strstr(str, TWO_STR);

  printf("'%s'\n", str);

  memset(point_to_two, '\0', n);

  printf("'%s'\n", str);

  memset(str, '\0', n);

  printf("'%s'\n", str);
```

```
  return EXIT_SUCCESS;
}
```

```
'fortytwo'
'forty'
''
```

strspnとstrcspn

がえられた、 strspnはのspanのさをのリストのみからします。 strcspnはていますが、リストされたのからなるののさをするをいてはじです

```
/*
  Provided a string of "tokens" delimited by "separators", print the tokens along
  with the token separators that get skipped.
*/
#include <stdio.h>
#include <string.h>

int main(void)
{
    const char sepchars[] = ",.;!?";
    char foo[] = ";ball call,.fall gall hall!?.,";
    char *s;
    int n;

    for (s = foo; *s != 0; /*empty*/) {
        /* Get the number of token separator characters. */
        n = (int)strspn(s, sepchars);

        if (n > 0)
            printf("skipping separators: << %.*s >> (length=%d)\n", n, s, n);

        /* Actually skip the separators now. */
        s += n;

        /* Get the number of token (non-separator) characters. */
        n = (int)strcspn(s, sepchars);

        if (n > 0)
            printf("token found: << %.*s >> (length=%d)\n", n, s, n);

        /* Skip the token now. */
        s += n;
    }

    printf("== token list exhausted ==\n");

    return 0;
}
```

ワイドをするはwcsspnとwcscspnです。らはじようにされています。

のコピー

ポインタのりてはをコピーしません

あなたはすることができます=をコピーするをしますが、することはできません=そうして、のヌルとのとしてされているCにCでのをコピーするを= アドレスのみがされますをポインタ。

```
#include <stdio.h>

int main(void) {
    int a = 10, b;
    char c[] = "abc", *d;

    b = a; /* Integer is copied */
    a = 20; /* Modifying a leaves b unchanged - b is a 'deep copy' of a */
    printf("%d %d\n", a, b); /* "20 10" will be printed */

    d = c;
    /* Only copies the address of the string -
    there is still only one string stored in memory */

    c[1] = 'x';
    /* Modifies the original string - d[1] = 'x' will do exactly the same thing */

    printf("%s %s\n", c, d); /* "axc axc" will be printed */

    return 0;
}
```

のは、 `char d[3]`ではなく`char *d`をしているためコンパイルされています。 をするとコンパイルエラーがします。 Cのにりてることはできません。

```
#include <stdio.h>

int main(void) {
    char a[] = "abc";
    char b[8];

    b = a; /* compile error */
    printf("%s\n", b);

    return 0;
}
```

をしたのコピー

strcpy()

にをコピーするには、 `strcpy()`を`string.h`でできます。 コピーするに、なスペースをにりてるがあります。

```
#include <stdio.h>
#include <string.h>
```

```
int main(void) {
    char a[] = "abc";
    char b[8];

    strcpy(b, a); /* think "b special equals a" */
    printf("%s\n", b); /* "abc" will be printed */

    return 0;
}
```

C99

snprintf()

バッファオ―バ―ランをけるために、 `snprintf()` をすることができます。テンプレートをするがあるため、パフォ―マンスのでなソリュ―ションではありませんが、ライブラリですぐにできるをコピ―するためののバッファセ―フです。

```
#include <stdio.h>
#include <string.h>

int main(void) {
    char a[] = "012345678901234567890";
    char b[8];

#if 0
    strcpy(b, a); /* causes buffer overrun (undefined behavior), so do not execute this here!
*/
#endif

    snprintf(b, sizeof(b), "%s", a); /* does not cause buffer overrun */
    printf("%s\n", b); /* "0123456" will be printed */

    return 0;
}
```

strncat()

`strncat()` `strcat()`バッファオ―バ―フロ―チェックバ―ジョン`strncat()` をすることです - それはコピ―するバイトをする3のをとります

```
char dest[32];

dest[0] = '\0';
strncat(dest, source, sizeof(dest) - 1);
    /* copies up to the first (sizeof(dest) - 1) elements of source into dest,
    then puts a \0 on the end of dest */
```

このは`sizeof(dest) - 1`することにしてください。 `strncat()`に`strncat()` をしますが、それはのサイズでカウントされませんとバッファのきの`strncat()`ので、これはです。

また、でないのの - はさらにであることにしてください。してください

```
char dst[24] = "Clownfish: ";
char src[] = "Marvin and Nemo";
size_t len = strlen(dst);

strncat(dst, src, sizeof(dst) - len - 1);
printf("%zu: [%s]\n", strlen(dst), dst);
```

はのとおりです。

```
23: [Clownfish: Marvin and N]
```

ただし、さとしてされたサイズは、ののサイズではなく 、そのにっているスペ―スのであり、のヌルバイトをえないことにしてください。これはきなきのをきこすがあります。それはまた、しです。 lengthをしくするには、のデ―タのさをっているので、 `strncat()`スキャンをして、のコンテンツのにnullバイトのアドレスをすることができます。

```
    strcpy(dst, "Clownfish: ");
    assert(len < sizeof(dst) - 1);
    strncat(dst + len, src, sizeof(dst) - len - 1);
    printf("%zu: [%s]\n", strlen(dst), dst);
```

これはとじをしますが、 `strncat()`はコピ―をするに`dst`ののをスキャンするはありません。

`strncpy()`

のオプションは`strncpy()`です。あなたがにるはずだとうかもしれませんが、それは2つのなをっている、

1. `strncpy()`をしたコピ―がバッファに`strncpy()`と、ヌルはきまれません。
2. `strncpy()`は、にじてにデスティネ―ションをにたします。

このようななはなもので、はUNIXのファイルをうためのものでした

それをするのしいは、でヌルをすることです。

```
strncpy(b, a, sizeof(b)); /* the third parameter is destination buffer size */
b[sizeof(b)/sizeof(*b) - 1] = '\0'; /* terminate the string */
printf("%s\n", b); /* "0123456" will be printed */
```

それでもきなバッファがある、のヌルパディングのために`strncpy()`をすることはににになります。

をにするatoi、atofな、しないでください

`atoi` 、 `atol` 、 `atoll` 、 `atof`はにではありません のができない、そのはです。 7.20.1p1

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv)
{
```

```
    int val;
    if (argc < 2)
    {
        printf("Usage: %s <integer>\n", argv[0]);
        return 0;
    }

    val = atoi(argv[1]);

    printf("String value = %s, Int value = %d\n", argv[1], val);

    return 0;
}
```

されるがのな10の、はのようにします。

```
$ ./atoi 100
String value = 100, Int value = 100
$ ./atoi 200
String value = 200, Int value = 200
```

でまり、かがくの、のだけがされます。

```
$ ./atoi 0x200
0
$ ./atoi 0123x300
123
```

のすべての、そのはです。

```
$ ./atoi hello
Formatting the hard disk...
```

のあいまいさとこののののため、 `atoi`ファミリのはしてしないでください。

- `long int`にするには、 `atol()`わりに`strtol()`します。
- `double`にするには、 `strtod()`わりに`atof()`ます。

C99

- `long long int`するには、 `atoll()`わりに`strtoll()`します。

のデ―タのみき

フォ―マットされたデ―タをにきす

```
int sprintf ( char * str, const char * format, ... );
```

`sprintf`をしてfloatデ―タをにきします。

```
#include <stdio.h>
```

```
int main ()
{
  char buffer [50];
  double PI = 3.1415926;
  sprintf (buffer, "PI = %.7f", PI);
  printf ("%s\n",buffer);
  return 0;
}
```

からされたデ―タをみむ

```
int sscanf ( const char * s, const char * format, ...);
```

`sscanf`をして、されたデ―タをします。

```
#include <stdio.h>
int main ()
{
  char sentence []="date : 06-06-2012";
  char str [50];
  int year;
  int month;
  int day;
  sscanf (sentence,"%s : %2d-%2d-%4d", str, &day, &month, &year);
  printf ("%s -> %02d-%02d-%4d\n",str, day, month, year);
  return 0;
}
```

にストリングをにする**strtoX**

C99

C99、Cライブラリには、をとしてするのがあります。それらのは`strtoX`のであり、 `X`は`l` 、 `ul` 、`d`などのいずれかであり、のタ―ゲット.タイプをします

```
double strtod(char const* p, char** endptr);
long double strtold(char const* p, char** endptr);
```

それらは、コンバ―ジョンにオ―バ―フロ―またはアンダ―フロ―があることをします。

```
double ret = strtod(argv[1], 0); /* attempt conversion */

/* check the conversion result. */
if ((ret == HUGE_VAL || ret == -HUGE_VAL) && errno == ERANGE)
    return;  /* numeric overflow in in string */
else if (ret == HUGE_VAL && errno == ERANGE)
    return; /* numeric underflow in in string */

/* At this point we know that everything went fine so ret may be used */
```

ににがまったくまれていない、 `strtod`このは`0.0`します。

これがのいくものでなければ、のパラメ―タ`endptr`をできます。のされたのわりをすポインタへ

のポインタです。のように0 、またはNULLにされているは、にされます。

このendptrパラメータは、がしたかどうかをし、したはしたをしendptr。

```
char *check = 0;
double ret = strtod(argv[1], &check); /* attempt conversion */

/* check the conversion result. */
if (argv[1] == check)
    return; /* No number was detected in string */
else if ((ret == HUGE_VAL || ret == -HUGE_VAL) && errno == ERANGE)
    return; /* numeric overflow in in string */
else if (ret == HUGE_VAL && errno == ERANGE)
    return; /* numeric underflow in in string */

/* At this point we know that everything went fine so ret may be used */
```

よりいにするのがあります

```
long strtol(char const* p, char** endptr, int nbase);
long long strtoll(char const* p, char** endptr, int nbase);
unsigned long strtoul(char const* p, char** endptr, int nbase);
unsigned long long strtoull(char const* p, char** endptr, int nbase);
```

これらのには、がきまれるベースをする3のパラメータnbaseがあります。

```
long a = strtol("101",   0, 2 ); /* a = 5L */
long b = strtol("101",   0, 8 ); /* b = 65L */
long c = strtol("101",   0, 10); /* c = 101L */
long d = strtol("101",   0, 16); /* d = 257L */
long e = strtol("101",   0, 0 ); /* e = 101L */
long f = strtol("0101",  0, 0 ); /* f = 65L */
long g = strtol("0x101", 0, 0 ); /* g = 257L */
```

nbaseのな0は、がCプログラムでリテラルがされるのとじでされることをします。0xは16にし、それのはの0は8であり、のすべてのは10でされます。

したがって、コマンドラインのをとしてするもなは、のようになります。

```
int main(int argc, char* argv[] {
    if (argc < 1)
        return EXIT_FAILURE; /* No number given. */

    /* use strtoull because size_t may be wide */
    size_t mySize = strtoull(argv[1], 0, 0);

    /* then check conversion results. */

     ...

    return EXIT_SUCCESS;
}
```

つまり、プログラムは8、10、または16のパラメータでびすことができます。

オンライ ンでをむ https://riptutorial.com/ja/c/topic/1990/

49: およびな

- 「キャスティング」タイプ

「 」は、に「キャスティング」ともばれます。

Examples

びしにおける

がなプロトタイプをっているとすれば、のにってへのびしのためにがされる。C11 6.3.1.3。

> **6.3.1.3**きとなし
> のが_Boolのにされた、そのをしいですことができるはされません。
>
> それの、しいがなしの、そのがしいのにるまで、しいでできるよりも1をりしてまたは することによってがされます。
>
> それのは、しいがされ、そのをそのですることはできません。がインプリメンテーション によってされるか、インプリメンテーションのシグナルがします。

らかにがまりきれず、これにはながないので、は、のあるきのをよりいきのにりてるべきではありません。でしたCは、これらのケースを「 」としています。つまり、がありません。

のでは、 `int`が32ビットであるとしています。

```
#include <stdio.h>
#include <stdint.h>

void param_u8(uint8_t val) {
    printf("%s val is %d\n", __func__, val);  /* val is promoted to int */
}

void param_u16(uint16_t val) {
    printf("%s val is %d\n", __func__, val);  /* val is promoted to int */
}

void param_u32(uint32_t val) {
    printf("%s val is %u\n", __func__, val);  /* here val fits into unsigned */
}

void param_u64(uint64_t val) {
    printf("%s val is " PRI64u "\n", __func__, val); /* Fixed with format string */
}

void param_s8(int8_t val) {
    printf("%s val is %d\n", __func__, val);  /* val is promoted to int */
}

void param_s16(int16_t val) {
    printf("%s val is %d\n", __func__, val);  /* val is promoted to int */
```

```
}

void param_s32(int32_t val) {
    printf("%s val is %d\n", __func__, val); /* val has same width as int */
}

void param_s64(int64_t val) {
    printf("%s val is " PRI64d "\n", __func__, val); /* Fixed with format string */
}

int main(void) {

    /* Declare integers of various widths */
    uint8_t  u8  = 127;
    uint8_t  s64  = INT64_MAX;

    /* Integer argument is widened when function parameter is wider */
    param_u8(u8);   /* param_u8 val is 127 */
    param_u16(u8);  /* param_u16 val is 127 */
    param_u32(u8);  /* param_u32 val is 127 */
    param_u64(u8);  /* param_u64 val is 127 */
    param_s8(u8);   /* param_s8 val is 127 */
    param_s16(u8);  /* param_s16 val is 127 */
    param_s32(u8);  /* param_s32 val is 127 */
    param_s64(u8);  /* param_s64 val is 127 */

    /* Integer argument is truncated when function parameter is narrower */
    param_u8(s64);  /* param_u8 val is 255 */
    param_u16(s64); /* param_u16 val is 65535 */
    param_u32(s64); /* param_u32 val is 4294967295 */
    param_u64(s64); /* param_u64 val is 9223372036854775807 */
    param_s8(s64);  /* param_s8 val is implementation defined */
    param_s16(s64); /* param_s16 val is implementation defined */
    param_s32(s64); /* param_s32 val is implementation defined */
    param_s64(s64); /* param_s64 val is 9223372036854775807 */

    return 0;
}
```

びしにおけるポインタ

`void*`へのポインタはですが、のポインタはでなければなりません。コンパイラはのポインタからデータへのなをしますが、ったきポインタによるオブジェクトへのアクセスはりであり、のにつながります。これらがされるのケースは、ががあるか、またはオブジェクトをているポインタがであるかどうかです。

```
#include <stdio.h>

void func_voidp(void* voidp) {
    printf("%s Address of ptr is %p\n", __func__, voidp);
}

/* Structures have same shape, but not same type */
struct struct_a {
    int a;
    int b;
} data_a;
```

```
struct struct_b {
    int a;
    int b;
} data_b;

void func_struct_b(struct struct_b* bp) {
    printf("%s Address of ptr is %p\n", __func__, (void*) bp);
}

int main(void) {


    /* Implicit ptr conversion allowed for void* */
    func_voidp(&data_a);

    /*
     * Explicit ptr conversion for other types
     *
     * Note that here although the have identical definitions,
     * the types are not compatible, and that the this call is
     * erroneous and leads to undefined behavior on execution.
     */
    func_struct_b((struct struct_b*)&data_a);

    /* My output shows: */
    /* func_charp Address of ptr is 0x601030 */
    /* func_voidp Address of ptr is 0x601030 */
    /* func_struct_b Address of ptr is 0x601030 */

    return 0;
}
```

オンラインでおよびなをむ https://riptutorial.com/ja/c/topic/2529/およびな

50: の

き

Cでは、いくつかのはのをもたらします 。では、コンパイラがそのようなにしたのをしないことをにします。として、コンパイラは、たにったものをにすることができ、な、しない、またはクラッシュさえきこすがあります。

UBをびすコードは、のコンパイラをしてのシステムでされたとおりにするがありますが、のシステムやのコンパイラ、コンパイラのバージョンまたはコンパイラのではしないがあります。

UBとはですか

のは、Cでされるです。 C11ISO / IEC 98992011では、されていないというを

> のないまたはったプログラムのまたはったデータのに、このはをさない

のコードに**UB**があるとどうなりますか

これらは、にったのるいのためにこりるです

> ののは、なをにしてをすること、のされたでまたはプログラムのにメッセージののにかかわらずメッセージの。

のは、されていないからじるではありませんがをするためによくされます。

> コンパイラがANSI Cにすることなくコードをするためにのなをするがあることをしています

なぜ**UB**はするのですか

それがいのであれば、なぜにするのか、それともにするのでしょうか

のにより、のがえます。コンパイラは、すべてのコードにのビヘイビアがまれていないとすることができます。のチェックをけ、がまたはなをできます。

なぜ**UB**はがしいのですか

されていないがしにくいバグをするはなくとも2つあります。

- コンパイラは、のについてするはありません。にそうすることをすることは、のるいがするににがるでしょう。
- ビヘイビアがのコンストラクトがするのなで、できないがされないがあります。のビヘイビアはをし、そのはいつでもするがあります。コンストラクトの、、または

NULLポインタのをしてください。コンパイラは、NULLポインタのをするはなく、ににされたポ

ィンタやグロ―バルでNULLがするもありません。 また、ヌルポインタがすると、ではプログラムがクラッシュするがあるとはりません。むしろ、プログラムがにクラッシュしたり、でクラッシュしたり、クラッシュしたりすることはありません。まるでヌルポインタがなオブジェクトをしし、ににし、のではクラッシュするようにするがあります。

nullポインタのの、CはnullポインタののがされているJavaやCなどのとはなります。なにがスロ―されますJavaでは`NullPointerException` 、 Cでは`NullReferenceException` したがって、JavaまたはCからのメッセ―ジは、メッセ―ジののにかかわらず、Cプログラムがクラッシュしなければならないとってじているがあります 。

このようなをにするがあります。

- にされていない、つまり、Cによってにオフリミットであることがされます。
- のうちにされていないるい。には、プログラムをってきたのるいをするテキストはまったくありません。

また、コンパイラやライブラリのがのをすをすために、Cではにののがにされていないことがくのでされています。いはシグナルとシグナルハンドラです。POSIXオペレ―ティングシステムののような、Cへのがはるかになル―ルをしています。そのようなは、プラットフォ―ムのドキュメントをチェックするだけです。 Cのはあなたにもえることができません。

プログラムでのがした、のビヘイビアがしただけがになり、プログラムがになることをしないことにもしてください。

このようながあるため、Cでのプログラミングのにとっては、なくとものをきこすようなことにしていることがですに、コンパイラはUBについてにしているわけではないため。

のをするのについくつかのツ―ルえば、PC-Lintなどのツ―ルがありますが、ののすべてのをすることはできません。

Examples

NULLポインタをする

これはNULLポインタののであり、のをきこします。

```
int * pointer = NULL;
int value = *pointer; /* Dereferencing happens here */
```

`NULL`ポインタは、Cによってされ、なオブジェクトへのポインタとしくないものをし、はのをびします。

2つのシ―ケンスポイントでオブジェクトをする

```
int i = 42;
i = i++; /* Assignment changes variable, post-increment as well */
```

```
int a = i++ + i--;
```

このようなコ―ドは、しばしばi「」についてのにつながります。しかし、をするのではなく、Cのでは、そのようなをするとのがされることがされています。 C2011よりのこのは、いわゆるシ―ケンスポイントのからこれらのル―ルをしました。

> のシ―ケンスポイントとのシ―ケンスポイントのでは、スカラ―オブジェクトは、でも1ののによってストアされたをしなければならない。さらに、のは、されるをするためにのみみされるものとする。

C99、6.5、パラグラフ2

そのスキ―ムはあまりにもすぎるとし、いくつかのが、C99にしてはあまりがないはずののをすとなった。 C2011はシ―クエンスポイントをしますが、 シ―クエンシングと「 シ―クエンシングされた」とばれるにづいて、このへのよりなアプロ―チをしています。

> じスカラオブジェクトのなるまたはじスカラ―オブジェクトのをするのいずれかにして、スカラオブジェクトのがけされていない、そのはです。ののながある、そのけされていないがいずれかのですると、そのはされません。

C2011、6.5、パラグラフ2

「けられた」のはここではするにはがかかりすぎますが、シ―ケンスポイントをきえるのではなくシ―ケンスポイントをするので、るいがはされていないのるいをするがあります。に、2つののにシ―ケンスポイントがある、シ―ケンスポイントののシ―ケンスポイントは、シ―ケンスポイントののシ―ケンスポイントに「け」されます。

のは、にされたをしています。

```
int i = 42;
i = (i++, i+42); /* The comma-operator creates a sequence point */
```

のでは、のがあります。

```
int i = 42;
printf("%d %d\n", i++, i++); /* commas as separator of function arguments are not comma-
operators */
```

されていないるいとに、けル―ルにするをするのをすることは、なをいてではありません。は、そのようなが、じプログラムののであってもであるとするためのをしない。

をすにreturnステ―トメントがありません

```
int foo(void) {
  /* do stuff */
  /* no return here */
}
```

```
int main(void) {
  /* Trying to use the (not) returned value causes UB */
  int value = foo();
  return 0;
}
```

がをすとされた、それをるのあるすべてのコ―ドパスでをするがあります。のは、びしりががり[1]をしようとするとすぐにします。

されていないは、びしがのを/アクセスしようとしたにのみします。えば、

```
int foo(void) {
  /* do stuff */
  /* no return here */
}

int main(void) {
  /* The value (not) returned from foo() is unused. So, this program
   * doesn't cause *undefined behaviour*. */
  foo();
  return 0;
}
```

C99

`main()`は、りので、それがreturnステ―トメントなしですることがであるというで、こののである`0`にこのケ―ス[2]にされます。

---

[1] *ISO / IEC 9899201x* 、6.9.1 / 12

をする}にし、びしのがびしによってされている、そのはです。

[2] *ISO / IEC 9899201x* 、5.1.2.2.3 / 1

}にすると、mainをし、0のをします。

きオ―バ―フロ―

C99とC11の6.5 / 5では、がののなでない、のによってのがされます。の、それはオ―バ―フロ―とばれます。6.2.5 / 9がされるため、なしがオ―バ―フロ―しないため、のなしがのにされます。しかし、 きにはのはありません。これらはオ―バ―フロ―するがあり、のをきこします。えば、

```
#include <limits.h>      /* to get INT_MAX */

int main(void) {
    int i = INT_MAX + 1; /* Overflow happens here */
    return 0;
}
```

このタイプののビヘイビアのほとんどのインスタンスは、またはするのがよりです。オ―バ―フ

ロ―はとして、きののをけるにする、、またはからするがあります。たとえば、このはのようになります。

```
int square(int x) {
    return x * x;  /* overflows for some values of x */
}
```

であり、のがさいはなをいますが、のがきいはそのはされていません。をびすプログラムがとしてのるいをすかどうかは、だけでることはできません。それはらがどのをすかによってなります。

、オ―バ―フロ―セ―フきのなをえてみましょう

```
int zero(int x) {
    return x - x;  /* Cannot overflow */
}
```

のオペランドのにより、がオ―バ―フロ―しないことがされます。または、これをややなとしてえてみましょう。

```
int sizeDelta(FILE *f1, FILE *f2) {
    int count1 = 0;
    int count2 = 0;
    while (fgetc(f1) != EOF) count1++;  /* might overflow */
    while (fgetc(f2) != EOF) count2++;  /* might overflow */

    return count1 - count2; /* provided no UB to this point, will not overflow */
}
```

カウンタがにオ―バ―フロ―しないり、のオペランドはともではありません。そのような2つののはすべて`int`としてできます。

されていないの

```
int a;
printf("%d", a);
```

`a`は、をつ`int`です。ののコ―ドは、されていないのをしようとしています `a`はされていません。されていないにはながあります。これらにアクセスするとのにつながるがあります。

`static`キ―ワ―ドのないグロ―バルをむまたはスレッドロ―カルをつは、ゼロまたはされたにされます。したがって、はです。

```
static int b;
printf("%d", b);
```

にないは、カウンタ―としてするを0にしないことです。をしますが、はガ―ベジであるため、タ―ミナルのコンパイルにポインタをすなどのビヘイビア―がびされます。なシンボル 。

```
#include <stdio.h>

int main(void) {
    int i, counter;
    for(i = 0; i < 10; ++i)
        counter += i;
    printf("%d\n", counter);
    return 0;
}
```

```
C02QT2UBFVH6-lm:~ gsamaras$ gcc main.c -Wall -o main
main.c:6:9: warning: variable 'counter' is uninitialized when used here [-Wuninitialized]
        counter += i;
        ^~~~~~~
main.c:4:19: note: initialize the variable 'counter' to silence this warning
    int i, counter;
                  ^
                   = 0
1 warning generated.
C02QT2UBFVH6-lm:~ gsamaras$ ./main
32812
```

のは、ポインタにもできます。たとえば、のはのになります

```
int main(void)
{
    int *p;
    p++; // Trying to increment an uninitialized pointer.
}
```

のコードはでエラーまたはセグメンテーションフォールトをきこさないかもしれないが、このポインターをでしようとするとのがこることにしてください。

へのポインタをそのをえてする

```
int* foo(int bar)
{
    int baz = 6;
    baz += bar;
    return &baz; /* (&baz) copied to new memory location outside of foo. */
} /* (1) The lifetime of baz and bar end here as they have automatic storage
   * duration (local variables), thus the returned pointer is not valid! */

int main (void)
{
    int* p;

    p = foo(5);  /* (2) this expression's behavior is undefined */
    *p = *p - 6; /* (3) Undefined behaviour here */

    return 0;
}
```

いくつかのコンパイラがこれをにしています。たとえば、 `gcc`ようにします。

```
warning: function returns address of local variable [-Wreturn-local-addr]
```

そして`clang`ようにします

```
warning: address of stack memory associated with local variable 'baz' returned
[-Wreturn-stack-address]
```

のコードのために。しかし、コンパイラはなコードではけできないかもしれません。

1 `static`れたへののは、のスコープをれたにがされないようにされたです。

2ISO / IEC 98992011 6.2.4§2によると、ポインタがすオブジェクトがそののわりにすると、ポインタのはになります。

3`foo`によってされたポインタをすることは、されるメモリがなをするため、のです。

ゼロ

```
int x = 0;
int y = 5 / x;  /* integer division */
```

または

```
double x = 0.0;
double y = 5.0 / x;  /* floating point division */
```

または

```
int x = 0;
int y = 5 % x;  /* modulo operation */
```

の2では、2のオペランドxのが0であるため、はです。

のほとんどのは IEEE 754などにうことにしてください。この、Cではがされていないとしても、ゼロのようなにはした `INFINITY` があります。

りてられたチャンクをえてメモリにアクセスする

`n`をむメモリへのポインタは、`memory`と`memory + (n - 1)`にのみすることができます。そののポインタをすると、のになります。として、のコードをえてみましょう

```
int array[3];
int *beyond_array = array + 3;
*beyond_array = 0; /* Accesses memory that has not been allocated. */
```

3は、3のさのの4のにアクセスし、のにつながります。に、のコードフラグメントの2のもにされていません。

```
int array[3];
array[3] = 0;
```

ののをすことはのではないことにしてください `beyond_array = array + 3`はここでよくされていますが、は `*beyond_array`はのです。このルーールは、にりてられたメモリ `malloc`によってされたバッファなどにもされます。

するメモリのコピーー

なライブラリは、1つのメモリからのメモリにバイトシーケンスをコピーするというがあります。これらののほとんどは、ソースとがしている、のをします。

たとえば、これは...

```
#include <string.h> /* for memcpy() */

char str[19] = "This is an example";
memcpy(str + 7, str, 10);
```

...ソースとコピーのメモリが3バイトオーバーラップする10バイトをコピーしようとします。するには

```
               overlapping area
               |
               _ _
              |   |
              v   v
T h i s   i s   a n   e x a m p l e \0
^             ^
|             |
|             destination
|
source
```

オーバーラップのため、のはです。

こののをつライブラリには、 `memcpy()` 、 `strcpy()` 、 `strcat()` 、 `sprintf()` 、および`sscanf()`ます。では、これらのやそののについてしています。

> オーバーラップするオブジェクトでコピーがわれる、はです。

`memmove()`は、このルーールのなです。このでは、ソースデータがにバッファにコピーされ、アドレスにきまれたかのようにがするようにされています。ソースとがしていることやがないことはではないので、 `memmove()`はそのようなになをします。

いは、 をしています。トレードオフ。これらののようなコピーは、、メモリのされたでわれ、にメモリコピーののインスタンスがそのカテゴリにまれるかどうかをることができます。ノンオーバーラップをすると、がしないにしいをにしないながられる。ほとんどのCライブラリは、よりなをされ、そして`memmove()` 、ソースとがはオーバーラップをうことができるにサービスをする、

ギャップをめます。しかし、すべてのケースでしいをすためには、のテストをしなければならず、かつ/またはのいをするがあります。

メモリにバックアップされていないされていないオブジェクトのみり

C11

オブジェクトをみると、オブジェクトが[1]の、のがします。

- されていない
- で
- そのアドレスはしてられません

ののaは、これらのをすべてたしています。

```
void Function( void )
{
    int a;
    int b = a;
}
```

---

[1] ISOIEC 9899201X 6.3.2.1、、および2
がレジスタクラスでされたのあるのオブジェクトをしていて、そのオブジェクトがされていないでされておらず、にされていない 、そのはです。

データ

C11

C11ではのスレッドがサポートされ、データのがじました。プログラムは、2つのなるスレッドによって1つのオブジェクトがアクセスされた、なくとも1つのアクセスはアトミックであり、なくとも1つはオブジェクトをし、プログラムのセマンティクスは2つのアクセスがしないことをしませんに。 [2]するアクセスののはデータのではないことにしてください。データは、なるスレッドのメモリビューにおけるされるからじるよりいクラスのをカバーします。

このをえてみましょう。

```
#include <threads.h>

int a = 0;

int Function( void* ignore )
{
    a = 1;

    return 0;
}

int main( void )
```

```
{
    thrd_t id;
    thrd_create( &id , Function , NULL );

    int b = a;

    thrd_join( id , NULL );
}
```

メインスレッドは`thrd_create`をびして、しいスレッド`Function`をします。 2のスレッドは`aa` 、メインスレッドは`a`みります。これらのアクセスはどちらもアトミックではなく、2つのスレッドは、しないようにまたはでもしないため、デ―タがします。

このプログラムがデ―タをけるのには、

- メインスレッドはのスレッドを`a`に`a`みりをできます。
- メインスレッドは`thrd_join`でのスレッドがしたことをし`a`に`a`みりをできます。
- スレッドはmutexをしてそれらのアクセスをさせることができ、それぞれがそのmutexをロックしてからアクセス`a` 、ロックをすることができます。

mutexオプションがすように、デ―タをするために、メインスレッドがそれをみむに、スレッドが`a`するなど、のをするはありません。デ―タのをけるためには、のにして、のアクセスがのアクセスのにこることをすることでです。

1オブジェクトのまたはみり。

2 ISOIEC 9889201x、セクション5.1.2.4「マルチスレッドとデ―タ」より
プログラムのには、なるスレッドに2つのするアクションがまれていて、なくとも1つがアトミックではなく、どちらもアトミックではない、デ―タがまれます。そのようなデ―タは、のをもたらす。

されたポインタのをみむ

されたポインタのをみんだだけでもつまり、ポインタをしようとすることなく、のるいUBです。

```
char *p = malloc(5);
free(p);
if (p == NULL) /* NOTE: even without dereferencing, this may have UB */
{

}
```

ISO / IEC 98992011 、セクション6.2.4§2

> [...]ポインター―のは、それがしすオブジェクトまたはのものがそののわりにすると、になります。

らかにのやをむ、なメモリのは、そのがそののトラップであるがある、のるいをつがあります。

リテラルの

このコードでは、charポインタ`p`はリテラルのアドレスにされています。リテラルをしようとすると、のがします。

```
char *p = "hello world";
p[0] = 'H'; // Undefined behavior
```

しかし、`char`なをするか、ポインタをってすることは、がリテラルであってものことながらのではありません。はありません

```
char a[] = "hello, world";
char *p = a;

a[0] = 'H';
p[7] = 'W';
```

これは、がされるたびにリテラルがににコピーされるためですをつ、またはスレッドをつにしてがされるたびに、りてのはされません。のをしてもありません。

メモリを2する

メモリを2することはのです。

```
int * x = malloc(sizeof(int));
*x = 9;
free(x);
free(x);
```

からの7.20.3.2。C99のな

> それの、がcalloc、malloc、またはreallocによってされたポインタにしない、またはfreeまたはreallocのびしによってがりてされた、はです。

printfでなをう

`printf`ののになをすると、のがします。たとえば、のコードはのビヘイビアをびします。

```
long z = 'B';
printf("%c\n", z);
```

のがあります

```
printf("%f\n",0);
```

コードののはのです。`%f`はになります。ただし、0は`int`です。

コンパイルになフラグをオンにすると、コンパイラは、このようなケースをけることができます`clang`と`gcc -Wformat`。 のから

```
warning: format specifies type 'double' but the argument has type
      'int' [-Wformat]
    printf("%f\n",0);
            ~~    ^
            %d
```

ポインタのにより、がしくしない

ポインタのがしくないため、のようながになるがあります。

```
 char *memory_block = calloc(sizeof(uint32_t) + 1, 1);
 uint32_t *intptr = (uint32_t*)(memory_block + 1);  /* possible undefined behavior */
 uint32_t mvalue = *intptr;
```

ポインタがされると、のがします。 C11によれば、 *2*つのポインタののがってされたをする *6.3.2.3*、はです。ここで`uint32_t`は、たとえば2または4のアラインメントがながあります。

、 `calloc`は、のオブジェクトにしてにされたポインタをすがあります。これ`memory_block`にむようにされる`uint32_t`そののでは。に、 `uint32_t`が2または4のアライメントをとするシステムでは、`memory_block + 1`はアドレスになり、したがってしくされません。

Cは、すでにキャストがであることをしていることにしてください。これは、アドレスがセグメントされているプラットフォームでは、バイトアドレス`memory_block + 1`がポインタとしてなさえできないことがあるため、される。

`char *`をのへのポインタにキャストすることは、ファイルのヘッダやネットワークパケットなどのパックされたをデコードするために、をにすることがないことがあります。

`memcpy`をすると、ポインタのがずれることによるのをできます。

```
memcpy(&mvalue, memory_block + 1, sizeof mvalue);
```

`uint32_t*`へのポインターはわれず、バイトは1つずつコピーされます。

こののコピーでは、 `mvalue`なになるのは、 `mvalue`によるものです。

- `calloc`をしたので、バイトはにされています。たちの、すべてのバイトは`0`ちますが、のながいます。
- `uint32_t`はなので、パディングビットはありません
- のビットパターンは、のなしタイプのなです。

ポインタのまたはがしくされていない

のコードはがです。

```
char buffer[6] = "hello";
char *ptr1 = buffer - 1;  /* undefined behavior */
char *ptr2 = buffer + 5;  /* OK, pointing to the '\0' inside the array */
char *ptr3 = buffer + 6;  /* OK, pointing to just beyond */
char *ptr4 = buffer + 7;  /* undefined behavior */
```

C11によれば、オブジェクトとへのポインタのまたはが、じオブジェクトをしさない、またはそれをえたをする、そのはされていません6.5.6 。

さらに、をちょうどすポインタをすることはのことながらのです

```
char buffer[6] = "hello";
char *ptr3 = buffer + 6;  /* OK, pointing to just beyond */
char value = *ptr3;       /* undefined behavior */
```

ポインタをしてconstをする

```
int main (void)
{
    const int foo_readonly = 10;
    int *foo_ptr;

    foo_ptr = (int *)&foo_readonly; /* (1) This casts away the const qualifier */
    *foo_ptr = 20; /* This is undefined behavior */

    return 0;
}
```

ISO / IEC 9899201x 、セクション6.7.3§2

> constのをしてconstでされたオブジェクトをしようとすると、そのはです。 [...]

---

[1] GCCでは、のがスロ―されることがあります `warning: assignment discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]`

printfsへのヌルポインタのけし

`printf`の`%s`は、するがののののへのポインタであることをします。ヌルポインタはののののをしていないため、のはされていません。

```
char *foo = NULL;
printf("%s", foo); /* undefined behavior */
```

しかし、のがずしもプログラムがクラッシュすることをするわけではありません。ヌルポインタ―がされたときにするクラッシュをするためのシステムもあります。えば、Glibcはすることがられている

```
(null)
```

のコードのただし、にをすると、クラッシュすることになります。

```
char *foo = 0;
printf("%s\n", foo); /* undefined behavior */
```

この、GCCには`printf("%s\n", argument);`をすがあるためにこり`printf("%s\n", argument);`びしに`puts`と`puts(argument)` 、および`puts` Glibcのにヌルポインタをしません。このはすべてにしています。

ヌルポインタはとはなります 。したがって、はで、のはありません。それはにをします

```
char *foo = "";
printf("%s\n", foo);
```

ののないリンケージ

```
extern int var;
static int var; /* Undefined behaviour */
```

C11、§6.2.2、7はう

> でじがおよびののリンケージとともにされる、そのはです。

ののがであれば、それはののリンケージをつことにしてください。 *C11*、§6.2.2、4はそれをにする

> そののがえるでクラスでされたについては、31がまたはをする、のでののは、のでされた。のがえない、またはのがリンケージをしていない、はリンケージをちます。

```
/* 1. This is NOT undefined */
static int var;
extern int var;


/* 2. This is NOT undefined */
static int var;
static int var;

/* 3. This is NOT undefined */
extern int var;
extern int var;
```

ストリームでの**fflush**の

POSIXとCのでは、ストリームでの`fflush`はのであることをしています。 `fflush`はストリームにしてのみされます。

```
#include <stdio.h>
```

```
int main()
{
    int i;
    char input[4096];

    scanf("%i", &i);
    fflush(stdin); // <-- undefined behavior
    gets(input);

    return 0;
}
```

をストリームからするなはありません。、いくつかのでは、 `stdin`バッファをクリアするために`fflush`をします。マイクロソフトでは、ストリームで`fflush`のをしています。ストリームがにいている、 `fflush`はバッファのをクリアします。 POSIX.1-2008によれば、ファイルがシークでないり、 `fflush`のはです。

は、 `fflush(stdin)`をしてください。

のカウントをするか、またはタイプのをえるビットシフト

シフトカウントがのの、 シフトとシフトののはされていません[1]

```
int x = 5 << -3; /* undefined */
int x = 5 >> -3; /* undefined */
```

シフトがのでされる 、それはです

```
int x = -5 << 3; /* undefined */
```

シフトがのでされ、のがそのでできない、それはです[1]

```
/* Assuming an int is 32-bits wide, the value '5 * 2^72' doesn't fit
 * in an int. So, this is undefined. */

int x = 5 << 72;
```

の .eg `-5 >> 3` のシフト はではなく 、 であることにしてください。

---

[1]*ISO / IEC 9899201x* 、セクション6.5.7

> オペランドのがであるか、またはされたオペランドのである、そのはです。

getenv、strerror、および**setlocale**によってされるをする

`getenv()` 、 `strerror()`および`setlocale()`によってされるをすることはされていません。では、これらのにストレージをすることがあります。

getenv、C11、§7.22.4.7、4は 、のようにべています。

getenvは、したリストメンバーにけられたへのポインタをします。にされたは、プログラムによってされてはならないが、のgetenvのびしによってきされることがある。

strerror、C11、§7.23.6.3、4はこうっています

strerrorはへのポインタをします。そのはローカルでされています。にされたは、プログラムによってされてはならないが、そののstrerrorのびしによってきされる。

setlocale、C11、§7.11.1.1、8はこうっています

setlocaleによってされるへのポインタは、そのとそれにするカテゴリをつのびしがプログラムのロケールのそのをするようなものです。にされたは、プログラムによってされてはならないが、そののsetlocaleのびしによってきされることがある。

に、 `localeconv()`はされない`struct lconv`へのポインタをします。

localeconv、C11、§7.11.2.1、8はこうっています

localeconvは、りつぶしオブジェクトへのポインタをします。りがすは、プログラムによってされてはならないが、そののlocaleconvのびしによってきされることがある。

`_Noreturn`または`noreturn`でされたからのり

C11

`_Noreturn`はC11でされました。ヘッダー`<stdnoreturn.h>`は、 `_Noreturn`されるマクロ`noreturn`をします。したがって、 `<stdnoreturn.h>` `_Noreturn`または`noreturn`をしても`_Noreturn`。

`_Noreturn` または`noreturn` でされたは、びしにることはできません。このようながびしにっても、はです。

のでは、 `func()`は`noreturn`でされていますが、びしにります。

```
#include <stdio.h>
#include <stdlib.h>
#include <stdnoreturn.h>

noreturn void func(void);

void func(void)
{
    printf("In func()...\n");
} /* Undefined behavior as func() returns */

int main(void)
{
    func();
    return 0;
}
```

`gcc`と`clang`はのプログラムにしてをします

```
$ gcc test.c
test.c: In function ‘func’:
test.c:9:1: warning: ‘noreturn’ function does return
 }
 ^
$ clang test.c
test.c:9:1: warning: function declared 'noreturn' should not return [-Winvalid-noreturn]
}
^
```

よくされたるいをつ`noreturn`をった

```
#include <stdio.h>
#include <stdlib.h>
#include <stdnoreturn.h>

noreturn void my_exit(void);

/* calls exit() and doesn't return to its caller. */
void my_exit(void)
{
    printf("Exiting...\n");
    exit(0);
}

int main(void)
{
    my_exit();
    return 0;
}
```

オンラインでのをむ https://riptutorial.com/ja/c/topic/364/の

51:

き

は、なののセットをのメモリにグループするをします。はのまたはポインタでできます。にもにアクセスすることができます。はにされ、からされます。それらはキーワード`struct`をしてされます。

Examples

なデータ

データは、するデータをパッケージし、ののようにさせるのにです。

2つの`int`メンバーをするな`struct`をする

```
struct point
{
    int x;
    int y;
};
```

`x`と`y`は`point`のメンバまたはフィールド とばれます。

のと

```
struct point p;    // declare p as a point struct
p.x = 5;           // assign p member variables
p.y = 3;
```

はにできます。はのものとじです

```
struct point p = {5, 3};
```

は、 されたをしてすることもできます 。

フィールドへのアクセスは、 .オペレーター

```
printf("point is (x = %d, y = %d)", p.x, p.y);
```

Typedef

`struct`と`typedef`をみわせることで、コードをよりにすることができます。えば

```
typedef struct
```

```
{
    int x, y;
} Point;
```

とはに

```
struct Point
{
    int x, y;
};
```

のようにできます。

```
Point point;
```

のわりに

```
struct Point point;
```

のものをうのもい

```
typedef struct Point Point;

struct Point
{
    int x, y;
};
```

な`point`のをすること。このようなは、C ++をにんだにもです。があいまいでないは、 `struct`キーワードをすることができます。

の`typedef`は、プログラムのののののとするがあります。これはながあるとえているもいますが、`struct`とのをつほとんどのにとっては、じことがかなりになります。 NotoriousはえばPOSIXの`stat`

```
int stat(const char *pathname, struct stat *buf);
```

`struct stat`という1つのをつ`stat`がされます。

タグをたない`typedef`は、 `struct`がそれをするコードからえるようにしなければなりません。`struct`をヘッダファイルにするがあります。

してください

```
#include "bar.h"

struct foo
{
    bar *aBar;
};
```

したがって、タグをたない`typedef struct`では、 `bar.h`ファイルはに`bar`をむがあり`bar`。たちがうなら

```
typedef struct bar bar;
```

`bar.h`では、 `bar`のをすことができます。

typedefをしてください

へのポインタ

`struct`をむがあるは、ドット `.` をしてそのフィールドにアクセスできます。しかし、 `struct`へのポインタがあれば、これはしません。フィールドにアクセスするには、 `->` をするがあります。ここでは、 `struct`のポインタをしてをすスタックのになたぶん "ろしいな"のをします。

```
#include <stdlib.h>
#include <stdio.h>

/* structs */
struct stack
{
    struct node *top;
    int size;
};

struct node
{
    int data;
    struct node *next;
};

/* function declarations */
int push(int, struct stack*);
int pop(struct stack*);
void destroy(struct stack*);

int main(void)
{
    int result = EXIT_SUCCESS;

    size_t i;

    /* allocate memory for a struct stack and record its pointer */
    struct stack *stack = malloc(sizeof *stack);
    if (NULL == stack)
    {
        perror("malloc() failed");
        return EXIT_FAILURE;
    }

    /* initialize stack */
    stack->top = NULL;
    stack->size = 0;

    /* push 10 ints */
    {
```

```
        int data = 0;
        for(i = 0; i < 10; i++)
        {
            printf("Pushing: %d\n", data);
            if (-1 == push(data, stack))
            {
                perror("push() failed");
                result = EXIT_FAILURE;
                break;
            }

            ++data;
        }
    }

    if (EXIT_SUCCESS == result)
    {
        /* pop 5 ints */
        for(i = 0; i < 5; i++)
        {
            printf("Popped: %i\n", pop(stack));
        }
    }

    /* destroy stack */
    destroy(stack);

    return result;
}

/* Push a value onto the stack. */
/* Returns 0 on success and -1 on failure. */
int push(int data, struct stack *stack)
{
    int result = 0;

    /* allocate memory for new node */
    struct node *new_node = malloc(sizeof *new_node);
    if (NULL == new_node)
    {
        result = -1;
    }
    else
    {
        new_node->data = data;
        new_node->next = stack->top;
        stack->top = new_node;
        stack->size++;
    }

    return result;
}

/* Pop a value off of the stack. */
/* Returns the value popped off the stack */
int pop(struct stack *stack)
{
    struct node *top = stack->top;
    int data = top->data;
    stack->top = top->next;
    stack->size--;
```

```
    free(top);
    return data;
}

/* destroy the stack */
void destroy(struct stack *stack)
{
    /* free all pointers */
    while(stack->top != NULL)
    {
        pop(stack);
    }
}
```

なメンバ―

C99

なくとも1つのをするは、のにのさののをさらにむことができる。これはフレキシブルなメンバ―とばれます

```
struct ex1
{
    size_t foo;
    int flex[];
};

struct ex2_header
{
    int foo;
    char bar;
};

struct ex2
{
    struct ex2_header hdr;
    int flex[];
};

/* Merged ex2_header and ex2 structures. */
struct ex3
{
    int foo;
    char bar;
    int flex[];
};
```

サイズとパディングにする

フレキシブルなメンバは、のサイズをするときにサイズをたないものとしてわれますが、そのメンバとののメンバとののパディングはまだするかもしれません

```
/* Prints "8,8" on my machine, so there is no padding. */
printf("%zu,%zu\n", sizeof(size_t), sizeof(struct ex1));
```

```
/* Also prints "8,8" on my machine, so there is no padding in the ex2 structure itself. */
printf("%zu,%zu\n", sizeof(struct ex2_header), sizeof(struct ex2));

/* Prints "5,8" on my machine, so there are 3 bytes of padding. */
printf("%zu,%zu\n", sizeof(int) + sizeof(char), sizeof(struct ex3));
```

フレキシブルメンバ―はなであるとみなされるため、 `sizeof`をしてそのサイズをすることはできません。

フレキシブルなメンバ―をむタイプのオブジェクトをしてすることはできますが、フレキシブルなメンバ―がしないかのようにされるため、フレキシブルなメンバ―をしないでください。これをうことはじられ、コンパイルエラ―がします。

に、このでをするときは、フレキシブルメンバののにをしてはいけません。しかし、コンパイラはずしもこれをうことをげません。そのため、のにつながるがあります。

```
/* invalid: cannot initialize flexible array member */
struct ex1 e1 = {1, {2, 3}};
/* invalid: hdr={foo=1, bar=2} OK, but cannot initialize flexible array member */
struct ex2 e2 = {{1, 2}, {3}};
/* valid: initialize foo=1, bar=2 members */
struct ex3 e3 = {1, 2};

e1.flex[0] = 3; /* undefined behavior, in my case */
e3.flex[0] = 2; /* undefined behavior again */
e2.flex[0] = e3.flex[0]; /* undefined behavior */
```

わりに、 `malloc`、 `calloc`、 `realloc`いずれかをしてをなストレ―ジにりて、でることもできます。これにより、なメンバ―をにできます。

```
/* valid: allocate an object of structure type `ex1` along with an array of 2 ints */
struct ex1 *pe1 = malloc(sizeof(*pe1) + 2 * sizeof(pe1->flex[0]));

/* valid: allocate an object of structure type ex2 along with an array of 4 ints */
struct ex2 *pe2 = malloc(sizeof(struct ex2) + sizeof(int[4]));

/* valid: allocate 5 structure type ex3 objects along with an array of 3 ints per object */
struct ex3 *pe3 = malloc(5 * (sizeof(*pe3) + sizeof(int[3])));

pe1->flex[0] = 3; /* valid */
pe3[0]->flex[0] = pe1->flex[0]; /* valid */
```

C99

'struct hack'は、

フレキシブルアレイメンバ―は、C99はせず、エラ―としてわれていました。なは、 'struct hack' とばれるさ1のをすることです。

```
struct ex1
{
```

```
    size_t foo;
    int flex[1];
};
```

しかし、これはのフレキシブルなメンバとはなり、のサイズにします

```
/* Prints "8,4,16" on my machine, signifying that there are 4 bytes of padding. */
printf("%d,%d,%d\n", (int)sizeof(size_t), (int)sizeof(int[1]), (int)sizeof(struct ex1));
```

するには`flex`フレキシブルなメンバーとしてメンバーを、あなたがそれをりてたい`malloc`ことをいて、のように`sizeof(*pe1)`または`sizeof(struct ex1)` にきえられる`offsetof(struct ex1, flex)` `sizeof(*pe1)-sizeof(pe1->flex)`いずれかであるかどうかを`sizeof(*pe1)-sizeof(pe1->flex)`。 あるいは、のさが0よりきいとして、サイズににまれているので、 "な"ののさから1をくことができます。のにもじロジックをできます。

フレキシブルなメンバーをサポートしていないコンパイラとのがなは、の`FLEXMEMB_SIZE`ようなマクロをすることができます

```
#if __STDC_VERSION__ < 199901L
#define FLEXMEMB_SIZE 1
#else
#define FLEXMEMB_SIZE /* nothing */
#endif

struct ex1
{
    size_t foo;
    int flex[FLEXMEMB_SIZE];
};
```

オブジェクトをりてるときには、のサイズフレキシブルメンバをく`offsetof(struct ex1, flex)`をするには`offsetof(struct ex1, flex)`フォームをするがあります。これは、フレキシブルなメンバをサポートするコンパイラと、ない

```
struct ex1 *pe10 = malloc(offsetof(struct ex1, flex) + n * sizeof(pe10->flex[0]));
```

は、プリプロセッサをして、されたさから1をきですることです。こののとなエラーのがまったため、はロジックをのにしました。

```
struct ex1 *ex1_alloc(size_t n)
{
    struct ex1 tmp;
#if __STDC_VERSION__ < 199901L
    if (n != 0)
        n--;
#endif
    return malloc(sizeof(tmp) + n * sizeof(tmp.flex[0]));
}
...

/* allocate an ex1 object with "flex" array of length 3 */
```

```
struct ex1 *pe1 = ex1_alloc(3);
```

をにす

Cでは、すべてのはstructをむによってにされます。さなの、これはポインタをしてデ―タにアクセスすることによるオ―バヘッドがないことをするので、いことです。しかし、プログラマがをしするのにれているは、なをってしてパフォ―マンスがすることもににになります。

```
struct coordinates
{
    int x;
    int y;
    int z;
};

// Passing and returning a small struct by value, very fast
struct coordinates move(struct coordinates position, struct coordinates movement)
{
    position.x += movement.x;
    position.y += movement.y;
    position.z += movement.z;
    return position;
}

// A very big struct
struct lotsOfData
{
    int param1;
    char param2[80000];
};

// Passing and returning a large struct by value, very slow!
// Given the large size of the struct this could even cause stack overflow
struct lotsOfData doubleParam1(struct lotsOfData value)
{
    value.param1 *= 2;
    return value;
}

// Passing the large struct by pointer instead, fairly fast
void doubleParam1ByPtr(struct lotsOfData *value)
{
    value->param1 *= 2;
}
```

をしたオブジェクトベ―スのプログラミング

は、オブジェクトのでコ―ドをするためにできます。はクラスにていますが、はクラスのでもあるがけています。これらをポインタのメンバとしてできます。たちののにとどまるには

```
/* coordinates.h */

typedef struct coordinate_s
{
    /* Pointers to method functions */
```

```
    void (*setx)(coordinate *this, int x);
    void (*sety)(coordinate *this, int y);
    void (*print)(coordinate *this);
    /* Data */
    int x;
    int y;
} coordinate;

/* Constructor */
coordinate *coordinate_create(void);
/* Destructor */
void coordinate_destroy(coordinate *this);
```

そして、Cファイルをしています

```
/* coordinates.c */

#include "coordinates.h"
#include <stdio.h>
#include <stdlib.h>

/* Constructor */
coordinate *coordinate_create(void)
{
    coordinate *c = malloc(sizeof(*c));
    if (c != 0)
    {
        c->setx = &coordinate_setx;
        c->sety = &coordinate_sety;
        c->print = &coordinate_print;
        c->x = 0;
        c->y = 0;
    }
    return c;
}

/* Destructor */
void coordinate_destroy(coordinate *this)
{
    if (this != NULL)
    {
        free(this);
    }
}

/* Methods */
static void coordinate_setx(coordinate *this, int x)
{
    if (this != NULL)
    {
        this->x = x;
    }
}

static void coordinate_sety(coordinate *this, int y)
{
    if (this != NULL)
    {
        this->y = y;
    }
```

```
}

static void coordinate_print(coordinate *this)
{
    if (this != NULL)
    {
        printf("Coordinate: (%i, %i)\n", this->x, this->y);
    }
    else
    {
        printf("NULL pointer exception!\n");
    }
}
```

クラスのはのとおりです。

```
/* main.c */

#include "coordinates.h"
#include <stddef.h>

int main(void)
{
    /* Create and initialize pointers to coordinate objects */
    coordinate *c1 = coordinate_create();
    coordinate *c2 = coordinate_create();

    /* Now we can use our objects using our methods and passing the object as parameter */
    c1->setx(c1, 1);
    c1->sety(c1, 2);

    c2->setx(c2, 3);
    c2->sety(c2, 4);

    c1->print(c1);
    c2->print(c2);

    /* After using our objects we destroy them using our "destructor" function */
    coordinate_destroy(c1);
    c1 = NULL;
    coordinate_destroy(c2);
    c2 = NULL;

    return 0;
}
```

オンラインでをむ https://riptutorial.com/ja/c/topic/1119/

52: のパディングとパッキング

き

デフォルトでは、Cコンパイラはをレイアウトして、メンバーに、アライメントのないアクセス、DEC AlphaやのARM CPUなどのRISCマシンのにするペナルティをすことなく、アクセスがです。

CPUアーキテクチャーとコンパイラーによって、は、そのメンバーのサイズのよりもくのメモリーをするがあります。コンパイラは、メンバーまたはのにパディングをできますが、はしません。

パッキングはデフォルトのパディングをオーバーライドします。

エリックレイモンドは、 Cパッキングのロストアートにするをんでいます。

Examples

パッキング

デフォルトでは、はCでめられます。このをけたいは、にするがあります。 GCCでは `__attribute__((__packed__))` です。 64ビットマシンでこのをえてみましょう。

```
struct foo {
    char *p;  /* 8 bytes */
    char c;   /* 1 byte  */
    long x;   /* 8 bytes */
};
```

はに`8-byte`になるようにパディングされ、のようになります。

```
struct foo {
    char *p;     /* 8 bytes */
    char c;      /* 1 byte  */

    char pad[7]; /* 7 bytes added by compiler */

    long x;      /* 8 bytes */
};
```

`sizeof(struct foo)`は`17`わりに`24` `sizeof(struct foo)`ます。これは、64ビットコンパイラがステップで8バイトのワードでメモリからみきするためにし、 `char c;`をきもうとするとらか`char c;`メモリの1バイトはな8バイトすなわちワードがフェッチされ、そののバイトのみをし、その7つのしたバイトはのままであり、パディングのためのしおよびみにしてアクセスできない。

パッキング

しかし、 `packed`をすると、コンパイラはパディングをしません。

```
struct __attribute__((__packed__)) foo {
    char *p;  /* 8 bytes */
    char c;   /* 1 byte  */
    long x;   /* 8 bytes */
};
```

`sizeof(struct foo)`は`17`をします。

に、パックがされます

- スペースをする。
- ネットワークのノードのアーキテクチャのアラインメントにすることなく、ネットワークをしてするようにデータをフォーマットすること。

ARM Cortex-M0などののプロセッサでは、アラインされていないメモリアクセスがされていないことをしなければなりません。そのような、パッキングはのにつながり、CPUをクラッシュさせるがあります。

のパディング

この`struct`がされ、32ビットコンパイラでコンパイルされたとします。

```
struct test_32 {
    int a;      // 4 byte
    short b;    // 2 byte
    int c;      // 4 byte
} str_32;
```

この`struct`はメモリの10バイトしかしないとされるかもしれませんが、 `sizeof(str_32)`をすると12バイトがされます。

これは、アクセスのためにコンパイラがをさせるためにこりました。なパターンは、がNバイトNは1,2,4,8,16などの2ので、それはそれほどきくないをめる、はNバイトのにするがありますNバイトの。

`sizeof(int) == 4`および`sizeof(short) == 2`でされたの、なレイアウトはのようになります。

- `int a;`オフセット0にされます。サイズ4。
- `short b;`オフセット4にされる。サイズ2。
- オフセット6ののないパディング。サイズ2。
- `int c;`オフセット8でされる。サイズ4。

したがって、 `struct test_32`は12バイトのメモリをします。このでは、のパディングはありませ

ん。

コンパイラは、 `struct test_32`のメンバーがアクセスのためににされるように、4バイトからまる`struct test_32`がにされるようにします。 `malloc()` 、 `calloc()` 、 `realloc()`などのメモリりては、されるポインタがどのデータでもにされ、にりてられたもしくされるようにするためにです。

32ビットモードでコンパイルすると、コンパイラが`double`にされた64ビットIntel x86_64プロセッサIntel Core i7 - MacOS SierraまたはMac OS Xをしているなどのようななにわることがあります。 4バイトの。じハードウェアでは、64ビット・モードでコンパイルすると、コンパイラは8バイトのに`double`されます。

オンラインでのパディングとパッキングをむ https://riptutorial.com/ja/c/topic/4590/のパディングとパッキング

53: な

- #include <math.h>
- ダブルパワ―double x、double y;
- float powffloat x、float y;
- long double powllong double x、long double y;

1. ライブラリとリンクするには、gccフラグをして`-lm`をします。
2. からエラ―をチェックするがあるなプログラムは、 `errno`を0に`errno`し、のようにびします `feclearexcept(FE_ALL_EXCEPT);`をびすに。 からのに、 `errno`がゼロの、またはのびしは0の `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW);`します `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW);`でエラ―がしました。は、math_errorのマンペ―ジをしてください。

Examples

fmod

このは、 `x/y`のをします。りはxとじをちます。

```
#include <math.h> /* for fmod() */
#include <stdio.h> /* for printf() */

int main(void)
{
    double x = 10.0;
    double y = 5.1;

    double modulus = fmod(x, y);

    printf("%lf\n", modulus); /* f is the same as lf. */

    return 0;
}
```

```
4.90000
```

このは、ののためにしないをすがあるため、してしてください。

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    printf("%f\n", fmod(1, 0.1));
    printf("%19.17f\n", fmod(1, 0.1));
    return 0;
}
```

```
0.1
0.09999999999999995
```

および**long**の**fmodf**、fmodl

C99

これらのは、 x/yのをします。りはxとじをちます。

```
#include <math.h> /* for fmodf() */
#include <stdio.h> /* for printf() */

int main(void)
{
    float x = 10.0;
    float y = 5.1;

    float modulus = fmodf(x, y);

    printf("%f\n", modulus); /* lf would do as well as modulus gets promoted to double. */
}
```

```
4.90000
```

ダブルダブル

```
#include <math.h> /* for fmodl() */
#include <stdio.h> /* for printf() */

int main(void)
{
    long double x = 10.0;
    long double y = 5.1;

    long double modulus = fmodl(x, y);

    printf("%Lf\n", modulus); /* Lf is for long double. */
}
```

```
4.90000
```

パワー**- pow**、powf、powl

のサンプルコードは、のライブラリのpowファミリをして、 *1 + 43 + 3 ^ 2 + 3 ^ 3 + 3 ^ 4 + ... + 3 ^ N*シリーズのをします。

```
#include <stdio.h>
#include <math.h>
#include <errno.h>
#include <fenv.h>

int main()
```

```
{
        double pwr, sum=0;
        int i, n;

        printf("\n1+4(3+3^2+3^3+3^4+...+3^N)=?\nEnter N:");
        scanf("%d",&n);
        if (n<=0) {
                printf("Invalid power N=%d", n);
                return -1;
        }

        for (i=0; i<n+1; i++) {
                errno = 0;
                feclearexcept(FE_ALL_EXCEPT);
                pwr = powl(3,i);
                if (fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW |
                        FE_UNDERFLOW)) {
                        perror("Math Error");
                }
                sum += i ? pwr : 0;
                printf("N= %d\tS= %g\n", i, 1+4*sum);
        }

        return 0;
}
```

```
1+4(3+3^2+3^3+3^4+...+3^N)=?
Enter N:10
N= 0    S= 1
N= 1    S= 13
N= 2    S= 49
N= 3    S= 157
N= 4    S= 481
N= 5    S= 1453
N= 6    S= 4369
N= 7    S= 13117
N= 8    S= 39361
N= 9    S= 118093
N= 10    S= 354289
```

オンラインでなをむ https://riptutorial.com/ja/c/topic/3170/な

54:

き

プログラミングのは、コンパイラまたはインタプリタに、の、またはをし、をするようにするシンボルです。

Cにはくのながあります。くのCは2です。つまり、2つのオペランドをちます。たとえば、 `a / b` では、 `/`は2つのオペランド `a` 、 `b` をけれるバイナリです。 1つのオペランドたとえば、 `~` 、 `++` をると、1つのしかないものがあり `? :` 。

- expr1
- expr2
- expr1expr2
- expr1 expr2expr3

は、 アリティ 、 、およびをっています 。

- *Arity*はオペランドのをします。 Cでは、3つのなるがします。
  - 1オペランド
  - バイナリ2オペランド
  - 33オペランド
- は、どのオペレ―タがにオペランドに「バインド」するかをします。すなわち、どのオペレ―タがそのオペランドをするをするかである。えば、Cは、とがとよりもされるというにいます。

  ```
  a * b + c
  ```

  じをえる

  ```
  (a * b) + c
  ```

  これがしかったものでない、すべてののでもいをつため、カッコをしてをすることができます。

  ```
  a * (b + c)
  ```

  このしいは、の2つのとはなるをします。

  Cにはくのレベルがあります。のは、すべてののののでしています。

`() [] -> .`	からへ
`! ~ ++ -- + - * (type) sizeof`	からへ
`* / %`	からへ
`+ -`	からへ
`<< >>`	からへ
`< <= > >=`	からへ
`== !=`	からへ
`&`	からへ
`^`	からへ
`|`	からへ
`&&`	からへ
`||`	からへ
`?:`	からへ
`= += -= *= /= %= &= ^= |= <<= >>=`	からへ
`,`	からへ

- *Associativity*は、のしいがデフォルトでどのようにバインドするかをし、 から 、 からの 2 があります。 からへのバインディングのは、 _ です。

```
a - b - c - d
```

3つののがありますが、

```
((a - b) - c) - d
```

のために_その2つのオペランドににバインドします。

からへののは、*および++です。どちらもじをっているので、

```
* ptr ++
```

、これは

```
* (ptr ++)
```

の ++ がにそのオペランドにバインドするためです。

Examples

は、2つのオペランドののがであるかどうかをチェックします。は1 をするまたは0 をするとされる。このは、 if 、 while 、 forしてフロ―にをえるためによくされますが、にすることもできます。

"=="

されたオペランドがしいかどうかをチェックします。

```
1 == 0;         /* evaluates to 0. */
1 == 1;         /* evaluates to 1. */

int x = 5;
int y = 5;
int *xptr = &x, *yptr = &y;
xptr == yptr;   /* evaluates to 0, the operands hold different location addresses. */
*xptr == *yptr; /* evaluates to 1, the operands point at locations that hold the same value.
*/
```

このを = としないでください

"="としくない

されたオペランドがしくないかどうかをチェックします。

```
1 != 0;         /* evaluates to 1. */
1 != 1;         /* evaluates to 0. */

int x = 5;
int y = 5;
int *xptr = &x, *yptr = &y;
xptr != yptr;   /* evaluates to 1, the operands hold different location addresses. */
*xptr != *yptr; /* evaluates to 0, the operands point at locations that hold the same value.
*/
```

このは、 == のとにのをします。

"ではない"

オブジェクトが0かどうかをしてください。

!のようににすることもできます。

```
!someVal
```

これとじがあります

```
someVal == 0
```

">"よりきい

オペランドがオペランドよりきいをつかどうかをチェックします。

```
5 > 4      /* evaluates to 1. */
4 > 5      /* evaluates to 0. */
4 > 4      /* evaluates to 0. */
```

"<"よりさい

オペランドのがオペランドのよりもさいかどうかをチェックします。

```
5 < 4      /* evaluates to 0. */
4 < 5      /* evaluates to 1. */
4 < 4      /* evaluates to 0. */
```

よりきいまたはしい "> ="

オペランドのがオペランドのよりきいかしいかをチェックします。

```
5 >= 4      /* evaluates to 1. */
4 >= 5      /* evaluates to 0. */
4 >= 4      /* evaluates to 1. */
```

よりさいかしい "<="

オペランドのがオペランドのよりさいかしいかをチェックします。

```
5 <= 4      /* evaluates to 0. */
4 <= 5      /* evaluates to 1. */
4 <= 4      /* evaluates to 1. */
```

オペランドのをオペランドにされたにりて、そのをします。

```
int x = 5;      /* Variable x holds the value 5. Returns 5. */
char y = 'c';   /* Variable y holds the value 99. Returns 99
                 * (as the character 'c' is represented in the ASCII table with 99).
                 */
float z = 1.5;  /* variable z holds the value 1.5. Returns 1.5. */
char const* s = "foo"; /* Variable s holds the address of the first character of the string
```

```
'foo'. */
```

いくつかのには、 があります。

```
a += b  /* equal to: a = a + b */
a -= b  /* equal to: a = a - b */
a *= b  /* equal to: a = a * b */
a /= b  /* equal to: a = a / b */
a %= b  /* equal to: a = a % b */
a &= b  /* equal to: a = a & b */
a |= b  /* equal to: a = a | b */
a ^= b  /* equal to: a = a ^ b */
a <<= b /* equal to: a = a << b */
a >>= b /* equal to: a = a >> b */
```

これらののりてのなの1つは、 `a` のが1だけされることです。たとえば、 `p`がポインタの

```
*p += 27;
```

`p`は1だけであるが、は2りす。

```
*p = *p + 27;
```

また、 `a = b`ようなのは、 *rvalue*としてられているものであることにもすべきである。したがって、りてにははにはのにりてることができるがあります。これにより、1つのステートメントでのをするためののりてのがになります。

このは、のやびしのにしてあるコードをする`if`またはループまたは`switch`のでできます。えば

```
char *buffer;
if ((buffer = malloc(1024)) != NULL)
{
    /* do something with buffer */
    free(buffer);
}
else
{
    /* report allocation failure */
}
```

このため、なバグにつながるがあるのタイプミスをけるためにをわなければなりません。

```
int a = 2;
/* ... */
if (a = 1)
    /* Delete all files on my hard drive */
```

`a = 1`はに`1`とされるため、 `if`ステートメントのはにですこののとしについてはこちらをしてください 。は、にすように、 `==` をすることをほぼにしていました。

```
int a = 2;
```

```
/* ... */
if (a == 1)
    /* Delete all files on my hard drive */
```

オペレ―ター―

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

これは、りてcするbす、 bにりてです、 。 aこれは、すべてのがしいをっているためです。つまり、ののがにされ、からにみます。

するをして、オペランドをオペランドにしたのをします。ののがされますつまり、とは、、、はありません。

+ は、2つのオペランドをにするためにされます。

```
#include <stdio.h>

int main(void)
{
    int a = 5;
    int b = 7;

    int c = a + b; /* c now holds the value 12 */

    printf("%d + %d = %d",a,b,c); /* will output "5 + 7 = 12" */

    return 0;
}
```

- は、1オペランドから2オペランドをするためにされます。

```
#include <stdio.h>

int main(void)
{
    int a = 10;
    int b = 7;

    int c = a - b; /* c now holds the value 3 */

    printf("%d - %d = %d",a,b,c); /* will output "10 - 7 = 3" */

    return 0;
}
```

* は、のオペランドをするためにされます。

```
#include <stdio.h>

int main(void)
{
```

```
    int a = 5;
    int b = 7;

    int c = a * b; /* c now holds the value 35 */

    printf("%d * %d = %d",a,b,c); /* will output "5 * 7 = 35" */

    return 0;
}
```

*としないでください。

/ は、1オペランドを2オペランドでします。ののオペランドがの、をし、りをしますをしてするためにモジュロ%をします。

オペランドの1つがの、はのになります。

```
#include <stdio.h>

int main (void)
{
    int a = 19 / 2 ; /* a holds value 9   */
    int b = 18 / 2 ; /* b holds value 9   */
    int c = 255 / 2; /* c holds value 127 */
    int d = 44 / 4 ; /* d holds value 11  */
    double e = 19 / 2.0 ; /* e holds value 9.5   */
    double f = 18.0 / 2 ; /* f holds value 9.0   */
    double g = 255 / 2.0; /* g holds value 127.5 */
    double h = 45.0 / 4 ; /* h holds value 11.25 */

    printf("19 / 2 = %d\n", a);    /* Will output "19 / 2 = 9"    */
    printf("18 / 2 = %d\n", b);    /* Will output "18 / 2 = 9"    */
    printf("255 / 2 = %d\n", c);   /* Will output "255 / 2 = 127" */
    printf("44 / 4 = %d\n", d);    /* Will output "44 / 4 = 11"   */
    printf("19 / 2.0 = %g\n", e);  /* Will output "19 / 2.0 = 9.5"    */
    printf("18.0 / 2 = %g\n", f);  /* Will output "18.0 / 2 = 9"      */
    printf("255 / 2.0 = %g\n", g); /* Will output "255 / 2.0 = 127.5" */
    printf("45.0 / 4 = %g\n", h);  /* Will output "45.0 / 4 = 11.25"  */

    return 0;
}
```

モジュロ

モジュロ % はオペランドのみをけり、1オペランドを2オペランドでしたのをするためにされます。

```
#include <stdio.h>

int main (void) {
    int a = 25 % 2;    /* a holds value 1  */
    int b = 24 % 2;    /* b holds value 0  */
    int c = 155 % 5;   /* c holds value 0  */
    int d = 49 % 25;   /* d holds value 24 */
```

```
    printf("25 % 2 = %d\n", a);     /* Will output "25 % 2 = 1"    */
    printf("24 % 2 = %d\n", b);     /* Will output "24 % 2 = 0"    */
    printf("155 % 5 = %d\n", c);    /* Will output "155 % 5 = 0"   */
    printf("49 % 25 = %d\n", d);    /* Will output "49 % 25 = 24"  */

    return 0;
}
```

インクリメント/デクリメント

インクリメント `a++` とデクリメント `a--` は、なしでそれらにするのをするでなります。インクリメントとデクリメントは、のにできます。オペレータのにより、の/のタイミングがにりてられるまたはにされます。

```
#include <stdio.h>

int main(void)
{
    int a = 1;
    int b = 4;
    int c = 1;
    int d = 4;

    a++;
    printf("a = %d\n",a);    /* Will output "a = 2" */
    b--;
    printf("b = %d\n",b);    /* Will output "b = 3" */

    if (++c > 1) { /* c is incremented by 1 before being compared in the condition */
        printf("This will print\n");    /* This is printed */
    } else {
        printf("This will never print\n");    /* This is not printed */
    }

    if (d-- < 4) {  /* d is decremented after being compared */
        printf("This will never print\n");    /* This is not printed */
    } else {
        printf("This will print\n");    /* This is printed */
    }
}
```

`c`と`d`がすように、のにはとの2つのがあります。ともをインクリメント `++` またはデクリメント `--` するのとじをちますが、りによってなります。プレフィックスではにをい、にをすのにし、ではにされ、にをいます。

このになるいのために、ののインクリメント/デクリメントのはのがあります。

AND

のオペランドがゼロの、1をす2つのオペランドのブールをします。ANDは`int`です。

```
0 && 0  /* Returns 0. */
0 && 1  /* Returns 0. */
```

```
2 && 0  /* Returns 0. */
2 && 3  /* Returns 1. */
```

OR

オペランドのいずれかがゼロの、1をす2つのオペランドのブールをします。ORは$_{int}$です。

```
0 || 0  /* Returns 0. */
0 || 1  /* Returns 1.  */
2 || 0  /* Returns 1.  */
2 || 3  /* Returns 1.  */
```

NOT

をします。NOTは$_{int}$です。 NOTは、なくとも1つのビットが1にしいかどうかをべ、0のは1をします。

```
!1 /* Returns 0. */
!5 /* Returns 0. */
!0 /* Returns 1. */
```

$_{\&\&}$と$_{||}$にするいくつかのながあります。

- オペランドRHSがにされるにオペランドLHSがにされ、
- オペランドとオペランドののにシーケンスポイントがあり、
- もなのは、のオペランドのがのをする、のオペランドはくされないということです。

このは

- LHSが「」ゼロとされた、$_{||}$されません「のORか」のが「」であるため
- LHSが 'false'ゼロとされた、$_{\&\&}$のRHSはされません 'false AND anything'のは 'false'なので。

これは、のようなコードをくことができるのでです。

```
const char *name_for_value(int value)
{
    static const char *names[] = { "zero", "one", "two", "three", };
    enum { NUM_NAMES = sizeof(names) / sizeof(names[0]) };
    return (value >= 0 && value < NUM_NAMES) ? names[value] : "infinity";
}
```

のがにされた、$_{value >= 0}$はとされ、$_{value < NUM_NAMES}$はされません。

インクリメント/デクリメント

およびはと のでします 。

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

tmp = ++a;         /* increments a by one, and returns new value; a == 2, tmp == 2  */
tmp = a++;         /* increments a by one, but returns old value; a == 3, tmp == 2 */
tmp = --b;         /* decrements b by one, and returns new value; b == 0, tmp == 0 */
tmp = b--;         /* decrements b by one, but returns old value; b == -1, tmp == 0 */
```

はシーケンスポイントをしないので、 `++`または`--`をするのはのをするがあることにしてください。

き/ 3

のオペランドをし、のがゼロでないは、2のオペランドをします。それのは、のにすように、3のオペランドをします。

```
a = b ? c : d;
```

のものとです。

```
if (b)
    a = c;
else
    a = d;
```

このコードはそれをし`condition ? value_if_true : value_if_false`。 は、されたのであるがあります。

```
int x = 5;
int y = 42;
printf("%i, %i\n", 1 ? x : y, 0 ? x : y); /* Outputs "5, 42" */
```

きはネストできます。たとえば、のコードは3つののうちきいをします。

```
big= a > b ? (a > c ? a : c)
           : (b > c ? b : c);
```

のでは、を1つのファイルに、のをのファイルにきみます。

```
#include<stdio.h>

int main()
{
    FILE *even, *odds;
    int n = 10;
    size_t k = 0;

    even = fopen("even.txt", "w");
    odds = fopen("odds.txt", "w");
```

```
    for(k = 1; k < n + 1; k++)
    {
        k%2==0 ? fprintf(even, "\t%5d\n", k)
               : fprintf(odds, "\t%5d\n", k);
    }
    fclose(even);
    fclose(odds);

    return 0;
}
```

は、からにけられます。のをしてください。

```
exp1 ? exp2 : exp3 ? exp4 : exp5
```

けがからにあるので、のはのようにされます。

```
exp1 ? exp2 : ( exp3 ? exp4 : exp5 )
```

コンマ

そののオペランドをし、のをし、にそのオペランドをし、はのオペランドのをもたらす。

```
int x = 42, y = 42;
printf("%i\n", (x *= 2, y)); /* Outputs "42". */
```

コンマは、そのオペランドのにシーケンスポイントをします。

々のをつびしでされるカンマは、 カンマではなく、 カンマとはなるセパレータとばれます 。したがって、 カンマのプロパティはありません。

の`printf()`コールには、 カンマとセパレータのがまれています 。

```
printf("%i\n", (x *= 2, y)); /* Outputs "42". */
/*           ^        ^ this is a comma operator */
/*           this is a separator */
```

コンマは、セクションと`for`ループのセクションでよくされます。えば

```
for(k = 1; k < 10; printf("\%d\\n", k), k += 2);   /*outputs the odd numbers below 9/*

/* outputs sum to first 9 natural numbers */
for(sumk = 1, k = 1; k < 10; k++, sumk += k)
    printf("\%5d\%5d\\n", k, sumk);
```

キャストオペレータ

されたをしたのから、されたへのなをします。

```
int x = 3;
int y = 4;
printf("%f\n", (double)x / y); /* Outputs "0.750000". */
```

ここで$_{x}$のは$_{double}$にされ、 $_{y}$のも$_{double}$にされ、の、 $_{double}$は$_{printf}$にされてされます。

のサイズ

をオペランドとしてする

されたのオブジェクトの$_{size_t}$のサイズをバイトでします。のりにかっこがです。

```
printf("%zu\n", sizeof(int)); /* Valid, outputs the size of an int object, which is platform-
dependent. */
printf("%zu\n", sizeof int); /* Invalid, types as arguments need to be surrounded by
parentheses! */
```

オペランドとしてのの

されたののオブジェクトの$_{size_t}$のサイズをバイトでします。はされません。カッコはではありません。ただし、えられたはでなければならないため、にすることをおめします。

```
char ch = 'a';
printf("%zu\n", sizeof(ch)); /* Valid, will output the size of a char object, which is always
1 for all platforms. */
printf("%zu\n", sizeof ch);  /* Valid, will output the size of a char object, which is always
1 for all platforms. */
```

ポインタ

ポインタの

えられたポインタとスカラ―$_{N}$は、しされたオブジェクトのにメモリでするさしの$_{N}$のへのポインタにされます。

```
int arr[] = {1, 2, 3, 4, 5};
printf("*(arr + 3) = %i\n", *(arr + 3)); /* Outputs "4", arr's fourth element. */
```

ポインタがオペランドまたはスカラ―としてされるかどうかはありません。これは、 $_{3 + arr}$などのものがであることをします。 $_{arr[k]}$がの$_{k+1}$メンバ―の、 $_{arr+k}$は$_{arr[k]}$へのポインタです。いえれば、 $_{arr}$または$_{arr+0}$は$_{arr[0]}$へのポインタであり、 $_{arr+1}$は$_{arr[2]}$へのポインタです。に、 $_{*(arr+k)}$は$_{arr[k]}$とじです。

のとはなり、 $_{int}$へのポインタに$_{1}$をえるとのアドレスに$_{4}$バイトがされます。はポインタなので、 $_{+}$はをってポインタでのメンバ―にアクセスするためにできるのです。しかし、へのポインタをすることで、のデ―タをよりにすることができます。たとえば、のようにのメンバ―をできます。

```
#include<stdio.h>
static const size_t N = 5

int main()
{
    size_t k = 0;
    int arr[] = {1, 2, 3, 4, 5};
    for(k = 0; k < N; k++)
    {
        printf("\n\t%d", *(arr + k));
    }
    return 0;
}
```

へのポインタをすることにより、のプログラムはのようになります

```
#include<stdio.h>
static const size_t N = 5

int main()
{
    size_t k = 0;
    int arr[] = {1, 2, 3, 4, 5};
    int *ptr = arr; /* or int *ptr = &arr[0]; */
    for(k = 0; k < N; k++)
    {
        printf("\n\t%d", ptr[k]);
        /* or   printf("\n\t%d", *(ptr + k)); */
        /* or   printf("\n\t%d", *ptr++); */
    }
    return 0;
}
```

`arr`メンバーには、`+`と`++`をしてアクセスすることができます。ポインタ`ptr`できるのは`-`と`--`です。

ポインタ

じへの2つのポインタがえられた、のポインタのをするために2のポインタにするがあるスカラーをする`ptrdiff_t`のオブジェクトをします。

```
int arr[] = {1, 2, 3, 4, 5};
int *p = &arr[2];
int *q = &arr[3];
ptrdiff_t diff = q - p;

printf("q - p = %ti\n", diff); /* Outputs "1". */
printf("*(p + (q - p)) = %d\n", *(p + diff)); /* Outputs "4". */
```

アクセス

メンバアクセスドット`.`は、`->`のメンバーにアクセスするためにされる`struct`。

オブジェクトのメンバー

アクセスされたオブジェクトのメンバであるオブジェクトをすをします。

```
struct MyStruct
{
    int x;
    int y;
};

struct MyStruct myObject;
myObject.x = 42;
myObject.y = 123;

printf(".x = %i, .y = %i\n", myObject.x, myObject.y); /* Outputs ".x = 42, .y = 123". */
```

pointed-toオブジェクトのメンバ―

のためのとそののメンバ―アクセス。には、 `x->y`というのは`(*x).y` - のですが、にポインタがネストされているは、がはっきりしています。

```
struct MyStruct
{
    int x;
    int y;
};

struct MyStruct myObject;
struct MyStruct *p = &myObject;

p->x = 42;
p->y = 123;

printf(".x = %i, .y = %i\n", p->x, p->y); /* Outputs ".x = 42, .y = 123". */
printf(".x = %i, .y = %i\n", myObject.x, myObject.y); /* Also outputs ".x = 42, .y = 123". */
```

`&`は、のアドレスです。えられたをします。のオブジェクトはでなければなりません。に、がのオブジェクトのへのポインタであるオブジェクトにされ、のオブジェクトのアドレスがされます。

```
int x = 3;
int *p = &x;
printf("%p = %p\n", (void *)&x, (void *)p); /* Outputs "A = A", for some implementation-
defined A. */
```

`*`、ポインタをデリファレンスします。えられたをしたられるポインタのからじるをします。

```
int x = 42;
int *p = &x;
printf("x = %d, *p = %d\n", x, *p); /* Outputs "x = 42, *p = 42". */

*p = 123;
printf("x = %d, *p = %d\n", x, *p); /* Outputs "x = 123, *p = 123". */
```

インデックス

けは、ポインタのとそれにくのなです。には`a[i]`というのは`*(a + i)`とじですが、ながされます。

```
int arr[] = { 1, 2, 3, 4, 5 };
printf("arr[2] = %i\n", arr[2]); /* Outputs "arr[2] = 3". */
```

インデックスの

へのポインタのはなですつまり、オペランドのはをしません。したがって、 `pointer + integer == integer + pointer`です。

その、 `arr[3]`と`3[arr]`はです。

```
printf("3[arr] = %i\n", 3[arr]); /* Outputs "3[arr] = 4". */
```

`3[arr]``arr[3]`わりに`3[arr]`はコードのにするため、にはおめできません。されたプログラミングコンテストではがいがあります。

びし

1オペランドはポインタでなければなりませんへのポインタにされるため、もされます。びしのをし、のすべてのオペランドがあるはそれらをしてびしの。それぞれのでなをびしたのりをします。

```
int myFunction(int x, int y)
{
    return x * 2 + y;
}

int (*fn)(int, int) = &myFunction;
int x = 42;
int y = 123;

printf("(*fn)(%i, %i) = %i\n", x, y, (*fn)(x, y)); /* Outputs "fn(42, 123) = 207". */
printf("fn(%i, %i) = %i\n", x, y, fn(x, y)); /* Another form: you don't need to dereference
explicitly */
```

ビット

ビットのをして、にしてビットレベルのをできます。
は、Cでサポートされている6つのビットのです。

シンボル	オペレーター
そして、	ビットAND

シンボル	オペレーター
|	ビットOR
^	ビットXOR
	ビットではない1の
<<	シフト
>>	シフト

のプログラムは、すべてのビットのをしています。

```
#include <stdio.h>

int main(void)
{
   unsigned int a = 29;    /* 29 = 0001 1101 */
   unsigned int b = 48;    /* 48 = 0011 0000 */
   int c = 0;

   c = a & b;              /* 32 = 0001 0000 */
   printf("%d & %d = %d\n", a, b, c );

   c = a | b;              /* 61 = 0011 1101 */
   printf("%d | %d = %d\n", a, b, c );

   c = a ^ b;              /* 45 = 0010 1101 */
   printf("%d ^ %d = %d\n", a, b, c );

   c = ~a;                 /* -30 = 1110 0010 */
   printf("~%d = %d\n", a, c );

   c = a << 2;             /* 116 = 0111 0100 */
   printf("%d << 2 = %d\n", a, c );

   c = a >> 2;             /* 7 = 0000 0111 */
   printf("%d >> 2 = %d\n", a, c );

   return 0;
}
```

このようなビットのビットはのをつため、きのをしたビットはけるがあります。シフトにはのがされます。

- 1ビットをきビットにシフトすることはりであり、のにつながる。
- のビット1をシフトすることはであるため、がありません。
- シフトのオペランドのがであるか、またはされたオペランドのである、そのはです。

マスキン グ

マスキングとは、ビットをしてからのビットをするまたは、のビットをするプロセスをす。マスキングのにされるオペランドまたはは、 マスクとばれます。

マスキングはさまざまなでされます。

- のビットパターンをする。
- えられたビットパターンのをしいにコピーし、しいのりのを0でめますビットのANDをします
- えられたビットパターンのをしいにコピーする、しいのりは1でめられますビットのORをします。
- えられたビットパターンのをしいにコピーし、のビットパターンのりのをしいでさせますビットのをします。

のは、マスクをしてのビットパターンをします。

```
#include <limits.h>
void bit_pattern(int u)
{
    int i, x, word;
    unsigned mask = 1;
    word = CHAR_BIT * sizeof(int);
    mask = mask << (word - 1);    /* shift 1 to the leftmost position */
    for(i = 1; i <= word; i++)
    {
        x = (u & mask) ? 1 : 0;  /* identify the bit */
        printf("%d", x);         /* print bit value */
        mask >>= 1;              /* shift mask to the right by 1 bit */
    }
}
```

_Alignof

C11

されたのをします。は、タイプの2つのオブジェクトをりてることができるバイトをす2ののである。 Cでは、は`size_t`でされ`size_t`。

は、でもでもないがあります。がとしてされている、のがされます。

このは、 `<stdalign.h>`なマクロの`alignof`によってアクセスされることがよくあります。

```
int main(void)
{
    printf("Alignment of char = %zu\n", alignof(char));
    printf("Alignment of max_align_t = %zu\n", alignof(max_align_t));
    printf("alignof(float[10]) = %zu\n", alignof(float[10]));
    printf("alignof(struct{char c; int n;}) = %zu\n",
            alignof(struct {char c; int n;}));
}
```

な

```
Alignment of char = 1
Alignment of max_align_t = 16
alignof(float[10]) = 4
alignof(struct{char c; int n;}) = 4
```

http://en.cppreference.com/w/c/language/_Alignof

の

は、なにif / while / ...のをスキップするです。 2つのオペランドののは、のオペランドがされまたは、がある&&をするとのオペランドがの、||をするとのオペランドはです、2のオペランドはされていない。

```
#include <stdio.h>

int main(void) {
  int a = 20;
  int b = -5;

  /* here 'b == -5' is not evaluated,
     since a 'a != 20' is false. */
  if (a != 20 && b == -5) {
    printf("I won't be printed!\n");
  }

  return 0;
}
```

あなたでそれをチェックしてください

```
#include <stdio.h>

int print(int i) {
  printf("print function %d\n", i);
  return i;
}

int main(void) {
  int a = 20;

  /* here 'print(a)' is not called,
     since a 'a != 20' is false. */
  if (a != 20 && print(a)) {
    printf("I won't be printed!\n");
  }

  /* here 'print(a)' is called,
     since a 'a == 20' is true. */
  if (a == 20 && print(a)) {
    printf("I will be printed!\n");
  }

  return 0;
}
```

```
$ ./a.out
print function 20
I will be printed!
```

は、コストがいのをけたいにです。さらに、こののようにプログラムのれにきくすることがあります なぜこのプログラムは "forked" 4

オンライインでをむ https://riptutorial.com/ja/c/topic/256/

55:

Examples

とのい

これは、メンバがメモリをし、メンバがメモリをしていないことをしています。

```
#include <stdio.h>
#include <string.h>

union My_Union
{
  int variable_1;
  int variable_2;
};

struct My_Struct
{
  int variable_1;
  int variable_2;
};

int main (void)
{
  union My_Union u;
  struct My_Struct s;
  u.variable_1 = 1;
  u.variable_2 = 2;
  s.variable_1 = 1;
  s.variable_2 = 2;
  printf ("u.variable_1: %i\n", u.variable_1);
  printf ("u.variable_2: %i\n", u.variable_2);
  printf ("s.variable_1: %i\n", s.variable_1);
  printf ("s.variable_2: %i\n", s.variable_2);
  printf ("sizeof (union My_Union): %i\n", sizeof (union My_Union));
  printf ("sizeof (struct My_Struct): %i\n", sizeof (struct My_Struct));
  return 0;
}
```

ユニオンをしてをする

いくつかのCでは、コードがunionの1つのメンバにきんだり、のからみんだりしてキャストをすることができますいのビットとしてしいをする。

しかし、これはCののまたはではされていませんが、のにつながることにしてください。コンパイラによってされるになはなくありませんしたがって、。

こののののとしては、IEEE 754ののにしてをするよりもくをする「」アルゴリズムがあります。このアルゴリズムは、ポインタキャスティングこれはにであり、なエイリアシングをるまたはunionこれはまだのですが、くのコンパイラでします

```
union floatToInt
{
    int32_t intMember;
    float floatMember; /* Float must be 32 bits IEEE 754 for this to work */
};

float inverseSquareRoot(float input)
{
    union floatToInt x;
    int32_t i;
    float f;
    x.floatMember = input;      /* Assign to the float member */
    i = x.intMember;            /* Read back from the integer member */
    i = 0x5f3759df - (i >> 1);
    x.intMember = i;            /* Assign to the integer member */
    f = x.floatMember;          /* Read back from the float member */
    f = f * (1.5f - input * 0.5f * f * f);
    return f * (1.5f - input * 0.5f * f * f);
}
```

このは、のコンピュータグラフィックスやゲームでは、のにべてスピードがかったためくされていましたが、であり、をい、スピードときえににポータブルではありませんでした。

1のにきみ、のからむ

のメンバーは、じをします。つまり、あるメンバーへのきみはのすべてのメンバーのデータをきし、あるメンバーからのみりはのすべてのメンバーからのみりとじデータになります。しかし、ユニオンメンバーはなるタイプとサイズをつことができるため、みまれるデータはなるがです。
http://www.riptutorial.com/c/example/9399/using-unions-to-reinterpret-values

のなは、じタイプの2つのメンバーをつをしています。これは、メンバーへのきみをすることをしている$_{m_1}$メンバーからみされきまれたに$_{m_2}$とメンバーへのきみ$_{m_2}$メンバーからみされきまれたに$_{m_1}$。

```
#include <stdio.h>

union my_union /* Define union */
{
    int m_1;
    int m_2;
};

int main (void)
{
    union my_union u;              /* Declare union */
    u.m_1 = 1;                     /* Write to m_1 */
    printf("u.m_2: %i\n", u.m_2); /* Read from m_2 */
    u.m_2 = 2;                     /* Write to m_2 */
    printf("u.m_1: %i\n", u.m_1); /* Read from m_1 */
    return 0;
}
```

```
u.m_2: 1
u.m_1: 2
```

オンラインでをむ https://riptutorial.com/ja/c/topic/7645/

56:

き

Cは、にコンパイルされたですするのではなく。 Cはフェーズをし、それをするはプログラムイメージまたはコンパイルされたプログラムです。 c11では、フェーズが5.1.1.2にリストされています。

ファイルの	
.c	ソースファイル。、とコードがまれています。
.h	ヘッダーファイル。、がまれます。
.o	オブジェクトファイル。でコンパイルされたコード。
.obj	オブジェクトファイルの。
.a	ライブラリファイル。オブジェクトファイルのパッケージ。
.dll	Windowsのダイナミックリンクライブラリ
.so	くのUnixライクなシステムのオブジェクトライブラリ。
.dylib	OSXのダイナミックリンクライブラリUnix
.exe、.com	Windowsファイル。オブジェクトファイルとライブラリファイルをリンクすることでされます。 Unixライクなシステムでは、ファイルのなファイルはありません。

POSIX c99コンパイラフラグ	
-o filename	ファイル。 bin/program.exe、 program
-I directory	direrctoryヘッダーをします。
-D name	マクロnameする
-L directory	directoryのライブラリをします。
-l name	リンクライブラリlibname。

POSIXプラットフォームLinux、メインフレーム、Macのコンパイラは、 c99とばれていなくても、これらのオプションをけれます。

- c99 - Cプログラムのコンパイル

GCCGNUコンパイラコレクションフラグ	
`-Wall`	にけれられているすべてのメッセージをにします。
`-Wextra`	よりくのメッセージをにし、あまりにも々しいことがあります。
`-pedantic`	コードがされたにしているにをす。
`-Wconversion`	のでをにしてください。してしてください。
`-c`	リンクせずにソースファイルをコンパイルします。
`-v`	コンパイルをします。

- `gcc`はPOSIXフラグにえてくのものをけれます。
- POSIXプラットフォーム `clang` 、ベンダのコンパイラののくのコンパイラでも、のフラグがされます。
- さらにくのオプションについては、 GCCのびしをしてください。

TCCTiny Cコンパイラフラグ	
`-Wimplicit-function-declaration`	のについてします。
`-Wunsupported`	TCCによってされるサポートされていないGCCについてします。
`-Wwrite-strings`	をchar *のわりにconst char *にする。
`-Werror`	がされたは、コンパイルをしてください。
`-Wall`	`-Werror` 、 `-Wunusupported` 、および`-Wwrite strings`のすべてのを`-Wwrite strings`。

Examples

リンカ―

リンカのは、のオブジェクトファイル `.o`ファイルをバイナリファイルにリンクすることです。 リンクのプロセスは、にアドレスをアドレスにすることをむ。リンクのは、プログラムです。

オブジェクトファイルをリンクに、リンカは、コマンドラインでされたすべてのオブジェクトモジュールをピックアップするに、いくつかのシステムのスタートアップコードをして、のオブジェクトファイルのをつオブジェクトモジュールのすべてのをしようとしますコマンドラインですることも、ライブラリからにすることもできます。に、オブジェクトファイルのロードアドレスをりてます。つまり、コードとデータがしたプログラムのアドレスにどこにされるのかをします。ロード・アドレスをすると、オブジェクト・コードのすべてのシンボリック・アドレスを、ターゲットのアドレスの「の」アドレスにきえることができます。プログラムはすぐにです。

これには、コンパイラがソースコードファイルからしたオブジェクトファイルと、にコンパイルしてライブラリファイルにしたオブジェクトファイルのがまれます。これらのファイルのは`.a`または`.so`でわり、リンカはほとんどのがわかっていて、にじてにリンクするため、はそれらについてるはありません。

リンカーのなびし

プリプロセッサとに、リンカはのプログラムで、しばしば`ld`とばれますただし、Linuxは`collect2`します。また、プリプロセッサのように、コンパイラをするときににリンカがびされます。したがって、リンカーをするのはのとおりです。

```
% gcc foo.o bar.o baz.o -o myprog
```

このは、3つのオブジェクトファイルにリンクするためのコンパイラし`foo.o`、`bar.o`、そして`baz.o`バイナリファイルのに`myprog`。これで`myprog`とばれるファイルをすることができ、うまくいけばクールでな`myprog`ができます。

なリンカのびし

リンカーをびすことはですが、これはおめできませんが、はプラットフォームのものです。つまり、Linuxでするオプションは、Solaris、AIX、macOS、Windowsなどのプラットフォームではしません。 GCCをってするは、`gcc -v`をして、あなたのためにされているものをることができます。

リンカのオプション

リンカは、そのをするためにいくつかのもとります。のコマンドはgccに`foo.o`と`bar.o`をリンクするようしますが、`ncurses`ライブラリもみます。

```
% gcc foo.o bar.o -o foo -lncurses
```

これはにはかれなかれです

```
% gcc foo.o bar.o /usr/lib/libncurses.so -o foo
```

が、`libncurses.so`があり`libncurses.a`でしたばかりのアーカイブで、`ar`。オブジェクトファイル

の`-lname` 、ライブラリをパスまたは`-lname`オプションのどちらかでリストするがあることにしてください。ライブラリでは、されたがです。ライブラリでは、はありません。

くのシステムで、 `<math.h>`からをしている、ライブラリをみむには`-lm`をするがありますが、Mac OS XとmacOS Sierraではこれをとしません。 LinuxやのUnixシステムではのライブラリがありますが、macOSではないライブラリがあります - POSIXスレッドとPOSIXリアルタイムとネットワ―キングライブラリはです。その、リンクプロセスはプラットフォ―ムによってなります。

そののコンパイルオプション

これはあなたのCプログラムのコンパイルをするためにるがあることです。に、 `-Wall`コマンドラインオプションをすることをおめします。

```
% gcc -Wall -c foo.cc
```

`-Wall`オプションをすると、コンパイラはでわしいコ―ドについてし、ににくのバグをつけられるようになります。

コンパイラがされているがわれていないやをすことをれるなどのをむコンパイラにをげたいは、`-Wall`というにもかかわらず、このオプションのセットをうことができます。 すべてののあるについて

```
% gcc -Wall -Wextra -Wfloat-equal -Wundef -Wcast-align -Wwrite-strings -Wlogical-op \
>     -Wmissing-declarations -Wredundant-decls -Wshadow …
```

その`clang`オプションであり`-Weverything`にのすべてのをオンにしない`clang`。

ファイルタイプ

Cプログラムをコンパイルするには、5のファイルをうがあります。

1. ソ―スファイル これらのファイルにはがまれており、により`.c`でわるがけられています。 `.cc`と`.cpp`はC ++ファイルです。 Cファイルではありません 。
   `foo.c`
2. ヘッダ―ファイル これらのファイルには、プロトタイプとさまざまなプリプロセッサステ―トメントがまれています。これらは、ソ―スコ―ドファイルがでされたにアクセスできるようにするためにされます。ヘッダ―ファイルはにより`.h`わります。
   `foo.h`
3. オブジェクトファイル これらのファイルは、コンパイラのとしてされます。バイナリのでされていますが、それらはではできません。オブジェクトファイルは、いくつかのオペレ―ティングシステムWindows、MS-DOSなどでは`.obj`わることがありますが、は`.o`わります。
   `foo.o foo.obj`
4. バイナリファイル これらは "リンカ"とばれるプログラムのとしてされます。リンカ―は、

のオブジェクトファイルをリンクして、なバイナリファイルをします。バイナリファイルには、UNIXオペレーティングシステムではなはありませんが、にWindowsでは`.exe`でわります。

```
foo foo.exe
```

5. ライブラリ ライブラリはコンパイルされたバイナリですが、それがファイルではありませんつまり、ライブラリに`main()`がありません。ライブラリには、のプログラムでできるがまれています。ライブラリには、ライブラリのすべてののプロトタイプをむヘッダファイルがしていなければなりません。これらのヘッダファイルは、ライブラリをするソースファイルでされるべきですえば、 `#include <library.h>`。 プログラムをにコンパイルできるようにリンカーをライブラリにするがあります。ライブラリには、との2があります。

   - ライブラリ スタティックライブラリ `.lib`をするDLLインポートライブラリファイルとしないように、POSIXシステムの`.a`ファイルとWindowsの`.lib`ファイルは、プログラムににみまれています。ライブラリーには、ライブラリーのどのバージョンがされているかにプログラムがっているというがあります。、されるすべてのライブラリがまれるため、ファイルのサイズはきくなります。

     ```
     libfoo.a foo.lib
     ```

   - ライブラリ ライブラリほとんどのPOSIXシステムの`.so`ファイル、OSXの`.dylib`、Windowsの`.dll`ファイルは、プログラムによってににリンクされます。 1つのライブラリイメージをくのプログラムでできるため、これらはライブラリとばれることもあります。ダイナミックライブラリには、のアプリケーションがライブラリをしているになディスクがなくてみます。また、ファイルをすることなく、ライブラリのバグがです。

     ```
     foo.so foo.dylib foo.dll
     ```

プリプロセッサ

Cコンパイラがソースコードファイルのコンパイルをするに、ファイルはフェーズでされます。このフェーズは、のプログラムでうことも、1つのファイルににすることもできます。いずれにしても、コンパイルがしくされるに、コンパイラによってにびされます。では、テキストをして、ソースコードをのソースコードまたはにします。あなたはそれを「された」または「された」ソースコードとえることができます。そのされたソースは、ファイルシステムにのファイルとしてすることもありますし、メモリにだけしてからさらにすることもできます。

プリプロセッサのコマンドはシャープ ""でまります。いくつかのプリプロセッサコマンドがあります。もなのは2つです。

1. `#define`はにのにされます。えば、

   ```
   #define BIGNUM 1000000
   int a = BIGNUM;
   ```

   になる

```
int a = 1000000;
```

このように`#define`をすると、ソースコードファイルのさまざまなにのをにきすがなくなります。でをするがあるには、これはです。 `#define`では、コードにらばっているのでするがあります。

`#define`はなとだけをうため、マクロをすることもできます。えば

```
#define ISTRUE(stm) do{stm = stm ? 1 : 0;}while(0)
// in the function:
a = x;
ISTRUE(a);
```

のようになる

```
// in the function:
a = x;
do {
    a = a ? 1 : 0;
} while(0);
```

のでは、このはインラインとほぼじですが、プリプロセッサは`#define`マクロのチェックをいません。これはエラーがこりやすいことがよくられており、そのにはきながです。

また、プリプロセッサは、のようにコメントをにきえることにしてください。

2. まれるもの

`#include`は、ソースコードファイルのでされたにアクセスするためにされます。えば

```
 #include <stdio.h>
```

プリプロセッサは、 `<stdio.h>`をコンパイルするに、 `#include`のにあるソースコードファイルにりけます。 `#include`はほとんどの、ヘッダファイルをインクルードするためにされます。ヘッダファイルは、にと`#define`をむファイルです。この、 `#include`をして`printf`や`scanf`などのをできるようにします。こののは`stdio.h`ファイルにあります。 Cコンパイラでは、にそのファイルでまたはされていないり、をすることはできません。 `#include`は、Cプログラムでにかれたコードをするです。

3.
```
#if defined A || defined B
variable = another_variable + 1;
#else
variable = another_variable * 2;
#endif
```

されます

```
variable = another_variable + 1;
```

にプロジェクトのどこかにAまたはBがされていた。これがてはまらない、もちろんプリプロセッサはこれをいます

```
variable = another_variable * 2;
```

これは、なるシステムでされるコードやなるコンパイラでコンパイルされるコードでよくされます。コンパイラ/システムのグローバルながあるので、それらのをテストして、コンパイラがずコンパイルするコードをさせることができます。

4. コメント

   プリプロセッサは、ソースファイルのすべてのコメントをのスペースできえます。コメントは、 `//`のまで、またはopening `/*`とclosing `*/`コメントのみわせによってされます。

コンパイラ

Cプリプロセッサがすべてのヘッダファイルをインクルードし、すべてのマクロをした、コンパイラはプログラムをコンパイルできます。これは、Cソースコードをオブジェクトコードファイルにすることによっています。これは、ソースコードのバイナリバージョンをむ`.o`でわるファイルです。ただし、オブジェクトコードはではありません。ファイルをするには、ファイルに`#include`をむすべてのライブラリのコードをするがありますこれは`#include`をむものとじではありません。これがリンカのです。

に、Cコンパイラのびしは、しているシステムによってきくなります。ここではGCCコンパイラをしていますが、もっとくのコンパイラがすることにしてください。

```
% gcc -Wall -c foo.c
```

`%`はOSのコマンドプロンプトです。これは、 `foo.c`ファイルでプリプロセッサをし、それをオブジェクトコードファイル`foo.o`コンパイルするようにコンパイラにします。 `-c`オプションは、ソースコードファイルをオブジェクトファイルにコンパイルし、リンカーをびさないことをします。このオプション`-c`は、LinuxやmacOSなどのPOSIXシステムでできます。のシステムではなるをすることがあります。

あなたのプログラムが1つのソースコードファイルにまれている、わりにこれをうことができます

```
% gcc -Wall foo.c -o foo
```

これはコンパイラに`foo.c`でプリプロセッサをさせ、コンパイルしてリンクして`foo`というファイルをするようにします。 `-o`オプションは、ののがバイナリファイルプログラムのであることをします。 `-o`しなかった `gcc foo.c`としただけの、ファイルはなから`a.out`というになります。

に、コンパイラは、 `.c`ファイルをファイルにするときに、の4つのステップをします。

1. - `.c`ファイルの`#include`ディレクティブと`#define`マクロをテキストで

2. **compilation** - プログラムをアセンブリにしますこのステップで`-S`オプションをすることでコンパイラをできます
3. アセンブリ - アセンブリをコードにする
4. リンケージ - オブジェクトコードをライブラリにリンクしてファイルをする

また、しているコンパイラのはGCCで、コンテキストにじて "GNU C compiler"と "GNU compiler collection"のです。のCコンパイラがする。 Unixライクなオペレーティングシステムでは、くの、のコンパイラとのシンボリックリンクである「Cコンパイラ」のが`cc`です。 Linuxシステムでは、`cc`はしばしばGCCのエイリアスです。 macOSまたはOS-Xではclangをします。

POSIXでは、C99のとして`c99`です。デフォルトではC99をサポートしています。のバージョンのPOSIXではコンパイラとして`c89`でした。 POSIXでは、このコンパイラがでしたオプション`-c`と`-o`をすることもしています。

の`gcc`にある`-Wall`オプションは、わしいにするをするようコンパイラーにします。これはくおめします。のオプション 、えば`-Wextra`をすることもいえ`-Wextra`。

フェーズ

*§5.1.1.2*フェーズにリストされているC 2011のでは、ソースコードからプログラムイメージへのえば、ファイルは、8つのけられたステップでするようにリストされています。

1. ソースファイルのは、にじてソースセットにマップされます。このステップでは、がきえられます。
2. `\`でわるは、のとされます。
3. ソースコードは、とトークンとしてされます。
4. プリプロセッサがされ、ディレクティブをし、マクロをし、プラグマをします。 `#include`によってっられたソースファイルは、フェーズ14にじてにをします。プリプロセッサのはすべてされます。
5. およびリテラルのソースセットは、セットにマップされます。
6. いにするリテラルがされます。
7. ソースコードは、をするトークンにされます。
8. がされ、プログラムイメージがされます。

Cコンパイラのでは、いくつかのステップをみわせることができますが、ののがので々にわれたかのようにするがあります。

オンラインでをむ https://riptutorial.com/ja/c/topic/1337/

57: リテラル

- {initializer-list}

CのはC11-§6.5.2.5/ 3でべている

> カッコでまれたのにカッコでまれたイニシャライザのリストがされたはリテラルです。それはリストによってがえられるのオブジェクトをします。 99

そして99によると

> これはキャストとはなります。たとえば、キャストはスカラまたは**void**のみへのをし、キャストのはではありません。

ごください

> リテラルとconstのリテラルは、のオブジェクトをするはありません。 101

> 101これにより、は、リテラルおよびリテラルのストレージを、じまたはするですることができます。

はでえられます
C11-§6.5.2.5/ 13

> リテラルとに、constされたリテラルはみりメモリにくことができ、することもできます。えば、
>
> ```
> (const char []){"abc"} == "abc"
> ```
>
> リテラルのストレージがされているは1がされます。

Examples

リテラルの*/*

リテラルは、されているスコープにされるのないオブジェクトです。このコンセプトはC99でめてされました。リテラルの

Cからの、C11-§6.5.2.5/ 9

```
int *p = (int [2]){ 2, 4 };
```

pは、2つのintののないののののアドレスにされます。

リテラルはです。のないオブジェクトのはファイルスコープでリテラルがされているまたはブロ

ックスコープでリテラルがされているのいずれかで、のは、がむブロックをれるとオブジェクトのがします。

```
void f(void)
{
    int *p;
    /*...*/
    p = (int [2]){ *p };
    /*...*/
}
```

pは、2つのintからなるののののアドレスがりてられます。のは、にpしされたをち、2のはゼロです。

ここで、 pはブロックのまでです。

きリテラル

C11からも

```
  struct point {
    unsigned x;
    unsigned y;
  };

  extern void drawline(struct point, struct point);

// used somewhere like this
drawline((struct point){.x=1, .y=1}, (struct point){.x=3, .y=4});
```

のdrawlineは、 struct pointの2つのをけります。のはx == 1とy == 1であり、2のはx == 3とy == 4

のさをしないリテラル

```
int *p = (int []){ 1, 2, 3};
```

この、のサイズはされていません。のさによってされます。

のさがされたサイズよりさいリテラル

```
int *p = (int [10]){1, 2, 3};
```

リテラルのりのはに0にされ0 。

みりリテラル

リテラルはなので、はです。 みりのリテラルは、 `const`を`(const int[]){1,2}`としてしてできます。

のをむリテラル

のでは、リテラルは、C99ののとに、のをつことができます。

```
void foo()
{
    int *p;
    int i = 2; j = 5;
    /*...*/
    p = (int [2]){ i+j, i*j };
    /*...*/
}
```

オンラインでリテラルをむ https://riptutorial.com/ja/c/topic/4135/リテラル

58: シーケンス

すべてのプリプロセッサがトリグラフシーケンスをサポートするわけではありません。いくつかのコンパイラは、それらをするためのなオプションやスイッチをしています。は、のプログラムをしてをします。

GCCコンパイラはにしないかぎり、それらをしませんにするには`-trigraphs`をし、 `-Wall`である`-Wtrigraphs`して、トリグラフについてのをします。

されているほとんどのプラットフォームはCでされているのをサポートしているため、よりもグラフがされますが、ののはにはされません。

また、ったグラフの `puts("What happened??!!");`などにもしてください。

Examples

トリグラフ

`[ ] { } ^ \ | ~ #`えばCプログラムでされるが、1980に、コードセットはISO 646のは、では、えば、をしていたこれらのASCIIは、バリアントのためにされたをされています`£#`では、 `Æ Å æ å ø Ø`のため`{ } { } | \`デンマークで、もありませんでした`~` EBCDICで。これは、これらのセットをするマシンにCコードをするのはしいことをしていました。

このをするために、Cでは3のみわせをして3という1をすることをしました。トリグラフは3のシーケンスで、の2はです。

は、 `#` 、 `{`と`}`わりにトリグラフシーケンスをするなです

```
??=include <stdio.h>

int main()
??<
    printf("Hello World!\n");
??>
```

これは、Cプリプロセッサによって、コードがかれているかのように、をにきえることによってされます

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
}
```

トリグラフ	
?? =	
?? /	\
?? '	^
??	[
??	]
??	|
?? <	{
??>	}
?? -	

えば、 ??/はバックスラッシュであり、コメントののにをぼし、やリテラルのでされなければならないことにしてくださいえば、 '??/??/'はです、バックスラッシュ。

グラフ

C99

1994には、5つのトリグラフのよりみいがされた。これらは2のみをし、グラフとしてられています。とはなり、グラフはトークンです。のトークンえば、リテラルまたはでグラフがした、それはグラフとしてわれることはありませんが、そのままになります。

に、グラフシーケンスののをします。

```
#include <stdio.h>

int main()
<%
    printf("Hello %> World!\n"); /* Note that the string contains a digraph */
%>
```

どちらもじようにわれます

```
#include <stdio.h>

int main()
{
    printf("Hello %> World!\n"); /* Note the unchanged digraph within the string. */
}
```

グラフ	
<	[
>	]
<	{
>	}

オンラインでシーケンスをむ https://riptutorial.com/ja/c/topic/7111/シーケンス

59: スコープ

Examples

ブロックスコープ

は、するがブロックにあるのパラメータがされる、ブロックスコープをちます。スコープは、するブロックのでします。

じをつなるエンティティはじスコープをつことはできませんが、スコープはするがあります。スコープがしているは、ものスコープでされているスコープのみがされます。

```
#include <stdio.h>

void test(int bar)                    // bar has scope test function block
{
    int foo = 5;                      // foo has scope test function block
    {
        int bar = 10;                 // bar has scope inner block, this overlaps with previous
test:bar declaration, and it hides test:bar
        printf("%d %d\n", foo, bar); // 5 10
    }                                 // end of scope for inner bar
    printf("%d %d\n", foo, bar);      // 5 5, here bar is test:bar
}                                     // end of scope for test:foo and test:bar

int main(void)
{
    int foo = 3;          // foo has scope main function block

    printf("%d\n", foo); // 3
    test(5);
    printf("%d\n", foo); // 3
    return 0;
}                         // end of scope for main:foo
```

のプロトタイプスコープ

```
#include <stdio.h>

/* The parameter name, apple, has function prototype scope.  These names
   are not significant outside the prototype itself.  This is demonstrated
   below. */

int test_function(int apple);

int main(void)
{
    int orange = 5;

    orange = test_function(orange);
    printf("%d\r\n", orange); //orange = 6
```

```
    return 0;
}

int test_function(int fruit)
{
    fruit += 1;
    return fruit;
}
```

プロトタイプにをすると、するエラーメッセージがることにしてください。

```
int function(struct whatever *arg);

struct whatever
{
    int a;
    // ...
};

int function(struct whatever *arg)
{
    return arg->a;
}
```

GCC 6.3.0では、このコードソースファイル`dc11.c`はをします。

```
$ gcc -O3 -g -std=c11 -Wall -Wextra -Werror -c dc11.c
dc11.c:1:25: error: ‘struct whatever’ declared inside parameter list will not be visible
outside of this definition or declaration [-Werror]
     int function(struct whatever *arg);
                         ^~~~~~~~
dc11.c:9:9: error: conflicting types for ‘function’
     int function(struct whatever *arg)
         ^~~~~~~~
dc11.c:1:9: note: previous declaration of ‘function’ was here
     int function(struct whatever *arg);
         ^~~~~~~~
cc1: all warnings being treated as errors
$
```

のにをくか、`struct whatever;`し`struct whatever;`ののとして、はありません。そのをするがないため、プロトタイプにしいをしないでください。したがって、そのをまたはするがないためです。

ファイルスコープ

```
#include <stdio.h>

/* The identifier, foo, is declared outside all blocks.
   It can be used anywhere after the declaration until the end of
   the translation unit. */
static int foo;

void test_function(void)
{
```

```
    foo += 2;
}

int main(void)
{
    foo = 1;

    test_function();
    printf("%d\r\n", foo); //foo = 3;

    return 0;
}
```

スコープはラベルのなスコープです。これは、そのしいによるものです。 ラベルは、されているをしてであり、じののからジャンプすることができます `goto label`をして。ではありませんが、のは、このをしています。

```
#include <stdio.h>

int main(int argc,char *argv[]) {
    int a = 0;
    goto INSIDE;
  OUTSIDE:
    if (a!=0) {
        int i=0;
      INSIDE:
        printf("a=%d\n",a);
        goto OUTSIDE;
    }
}
```

`INSIDE`でわれるかもしれない`if`、それはののように、ブロック`i`スコープはブロックされ、それはありません。これは、`goto INSIDE;`として、でされ`goto INSIDE;`する。したがって、じをつ2つのラベルを1つのにめることはできません。

なは、りてられたリソースのしいなクリーンアップをするために、のパターンです。

```
#include <stdlib.h>
#include <stdio.h>

void a_function(void) {
   double* a = malloc(sizeof(double[34]));
   if (!a) {
      fprintf(stderr,"can't allocate\n");
      return;                 /* No point in freeing a if it is null */
   }
   FILE* b = fopen("some_file","r");
   if (!b) {
      fprintf(stderr,"can't open\n");
      goto CLEANUP1;          /* Free a; no point in closing b */
   }
   /* do something reasonable */
   if (error) {
      fprintf(stderr,"something's wrong\n");
      goto CLEANUP2;       /* Free a and close b to prevent leaks */
   }
```

```
   /* do yet something else */
CLEANUP2:
   close(b);
CLEANUP1:
   free(a);
}
```

`CLEANUP1`や`CLEANUP2`などのラベルは、のすべてのとはなるをするなです。ラベルは、たとえラベルきののにされるであっても、のどこからでも、または`goto`がされないとできないでもされます。ラベルは、ではなくでされることがよくあります。

オンラインでスコープをむ https://riptutorial.com/ja/c/topic/1804/スコープ

60: ステートメント

Examples

if ステートメント

プログラムフローをするもなの1つは、 `if` を `if` ことです。このでコードのブロックをするかどうかをすることができます。

C の `if` のは `if` のようになります。

```
if(cond)
{
  statement(s);  /*to be executed, on condition being true*/
}
```

えば、

```
if (a > 1) {
    puts("a is larger than 1");
}
```

ここで `a > 1` は、 `if` ブロックのステートメントをするために `true` とするがあるです。このでは、 "a is greater than 1"は、 `a > 1` にのみされます。

`if` ステートメントはをすることができ `{` と `}` ブロックののステートメントがある。のは、のようにきすことができます。

```
if (a > 1)
    puts("a is larger than 1");
```

しかし、ブロックでのステートメントをするには、をするがあります。

`if` のには、のをめることができます。 expression のが true の `if` にのみアクションをします。

えば

```
if ((a > 1) && (b > 1)) {
    puts("a is larger than 1");
    a++;
}
```

`a` と `b` が `1` よりきいにのみ、 `printf` と `a++` します。

if... else ステートメントと

で`if`、そのはとされたにのみアクションを`true`、 `if` / `else`あなたはにのなるアクションをすることができます`true`とがあるとき`false`。

```
if (a > 1)
    puts("a is larger than 1");
else
    puts("a is not larger than 1");
```

`if`の`if`とに、 `if`または`else`のブロックが1つのでされている`if`、はできますしかし、そうすることは、でにをきこすがあるためしません。 `if`または`else`ブロックにのステートメントがある`if`は、そののブロックでをするがあります。

```
if (a > 1)
{
    puts("a is larger than 1");
    a--;
}
else
{
    puts("a is not larger than 1");
    a++;
}
```

switchステートメント

`switch`は、プログラムがのテストのにじてくのなることをしたいときにです。

`switch`のはのようになります

```
int a = 1;

switch (a) {
case 1:
    puts("a is 1");
    break;
case 2:
    puts("a is 2");
    break;
default:
    puts("a is neither 1 nor 2");
    break;
}
```

このは、

```
int a = 1;

if (a == 1) {
    puts("a is 1");
} else if (a == 2) {
    puts("a is 2");
} else {
    puts("a is neither 1 nor 2");
```

```
}
```

`switch`がされているときに`a`のが1の、 `a is 1`がされます。、に2であり、 `a a is 2`されます。それの`a is neither 1 nor 2`もされ`a is neither 1 nor 2`。

`case n: switch`にされたが*n*のときにフローがどこにジャンプするかをするためにされます。 *n*はコンパイルでなければならず、じ*n*がでも1つの`switch`にすることができます。

`default:`は、が`case n:`いずれのともしなかったことをするためにされ`default:`。のをするには、すべてのswitchに`default` caseをめることをおめします。

`break; switch`ブロックからびりるにはステートメントがです。

`case`のに`break`をすることをってれてしまった`case`、コンパイラは「フォールスルー」し、そのにくすべてのcaseがされるとします *break*がのケースステートメントがするかどうかにかかわらず、のケースのいずれか。こののプロパティは、 Duffのデバイスをするためにされます。このは、しばしばCのとなされます。

は、 `break;`をす`break;`

```
int a = 1;

switch (a) {
case 1:
case 2:
    puts("a is 1 or 2");
case 3:
    puts("a is 1, 2 or 3");
    break;
default:
    puts("a is neither 1, 2 nor 3");
    break;
}
```

`a`のが1または2のとき、 `a is 1 or 2`、 `a is 1, 2 or 3`をします。 `a`が3のときは`a`のみ`a is 1, 2 or 3`がされます。それの、 `a is neither 1, 2 nor 3`され`a is neither 1, 2 nor 3`。

`default`ケースはではないことにしてください。に、 `switch`したのセットがし、コンパイルにしているはしてください。

もいは、 `enum switch`をする`switch`です。

```
enum msg_type { ACK, PING, ERROR };
void f(enum msg_type t)
{
  switch (t) {
  case ACK:
    // do nothing
    break;
  case PING:
    // do something
    break;
```

```
  case ERROR:
    // do something else
    break;
  }
}
```

これにはのがあります。

- ほとんどのコンパイラはをしないとをします `default`はされません
- のから、 `enum`しいをすると、しいをするのをれたがすべてされます `default`は、そのようなをでするがあります
- は、「 `default:`されているもの`default:` 」、の`enum`があるかどうか、または「ちょうどの」のであるかどうかをするはありません。また、の`enum`がある、コーダーはに`default`をしましたか、をしたときにされたバグがありますか
- ワイルドカードのにれることができないので、`enum`をすることで、コードをにすることができます。それぞれをにするがあります。

それにもかかわらず、かがのようななコードをくのをぐことはできません

```
enum msg_type t = (enum msg_type)666; // I'm evil
```

したがって、になは、スイッチのになチェックをしてすることができます。

```
void f(enum msg_type t)
{
   if (!is_msg_type_valid(t)) {
      // Handle this unlikely error
   }

   switch(t) {
    // Same code than before
   }
}
```

if... else elseのif... elseステートメントをする

`if ()... else`ステートメントは、 `if ()`のがたされない`if ()`する1つのデフォルトをすることができますが、2つの`if () ... else`ステートメントをさせると、もしあれば、 "default"としてするの`else`ブランチにくに、よりくのるいをします。

```
int a = ... /* initialise to some value. */

if (a >= 1)
{
    printf("a is greater than or equals 1.\n");
}
else if (a == 0) //we already know that a is smaller than 1
{
    printf("a equals 0.\n");
}
else /* a is smaller than 1 and not equals 0, hence: */
```

```
{
    printf("a is negative.\n");
}
```

れになったif... else VS if.. else Ladder

ネストされた`if()...else`ステートメントは、 `if()...else`ステートメントのネストされた`if()...else`ステートメントがのをすべてチェックしてから、 `if()...else`ラダーとしてがくなりますき`if()`がたされますが、 `if()`または`else if()`のいずれかがtrueになると、 `if()..else`ラダーはテストをします。

`if()...else`ラダー

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int a, b, c;
  printf("\nEnter Three numbers = ");
  scanf("%d%d%d", &a, &b, &c);
  if ((a < b) && (a < c))
  {
    printf("\na = %d is the smallest.", a);
  }
  else if ((b < a) && (b < c))
  {
    printf("\nb = %d is the smallest.", b);
  }
  else if ((c < a) && (c < b))
  {
    printf("\nc = %d is the smallest.", c);
  }
  else
  {
    printf("\nImprove your coding logic");
  }
  return 0;
}
```

なケースでは、なれの`if()...else`よりもれているとえられ`if()...else`

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int a, b, c;
  printf("\nEnter Three numbers = ");
  scanf("%d%d%d", &a, &b, &c);
  if (a < b)
  {
    if (a < c)
      {
        printf("\na = %d is the smallest.", a);
      }
    else
      {
```

```
        printf("\nc = %d is the smallest.", c);
      }
  }
  else
  {
    if(b < c)
    {
      printf("\nb = %d is the smallest.", b);
    }
    else
    {
      printf("\nc = %d is the smallest.", c);
    }
  }
  return 0;
}
```

オンラインでステートメントをむ https://riptutorial.com/ja/c/topic/3073/ステートメント

61:

き

は、ののけられた「」をすデータです。 Cののは、いずれかののののをち、そのは、をギャップまたはパディングなしでメモリにしてします。 Cは、がのである、およびポインタのをします。

Cは、にサイズがされるにりてられたをサポートします。 C99では、またはVLAがサポートされています。

- タイプ[さ]; / * 'name'とさ 'length'の 'type'のをします。 * /
- int arr [10] = {0}; / *をし、すべてのを0にする* /
- int arr [10] = {42}; / *をし、のを42にし、りを0にします。 * /
- int arr [] = {4,2,3,1}; / *さ4のをしてする* /
- arr [n] = value; / *インデックスnのをします。 * /
- = arr [n]; / *インデックスnのをします。 * /

なぜがなのでしょうか

は、のをってオブジェクトをにするをします。たとえば、Cはの `char` sで、 "Hello、World"などのです。キャラクターに々にのものではないとしてのをちます。に、はなベクトルや、そしてくののリストをすためによくわれます。さらに、をグループするらかのがなくても、々のをしてなど、々ににするがあります。それはいにくいだけでなく、さまざまなさのコレクションをにけれることができません。

は、ほとんどのコンテキストでにポインタにされます 。

`sizeof`、 `_Alignof`C2011、 `&_Alignof` of、またはののにされるリテラルのオペランドとしてされるをいて、はにのへのポインタ。こののは、 `[]` のににしています。 `arr[idx]`は`*(arr + idx)`とであるとされています。さらに、ポインタはであるため、 `*(arr + idx)`も`*(idx + arr)`とであり、`idx[arr]`とです。 `idx`または`arr`いずれかがポインタまたはポインタにするであり、もうがであり、がへのなインデックスである、それらのはすべてであり、じにされますこれにポインタがす。

なケースとして、 `&(arr[0])`は`&*(arr + 0)`とであることに`arr`。これは`arr`になります。これらのはすべて、のポインタがえたところであればどこでもです。これは、がのへのポインタにすることをにします。

に、アドレスが`T[N]` すなわち `&arr` のにされた、のは`T (*)[N]`あり、をします。これはなくともポインター—のサイズにしてされたポインター—にして、のへのポインター—とはなります。

のパラメータはではありません 。

```
void foo(int a[], int n);
void foo(int *a, int n);
```

`foo`ののでは、パラメータ`a`にしてのようながされますが、このようなは、パラメータがそのパラメータをのへのポインタとしてするためにされます。したがって、 `foo()` 2のは、のとにじです。これは、とのパラメータはじでされているは、そののとしてびしでのにしているような、らはびしのとしてれるポインタへののにしますにけられた。

Examples

のと

1をするためのなはのとおりです。

```
type arrName[size];
```

`type`はなどのみみまたはユーザー`type`できます`arrName`はユーザーので、 `size`はです。

このは10のintのをするには、のようにします。

```
int array[10];
```

はなをしています。にゼロをするには、のようにします。

```
int array[10] = {0};
```

アレイはまた、をつことができ、このでは10のアレイを`int`「3の、 `int` sはがまれています」`1` 、 `2` 、 `3` 、のすべてのはゼロになります。

```
int array[10] = {1, 2, 3};
```

ののでは、リストののははののメンバーにりてられ、2のはの2のメンバーにりてられます。リストのサイズがのサイズよりもさい、ののように、のりのメンバーはゼロにされます。リストのISO C99により、メンバーのながです。えば、

```
int array[5] = {[2] = 5, [1] = 2, [4] = 9}; /* array is {0, 2, 5, 0, 9} */
```

ほとんどの、コンパイラはのさをできますが、これはをのままにすることでできます。

```
int array[] = {1, 2, 3}; /* an array of 3 int's */
int array[] = {[3] = 8, [0] = 9}; /* size is 4 */
```

ゼロのさのをすることはできません。

C99 C11

C99ではアレイVLAをし、C11ではオプションとしました。これらは、のとじですが、ないが1つあります。コンパイルにさをるはありません。 VLAにはがあります。 VLAへのポインタだけがを

っことができます。

```
size_t m = calc_length(); /* calculate array length at runtime */
int vla[m];               /* create array with calculated length */
```

VLAはにです。ののvlaがスタックになよりくのをとする、スタックはオ―バ―フロ―します。VLAのはスタイルガイドややではしばしばされません。

のをするゼロ

によっては、がしたにをゼロにするがあります。

```
#include <stdlib.h> /* for EXIT_SUCCESS */

#define ARRLEN (10)

int main(void)
{
  int array[ARRLEN]; /* Allocated but not initialised, as not defined static or global. */

  size_t i;
  for(i = 0; i < ARRLEN; ++i)
  {
    array[i] = 0;
  }

  return EXIT_SUCCESS;
}
```

のル―プのなmemset()は、 <string.h> memset() をすることです。のようにarrayをすと、のへのポインタへのがわれます。

```
memset(array, 0, ARRLEN * sizeof (int)); /* Use size explicitly provided type (int here). */
```

または

```
memset(array, 0, ARRLEN * sizeof *array); /* Use size of type the pointer is pointing to. */
```

こののように、 array はののへのポインタではなく のさがであるを、を0にする3のオプションがです

```
 memset(array, 0, sizeof array); /* Use size of the array itself. */
```

のさ

はのでられているです。それにもかかわらず、のさをすることはであり、にはです。に、のさがイニシャライザからにされるとき、コ―ドをよりにすることができます

```
int array[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

```
/* size of `array` in bytes */
size_t size = sizeof(array);

/* number of elements in `array` */
size_t length = sizeof(array) / sizeof(array[0]);
```

しかし、がにれるほとんどのコンテキストでは、のへのポインタににされます「する」。が`sizeof`のオペランドであるは、のの1つです。としてじるポインタはそれがではなく、それがされたのさにするをちません。したがって、ポインター—がにされたときなど、そのさがポインター—とになは、々にするがあります。

たとえば、`int`ののをすをしたいとします。からきき、のようにびすことができます。

```
/* array will decay to a pointer, so the length must be passed separately */
int last = get_last(array, length);
```

はのようにできます

```
int get_last(int input[], size_t length) {
    return input[length - 1];
}
```

に、パラメータ`input`のはのにてい`input`が、には**`input`**をポインタ `int` としてしています。 `input`を`int *input`としてするのとくじです。がえられてもじことがてはまります。これは、がののびしにれるときにポインタにするであることができず、ニーモニックとしてることができるためです。

のサイズをポインタからしようとするのはになエラーですが、これはうまくいきません。こんなことしないで

```
int BAD_get_last(int input[]) {
    /* INCORRECTLY COMPUTES THE LENGTH OF THE ARRAY INTO WHICH input POINTS: */
    size_t length = sizeof(input) / sizeof(input[0]));

    return input[length - 1];  /* Oops -- not the droid we are looking for */
}
```

には、そののエラーはにであり、のコンパイラはそれをしてします。たとえば、`clang`はのをするでしょう

```
warning: sizeof on array function parameter will return size of 'int *' instead of 'int []' [-
Wsizeof-array-argument]
        int length = sizeof(input) / sizeof(input[0]);
                           ^
note: declared here
int BAD_get_last(int input[])
                     ^
```

をでする

のへのアクセスは、、かっこでいます。

```
int val;
int array[10];

/* Setting the value of the fifth element to 5: */
array[4] = 5;

/* The above is equal to: */
*(array + 4) = 5;

/* Reading the value of the fifth element: */
val = array[4];
```

+にするオペランドのが - >なので、はです

```
*(array + 4) = 5;
*(4 + array) = 5;
```

に、のステートメントはです。

```
array[4] = 5;
4[array] = 5; /* Weird but valid C ... */
```

これらの2もです

```
val = array[4];
val = 4[array]; /* Weird but valid C ... */
```

Cはチェックをせず、されたののにアクセスすることはされていません りてられたチャンクをえるメモリへのアクセス 。

```
int val;
int array[10];

array[4] = 5;    /* ok */
val = array[4];  /* ok */
array[19] = 20;  /* undefined behavior */
val = array[15]; /* undefined behavior */
```

とアクセスをする

```
#include <stdio.h>

#define ARRLEN (10)

int main (void)
{

   int n[ ARRLEN ]; /* n is an array of 10 integers */
   size_t i, j; /* Use size_t to address memory, that is to index arrays, as its guaranteed to
```

```
                    be wide enough to address all of the possible available memory.
                    Using signed integers to do so should be considered a special use case,
                    and should be restricted to the uncommon case of being in the need of
                    negative indexes. */

   /* Initialize elements of array n. */
   for ( i = 0; i < ARRLEN ; i++ )
   {
      n[ i ] = i + 100; /* Set element at location i to i + 100. */
   }

   /* Output each array element's value. */
   for (j = 0; j < ARRLEN ; j++ )
   {
      printf("Element[%zu] = %d\n", j, n[j] );
   }

   return 0;
}
```

ユーザーのサイズでをりて、ゼロする

```
#include <stdio.h>
#include <stdlib.h>


int main (void)
{
  int * pdata;
  size_t n;

  printf ("Enter the size of the array: ");
  fflush(stdout); /* Make sure the prompt gets printed to buffered stdout. */

  if (1 != scanf("%zu", &n)) /* If zu is not supported (Windows?) use lu. */
  {
    fprintf("scanf() did not read a in proper value.\n");
    exit(EXIT_FAILURE);
  }

  pdata = calloc(n, sizeof *pdata);
  if (NULL == pdata)
  {
    perror("calloc() failed"); /* Print error. */
    exit(EXIT_FAILURE);
  }

  free(pdata); /* Clean up. */

  return EXIT_SUCCESS;
}
```

このプログラムは、からなしをスキャンし、 `calloc()` をびして `int` の `n` のに `n` メモリブロックを `calloc()` ます。メモリは、によってすべてゼロにされます。

の、メモリは `free()` びしによってされます。

かつでをする

Cのはしたメモリのチャンクとみなすことができます。よりには、ののはしたです。これをとびます。そののキャッシュのをぐために、キャッシュされていないデ―タにアクセスするに10000x10000のアレイにアクセスなぜ、たちがることができるキャッシュにこのキャッシュがなキャッシュラインをロ―ドしているというをする`array[0][0]` にロ―ドし`array[0][1]`キャッシュではなく、アクセス`array[1][0]` 、それがあるため、キャッシュフォ―ルトをした`sizeof(type)*10000`バイトかられる`array[0][0]`ってかではありませんじキャッシュラインにある。このようにするのは、なです。

```
#define ARRLEN 10000
int array[ARRLEN][ARRLEN];

size_t i, j;
for (i = 0; i < ARRLEN; ++i)
{
    for(j = 0; j < ARRLEN; ++j)
    {
        array[j][i] = 0;
    }
}
```

このようにするがです

```
#define ARRLEN 10000
int array[ARRLEN][ARRLEN];

size_t i, j;
for (i = 0; i < ARRLEN; ++i)
{
    for(j = 0; j < ARRLEN; ++j)
    {
        array[i][j] = 0;
    }
}
```

じように、これは、1とのインデックスをつをうときインデックスiとjをするためにここでは2としましょう、をのようにりしすることがです

```
#define DIM_X 10
#define DIM_Y 20

int array[DIM_X*DIM_Y];

size_t i, j;
for (i = 0; i < DIM_X; ++i)
{
    for(j = 0; j < DIM_Y; ++j)
    {
        array[i*DIM_Y+j] = 0;
    }
}
```

または、3とインデックスi、j、およびkをして、

```
#define DIM_X 10
#define DIM_Y 20
#define DIM_Z 30

int array[DIM_X*DIM_Y*DIM_Z];

size_t i, j, k;
for (i = 0; i < DIM_X; ++i)
{
    for(j = 0; j < DIM_Y; ++j)
    {
        for (k = 0; k < DIM_Z; ++k)
        {
            array[i*DIM_Y*DIM_Z+j*DIM_Z+k] = 0;
        }
    }
}
```

あるいは、よりなでは、 **N1 x N2 x ... x Nd**、 **d**、および**n1**、n2、...、ndとされるインデックスをつがある、オフセットはのようにされます

$$n_d + N_d \cdot (n_{d-1} + N_{d-1} \cdot (n_{d-2} + N_{d-2} \cdot (\cdots + N_2 n_1) \cdots)) = \sum_{k=1}^{d} \left(\prod_{\ell=k+1}^{d} N_\ell \right) n_k$$

/ https : //en.wikipedia.org/wiki/Row-major_order

Cでは、 がです。ここには、のながあります。

```
type name[size1][size2]...[sizeN];
```

たとえば、のは35 x 10 x 4をします。

```
int arr[5][10][4];
```

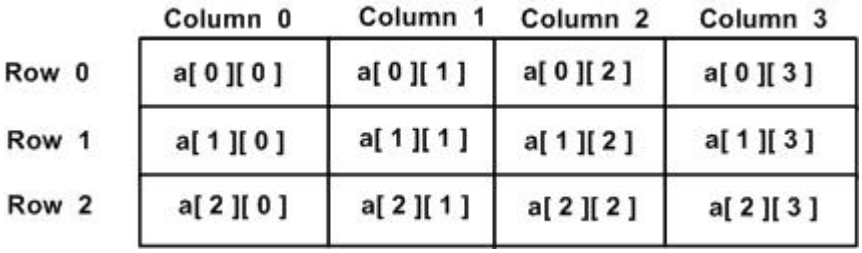
2

のもなは2です。 2は、に、1のリストである。mxnの2をするには、のようにできます。

```
type arrayName[m][n];
```

`type`はのなCデータ `int` 、 `float`などで、 `arrayName`はのなCです。 2は、 `mn`のとしてすることができます。 はCにアレイない `int a[4][3]`とじではない`int a[3][4]` 。 C はであるため、 がになります。

34の2_a 、のようにすことができます。



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

したがって、aすべてのは、 a[i][j]というのでされa[i][j]。 ここでaはの、 iは、 jはどのをします。とのインデックスはゼロであることをいしてください。これは、2のとによくています。

2の

は、にでまれたをすることによってすることができます。のでは、に4つのがある3のをしています。

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

のをすネストされたカッコはオプションです。のは、のとじです。

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

ネストされたカッコをしてをするはオプションですが、みやすくかりやすくするためにくされています。

2へのアクセス

2のは、え、すなわちのインデックスおよびインデックスをしてアクセスされます。えば -

```
int val = a[2][3];
```

のステートメントは、の3のから4のをします。ネストされたループをって2をったのプログラムをしましょう

```
#include <stdio.h>

int main () {

   /* an array with 5 rows and 2 columns*/
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   int i, j;

   /* output each array element's value */
   for ( i = 0; i < 5; i++ ) {

      for ( j = 0; j < 2; j++ ) {
         printf("a[%d][%d] = %d\n", i,j, a[i][j] );
      }
```

```
    }

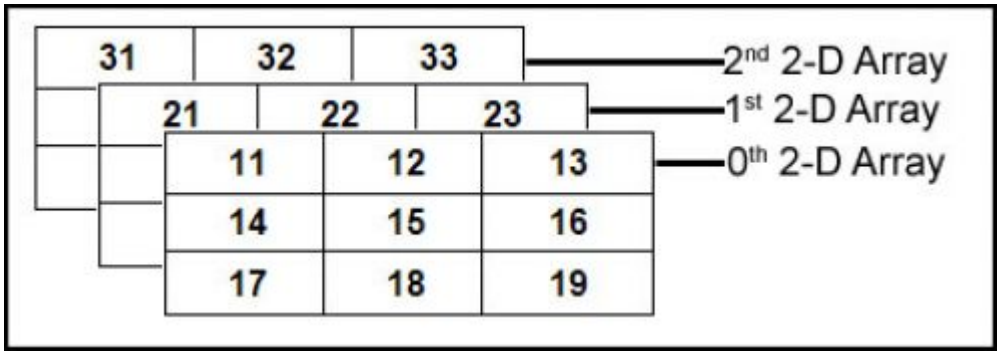
    return 0;
}
```

のコードをコンパイルしてすると、のがされます。

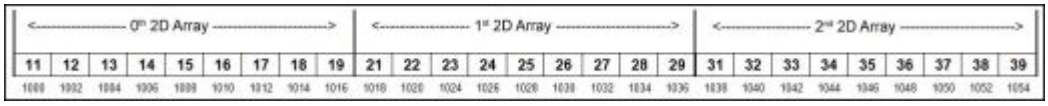
```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

3

3Dはにののです。または2Dのコレクションであり、2Dは1Dのです。



3Dアレイメモリマップ



3Dの

```
double cprogram[3][2][4]={
{{-0.1, 0.22, 0.3, 4.3}, {2.3, 4.7, -0.9, 2}},
 {{0.9, 3.6, 4.5, 4}, {1.2, 2.4, 0.22, -1}},
 {{8.2, 3.12, 34.2, 0.1}, {2.1, 3.2, 4.3, -2.0}}
};
```

ののをつことができますが、されるのは1または2です。

ポインタをしてをする

```
#include <stdio.h>
#define SIZE (10)
int main()
```

```
{
    size_t i = 0;
    int *p = NULL;
    int a[SIZE];

    /* Setting up the values to be i*i */
    for(i = 0; i < SIZE; ++i)
    {
        a[i] = i * i;
    }

    /* Reading the values using pointers */
    for(p = a; p < a + SIZE; ++p)
    {
        printf("%d\n", *p);
    }

    return 0;
}
```

ここで、のforループのpのでは、a は 、そのようながされているほぼすべてのとに、のへのポインタにします。

いで、 ++pポインタにポインタをいpおよびのを1つずつをく、そしてそれらをすることによってそれらをす*p 。

をにす

は、にすときに、1とじにいます。しかし、ディケイ･ツ―･ポインター、、およびのポインター―のをする2つのなるのみわせは、そのようなのをにするがあります。のは、をすしいをしています。

```
#include <assert.h>
#include <stdlib.h>

/* When passing a multidimensional array (i.e. an array of arrays) to a
   function, it decays into a pointer to the first element as usual.  But only
   the top level decays, so what is passed is a pointer to an array of some fixed
   size (4 in this case). */
void f(int x[][4]) {
    assert(sizeof(*x) == sizeof(int) * 4);
}

/* This prototype is equivalent to f(int x[][4]).
   The parentheses around *x are required because [index] has a higher
   precedence than *expr, thus int *x[4] would normally be equivalent to int
   *(x[4]), i.e. an array of 4 pointers to int.  But if it's declared as a
   function parameter, it decays into a pointer and becomes int **x,
   which is not compatable with x[2][4]. */
void g(int (*x)[4]) {
    assert(sizeof(*x) == sizeof(int) * 4);
}

/* An array of pointers may be passed to this, since it'll decay into a pointer
   to pointer, but an array of arrays may not. */
void h(int **x) {
    assert(sizeof(*x) == sizeof(int*));
}
```

```
int main(void) {
    int foo[2][4];
    f(foo);
    g(foo);

    /* Here we're dynamically creating an array of pointers.  Note that the
       size of each dimension is not part of the datatype, and so the type
       system just treats it as a pointer to pointer, not a pointer to array
       or array of arrays. */
    int **bar = malloc(sizeof(*bar) * 2);
    assert(bar);
    for (size_t i = 0; i < 2; i++) {
        bar[i] = malloc(sizeof(*bar[i]) * 4);
        assert(bar[i]);
    }

    h(bar);

    for (size_t i = 0; i < 2; i++) {
        free(bar[i]);
    }
    free(bar);
}
```

もしてください

へののけし

オンラインでをむ https://riptutorial.com/ja/c/topic/322/

62: のパラメータ

Cでは、するエラーをすためにりをするのがです。されたポインタをしてデータをすことができます。これはのでうことができます。ヒープにメモリをりてたり、がびされたでりてをしたりするはありません。

Examples

ポインタパラメータをしてのをす

Cのなパターンは、からのをすことをにするために、ポインタをすることです。

```
#include <stdio.h>

void Get( int* c , double* d )
{
    *c = 72;
    *d = 175.0;
}

int main(void)
{
    int a = 0;
    double b = 0.0;

    Get( &a , &b );

    printf("a: %d, b: %f\n", a , b );

    return 0;
}
```

へののけし

```
int getListOfFriends(size_t size, int friend_indexes[]) {
  size_t i = 0;
  for (; i < size; i++) {
    friend_indexes[i] = i;
  }
}
```

C99 C11

```
/* Type "void" and VLAs ("int friend_indexes[static size]") require C99 at least.
   In C11 VLAs are optional. */
void getListOfFriends(size_t size, int friend_indexes[static size]) {
  size_t i = 0;
  for (; i < size; i++) {
    friend_indexes[i] = 1;
  }
}
```

ここでは、のパラメータの`[]`の`static`は、になくともされた `size`があることをします。そのをできるようにするには、 `size`パラメータがリストのパラメータのにるようにするがあります。

のように`getListOfFriends()`します。

```
#define LIST_SIZE (50)

int main(void) {
  size_t size_of_list = LIST_SIZE;
  int friends_indexes[size_of_list];

  getListOfFriends(size_of_list, friend_indexes); /* friend_indexes decays to a pointer to the
                                                     address of its 1st element:
                                                       &friend_indexes[0] */

  /* Here friend_indexes carries: {0, 1, 2, ..., 49}; */

  return 0;
}
```

もしてください

をにす

パラメータはによってされます

Cでは、すべてのパラメータがされるので、びしでされたものをしても、びしのローカルにはしません。

```
#include <stdio.h>

void modify(int v) {
    printf("modify 1: %d\n", v); /* 0 is printed */
    v = 42;
    printf("modify 2: %d\n", v); /* 42 is printed */
}

int main(void) {
    int v = 0;
    printf("main 1: %d\n", v); /* 0 is printed */
    modify(v);
    printf("main 2: %d\n", v); /* 0 is printed, not 42 */
    return 0;
}
```

ポインターをすると、びしがびしのローカルをできるようになります。これはしではなく、ローカルをすポインタがされることにしてください。

```
#include <stdio.h>

void modify(int* v) {
    printf("modify 1: %d\n", *v); /* 0 is printed */
```

```
    *v = 42;
    printf("modify 2: %d\n", *v); /* 42 is printed */
}

int main(void) {
    int v = 0;
    printf("main 1: %d\n", v); /* 0 is printed */
    modify(&v);
    printf("main 2: %d\n", v); /* 42 is printed */
    return 0;
}
```

ただし、ローカルのアドレスをびしにすと、されていないになります。 をえたへのポインタのをしてください。

パラメータの

パラメータのは、Cプログラミングではされていません。ここでは、からへ、またはからへすることができる。はによってなります。

```
#include <stdio.h>

void function(int a, int b)
{
    printf("%d %d\n", a, b);
}

int main(void)
{
    int a = 1;
    function(a++, ++a);
    return 0;
}
```

エラーコードきのをむstructをすの

をすのほとんどのは、ポインタがの1つとしてされ、がのようにされたをできるようにします。ののりは、、のをすintのようなであるかどうかはありません。

```
int func (int *pIvalue)
{
    int iRetStatus = 0;             /* Default status is no change */

    if (*pIvalue > 10) {
        *pIvalue = *pIvalue * 45;   /* Modify the value pointed to */
        iRetStatus = 1;             /* indicate value was changed */
    }

    return iRetStatus;              /* Return an error code */
}
```

しかし、 structをりとしてすることもできます。これにより、エラーとののをすことができます。えば。

```
typedef struct {
    int    iStat;      /* Return status */
    int    iValue;     /* Return value */
}  RetValue;

RetValue func (int iValue)
{
    RetValue iRetStatus = {0, iValue};

    if (iValue > 10) {
        iRetStatus.iValue = iValue * 45;
        iRetStatus.iStat = 1;
    }

    return iRetStatus;
}
```

このは、のサンプルのようにできます。

```
int usingFunc (int iValue)
{
    RetValue iRet = func (iValue);

    if (iRet.iStat == 1) {
        /* do things with iRet.iValue, the returned value */
    }
    return 0;
}
```

または、のようにできます。

```
int usingFunc (int iValue)
{
    RetValue iRet;

    if ( (iRet = func (iValue)).iStat == 1 ) {
        /* do things with iRet.iValue, the returned value */
    }
    return 0;
}
```

オンラインでのパラメータをむ https://riptutorial.com/ja/c/topic/1006/のパラメータ

63: ポインタ

き

ポインタは、データのわりにをすポインタです。これらは、にびされるのをにするためにできます。

- returnType* nameparameters
- typedef returnType* nameparameters
- typedef returnTypeのパラメータ。
  *;
- typedef returnTypeのパラメータ。
  typedef* NamePtr;

Examples

ポインタのりて

```
#include <stdio.h>

/* increment: take number, increment it by one, and return it */
int increment(int i)
{
    printf("increment %d by 1\n", i);
    return i + 1;
}

/* decrement: take number, decrement it by one, and return it */
int decrement(int i)
{
    printf("decrement %d by 1\n", i);
    return i - 1;
}

int main(void)
{
    int num = 0;          /* declare number to increment */
    int (*fp)(int);       /* declare a function pointer */

    fp = &increment;      /* set function pointer to increment function */
    num = (*fp)(num);     /* increment num */
    num = (*fp)(num);     /* increment num a second time */

    fp = &decrement;      /* set function pointer to decrement function */
    num = (*fp)(num);     /* decrement num */
    printf("num is now: %d\n", num);
    return 0;
}
```

からポインタをす

```
#include <stdio.h>

enum Op
{
  ADD = '+',
  SUB = '-',
};


/* add: add a and b, return result */
int add(int a, int b)
{
    return a + b;
}

/* sub: subtract b from a, return result */
int sub(int a, int b)
{
    return a - b;
}

/* getmath: return the appropriate math function */
int (*getmath(enum Op op))(int,int)
{
    switch (op)
    {
        case ADD:
            return &add;
        case SUB:
            return &sub;
        default:
            return NULL;
    }
}

int main(void)
{
    int a, b, c;
    int (*fp)(int,int);

    fp = getmath(ADD);

    a = 1, b = 2;
    c = (*fp)(a, b);
    printf("%d + %d = %d\n", a, b, c);
    return 0;
}
```

ベストプラクティス

typedefの

ポインタをするのではなく、 `typedef`をうほうがかもしれません。

ポインタの`typedef`をするはのとおりです。

```
typedef returnType (*name)(parameters);
```

のような`compare`へのポインタをする`sort` があることをします。

> compare - ソートにされる2つのの。
>
> "compare"は、2つのがしいとみなされると0をすことがされ、のがあるでは "きい"はのをし、そうでないはがのをします。よりも「さい」。

`typedef`なければ、のようにポインタをのとしてします。

```
void sort(int (*compare)(const void *elem1, const void *elem2)) {
    /* inside of this block, the function is named "compare" */
}
```

`typedef` 、のようにします。

```
typedef int (*compare_func)(const void *, const void *);
```

`sort`のシグネチャをの`sort`にすることができます。

```
void sort(compare_func func) {
    /* In this block the function is named "func" */
}
```

どちらの`sort`も、フォームののをけれます

```
int compare(const void *arg1, const void *arg2) {
    /* Note that the variable names do not have to be "elem1" and "elem2" */
}
```

ポインタは、のポインタプロパティをめるべきのです。えば、 `typedef struct something_struct *something_type`ようなをしようとしないでください。これは、stdio.h `FILE`これはポインタではないことにづくでしょうのような、APIびしがアクセスするはずのメンバーをつにもてはまります。

コンテキストポインタの。

ポインタは、ほとんどの、ユーザがしたvoid *をコンテキストポインタとしてります。

```
/* function minimiser, details unimportant */
double findminimum( double (*fptr)(double x, double y, void *ctx), void *ctx)
{
    ...
```

```
    /* repeatedly make calls like this */
    temp = (*fptr)(testx, testy, ctx);
}

/* the function we are minimising, sums two cubics */
double *cubics(double x, double y, void *ctx)
{
    double *coeffsx = ctx;
    double *coeffsy = coeffx + 4;

    return coeffsx[0] * x * x * x + coeffsx[1] * x * x + coeffsx[2] * x + coeffsx[3] +
           coeffsy[0] * y * y * y + coeffsy[1] * y * y + coeffsy[2] * y + coeffsy[3];

}

void caller()
{
    /* context, the coefficients of the cubics */
    double coeffs[8] = {1, 2, 3, 4, 5, 6, 7, 8};
    double min;

    min = findminimum(cubics, coeffs);
}
```

コンテキストポインタをすることは、なパラメータが、しされたにハードコードされるはなく、グローバルをとすることをする。

ライブラリ`qsort()`はこのにわず、なのためのなしにげることができます。しかし、もっとなものについては、コンテキストポインタがになります。

もしてください

ポインタ

き

`char`と`int`ように、はCのなです。そのため、ポインタをすることができます。つまり、 をびすためにどのをすことができます。あなたがっていたたとえば、 `graph()`グラフをするを、あなたはにグラフするためにどのすことができます`graph()`。

```
// A couple of external definitions to make the example clearer
extern unsigned int screenWidth;
extern void plotXY(double x, double y);

// The graph() function.
// Pass in the bounds: the minimum and maximum X and Y that should be plotted.
// Also pass in the actual function to plot.
void graph(double minX, double minY,
           double maxX, double maxY,
           ???? *fn) {            // See below for syntax

    double stepX = (maxX - minX) / screenWidth;
    for (double x=minX; x<maxX; x+=stepX) {
```

```
        double y = fn(x);         // Get y for this x by calling passed-in fn()

        if (minY<=y && y<maxY) {
            plotXY(x, y);         // Plot calculated point
        } // if
    } for
} // graph(minX, minY, maxX, maxY, fn)
```

したがって、のコ―ドは、あなたがしたをグラフにします - そのがのをたしているり、つまり `double`インをして`double`アウトします。 `sin()` 、 `cos()` 、 `tan()` 、 `exp()`などくのがありますが、`graph()`のようなくのはありません

では、どのを`graph()`すことができないのか、どのをせないのかをどのようにしますかのは、みやすくしにくいをすることです。

```
double (*fn)(double); // fn is a pointer-to-function that takes a double and returns one
```

のは、にしようとする2つのことがありますのとそれがポインタであるというです。したがって、2つのをしてくださいしかし、 `typedef`をうことで、みやすくしやすいよりいをすることができます。

ポインタをくためのニ―モニック

すべてのCは、には、コ―ドがするプログラムメモリのへのポインタです。ポインタのなは、のに "コ―ルバック"をすることまたはクラスとオブジェクトをシミュレ―トすることです。

このペ―ジでさらにされているのはのとおりです。

returnType* nameparameters

ポインタをくためのニ―モニックは、のです。

1. のをくことからめ`returnType name(parameters)` `returnType name(parameters)`
2. ポインタのでをラップします `returnType (*name)(parameters)`

int 、 **char** 、 **float** 、 **array / string** 、 **struct**などへのポインタをつことができるように、へのポインタをつことができます。

ポインタの は、のり、の 、の/パラメ―タをけります。

のをしてしたとします

```
int addInt(int n, int m){
    return n+m;
}
```

このへのポインタをしてすることができます

```
int (*functionPtrAdd)(int, int) = addInt; // or &addInt - the & is optional
```

voidがある、のようになります。

```
void Print(void){
    printf("look ma' - no hands, only pointers!\n");
}
```

ポインタをすると、のようになります。

```
void (*functionPtrPrint)(void) = Print;
```

にアクセスするには、ポインタのがです。

```
sum = (*functionPtrAdd)(2, 3); //will assign 5 to sum
(*functionPtrPrint)(); //will print the text in Print function
```

このドキュメントのよりなにられるように、へのポインタをすることは、がいくつかのパラメータよりもくされるとになるがあります。じりのとじのパラメータをつへのポインタがいくつかあるは、 **typedef**コマンドをしてをき、コードをよりにすることをおめします。

```
typedef int (*ptrInt)(int, int);

int Add(int i, int j){
    return i+j;
}

int Multiply(int i, int j){
    return i*j;
}

int main()
{
    ptrInt ptr1 = Add;
    ptrInt ptr2 = Multiply;

    printf("%d\n", (*ptr1)(2,3)); //will print 5
    printf("%d\n", (*ptr2)(2,3)); //will print 6
    return 0;
}
```

ポインタのをすることもできます 。すべてのポインタがじ ""である

```
int (*array[2]) (int x, int y); // can hold 2 function pointers
array[0] = Add;
array[1] = Multiply;
```

あなたはこことここでもっとぶことができます 。

さまざまなのポインタのをすることもですが、のにアクセスするがあるときにはがになります。
あなたはここでもっとぶことができます 。

オンラインでポインタをむ https://riptutorial.com/ja/c/topic/250/ポインタ

クレジット

S. No		Contributors
1	Cをいめる	4444, Abhineet, Alejandro Caro, alk, Ankush, ArturFH, Bahm, bevenson, bfd, Blacksilver, blatinox, bta, chqrlie, Community, Dair, Dan Fairaizl, Daniel Jour, Daniel Margosian, David G., David Grayson, Donald Duck, Dov Benyomin Sohacheski, Ed Cottrell, employee of the month, EOF, EsmaeelE, Frosty The DopeMan, Iskar Jarak, Jens Gustedt, John Slegers, JonasCz, Jonathan Leffler, Juan T, juleslasne, Kusalananda, Leandros, LiHRaM, Lundin, Malick, Mark Yisri, MC93, MoultoB, msohng, Myst, Narox Nox, Neal, Nemanja Boric, Nicolas Verlet, OiciTrap, P.P., PSN, Rakitić, RamenChef, Roland Illig, Ryan Hilbert, Shoe , Shog9, skrtbhtngr, sohnryang, stackptr, syb0rg, techydesigner, tlhIngan, Toby, vasili111, Vin, Vraj Pandya
2	- のと	Alejandro Caro, Jonathan Leffler, Roland Illig, Toby
3	2Dをにす	deamentiaemundi, Malcolm McLean, Shrinivas Patgar, Toby
4	Typedef	Buser, Chandrahas Aroori, GoodDeeds, Jonathan Leffler, mame98, PhotometricStereo, Stephen Leppik, Toby
5	Valgrind	abacles, alk, Ankush, Chandrahas Aroori, Devansh Tandon, drov, Firas Moalla, J F, Jonathan Leffler, vasili111
6	Xマクロ	Cimbali, Jens Gustedt, John Bollinger, Leandros, MD XF, mpromonet, poolie, RamenChef, technosaurus, templatetypedef , Toby
7	アサーション	2501, AShelly, Blagovest Buyukliev, bta, eush77, greatwolf, J Wu, Jens Gustedt, Jonathan Leffler, Jossi, jxh, Leandros, Malcolm McLean, Ryan Haining, stackptr, syb0rg, Tim Post, Toby
8	アトミック	Jens Gustedt
9	インラインアセンブリ	bevenson, EsmaeelE, Jonathan Leffler
10	インライン	Alex, EsmaeelE, Jens Gustedt, Jonathan Leffler, Toby
11	エイリアシングとなタイプ	2501, 4386427, Jens Gustedt

12	エラー	Jens Gustedt, stackptr
13	コマンドライン	4386427, A B, alk, drov, dvhh, Jonathan Leffler, Malcolm McLean, Shog9, syb0rg, Toby, Woodrow Barlow, Yotam Salmon
14	コメント	Ankush, Chandrahas Aroori, Jonathan Leffler, Toby
15	シーケンスポイント	2501, Armali, bta, Community, haccks, Jens Gustedt, John Bode, Toby
16	ジャンプ	alk, Jens Gustedt, Jonathan Leffler, lordjohncena, Malcolm McLean, Sourav Ghosh, syb0rg, Toby
17	ストレージクラス	alk, Blagovest Buyukliev, Chrono Kitsune, greatwolf, Jean-Baptiste Yunès, Jens Gustedt, Jonathan Leffler, L.V.Rao, madD7, Neui, Nitinkumar Ambekar, P.P., Toby, tversteeg, vuko_zrno
18	スレッドネイティブ	alk, Jens Gustedt, P.P.
19	データ	2501, alk, Blagovest Buyukliev, Firas Moalla, Jens Gustedt, Keith Thompson, Ken Y-N, Leandros, P.P., Peter, WMios
20	テストフレームワーク	Community, EsmaeelE, Jonathan Leffler, lordjohncena, Toby, user2314737, vuko_zrno
21	ビットフィールド	alk, EvilTeach, Fantastic Mr Fox, haccks, Ishay Peled, Jens Gustedt, John Odom, Jonathan Leffler, Lundin, madD7, Paul Hutchinson, RamenChef, Rishikesh Raje, Toby, vkgade
22	ファイルとストリーム	alk, bevenson, EWoodward, haccks, iRove, Jean Vitor, Jens Gustedt, Jonathan Leffler, Jossi, Leandros, Malcolm McLean, Pedro Henrique A. Oliveira, RamenChef, reshad, Snaipe, stackptr, syb0rg, tkk, Toby, tversteeg, William Pursell
23	ブール	alk, Bob__, Braden Best, Chrono Kitsune, dhein, Insane, Jens Gustedt, Magisch, Mateusz Piotrowski, Peter, Toby
24	フォーマットされた/	alk, fluter, Jonathan Leffler, Jossi, lardenn, MikeCAT, polarysekt , StardustGogeta
25	プリプロセッサとマクロ	Alex Garcia, alk, bevenson, bwoebi, Dariusz, DrPrItay, Erlend Graff, EsmaeelE, EvilTeach, fastlearner, Firas Moalla, gman, hashdefine, hlovdal, javac, Jens Gustedt, Jonathan Leffler, Justin, Leandros, luser droog, Madhusoodan P, Maniero, mnoronha, Nitinkumar Ambekar, P.P., Paul J. Lucas, Peter, Richard Chambers, Robert Baldyga, stackptr, Toby, v7d8dpo4

26	プロセスIPC	cLdseed, EsmaeelE, Jonathan Leffler, Toby
27	ヘッダファイルのとインクルード	4444, Jonathan Leffler, patrick96, Sirsireesh Kodali
28	ポインタ	0xEDD1E, alk, Altece, Amani Kilumanga, Andrey Markeev, Ankush, Antti Haapala, Ashish Ahuja, Bjorn A., bruno, bta, chqrlie, Courtney Pattison, Dair, Daniel Porteous, David G., dhein, dkrmr, Don't You Worry Child, e.jahandar, elslooo, EOF, erebos, Faisal Mudhir, Fantastic Mr Fox, FedeWar, Firas Moalla , fluter, foxtrot9, Gavin Higham, gdc, Giorgi Moniava, gsamaras, haccks, haltode, Harry Johnston, Hemant Kumar, honk, Jens Gustedt, Jonathan Leffler, Jonnathan Soares, Josh de Kock, jpX , L.V.Rao, LaneL, Leandros, Luiz Berti, Malcolm McLean, Matthieu, Michael Fitzpatrick, MikeCAT, Neui, Nitinkumar Ambekar, OiciTrap, P.P., Pbd, Peter, RamenChef, raymai97, Rohan, Sergey, Shahbaz, signal, slugonamission, solomonope, someoneigna, Spidey, Srikar, stackptr, syb0rg, tbodt, the sudhakar, thndrwrks, Toby, Vality, vijay kant sharma, Vivek S, Wyzard, xhienne, Алексей Неудачин
29	マルチスレッド	Parham Alvani, Toby
30	メモリ	4386427, alk, Anderson Giacomolli, Andrey Markeev, Ankush, Antti Haapala, Cullub, Daksh Gupta, dhein, dkrmr, doppelheathen, dvhh, elslooo, EOF, EsmaeelE, Firas Moalla, fluter, gdc, greatwolf, honk, Jens Gustedt, Jonathan Leffler, juleslasne, Luiz Berti, madD7, Malcolm McLean, Mark Yisri, Matthieu, Neui, P.P., Paul Campbell, Paul V, reflective_mind, Seth, Srikar, stackptr, syb0rg, Tamarous, tbodt, the sudhakar, Toby, tofro, Vivek S, vuko_zrno, Wyzard
31	リンクされたリスト	4386427, alk, Andrea Biondo, bevenson, iRove, Jonathan Leffler, Jossi, Leandros, Mateusz Piotrowski, Ryan, Toby
32	なCプログラミングのイディオムとの	Chandrahas Aroori, Jonathan Leffler, Nityesh Agarwal, Shubham Agrawal
33	な	2501, Jens Gustedt, Sun Qingyao
34		dylanweber, ganchito55, haccks, hexwab, Jonathan Leffler, Leandros, Malcolm McLean, MikeCAT, Toby
35		3442, alk, Dariusz, Jens Gustedt, Leandros, mirabilos
36	のとし	abacles, Accepted Answer, alk, bevenson, Bjorn A., Chrono Kitsune, clearlight, Community, Dmitry Grigoryev, Dreamer, Dunno, FedeWar, Fred Barclay, Gavin Higham, Giorgi Moniava,

		hlovdal, Ishay Peled, Jeremy, John Hascall, Jonathan Leffler, Ken Y-N, Leandros, Lord Farquaad, MikeCAT, P.P., Roland Illig , rxantos, Sourav Ghosh, stackptr, Tamarous, techEmbedded, Toby, Waqas Bukhary
37		Alejandro Caro, alk, jasoninnn, Jens Gustedt, Jonathan Leffler, OznOg, Toby
38		Jonathan Leffler, Liju Thomas, P.P.
39		Armali, Toby, Vality
40		EsmaeelE, Jonathan Leffler, L.V.Rao, madD7, RamenChef, Sirsireesh Kodali, Toby
41	ステートメント/ループfor、while、do-while	alk, GoodDeeds, Jens Gustedt, jxh, L.V.Rao, Malcolm McLean, Nagaraj, RamenChef, reshad, Toby
42	の	2501, Blacksilver, eush77, Jean-Baptiste Yunès, Jonathan Leffler, Leandros, mirabilos, syb0rg, Toby
43		alk, Blagovest Buyukliev, Jens Gustedt, Jesferman, madD7, tversteeg
44	のるい	Jens Gustedt, John Bollinger, P.P.
45		alk, AnArrayOfFunctions, Blacksilver, Firas Moalla, J Wu, Jens Gustedt, Jonathan Leffler, Jonathon Reinhart
46	と	Ashish Ahuja, foxtrot9, Kerrek SB, Toby
47	、、のリテラル	Jens Gustedt, Jonathan Leffler, Klas Lindbäck, Neui, Paul92, Toby
48		4386427, alk, Amani Kilumanga, Andrey Markeev, bevenson, catalogue_number, Chris Sprague, Chrono Kitsune, Cody Gray, Damien, Daniel, depperm, dylanweber, FedeWar, Firas Moalla, haccks, Ishay Peled, jasoninnn, Jean-Baptiste Yunès, Jens Gustedt, John Bollinger, Jonathan Leffler, Leandros, Malcolm McLean, mantal, MikeCAT, P.P., Purag, Roland Illig, stackptr, still_learning, syb0rg, Toby, vasili111, Waqas Bukhary, Wolf, Wyzard, Алексей Неудачин
49	およびな	alk, Firas Moalla, Jens Gustedt, Jeremy Thien, kdopen, Lundin, Toby
50	の	2501, Abhineet, Aleksi Torhamo, alk, Antti Haapala, Armali, Ben Steffan, blatinox, bta, BurnsBA, caf, Christoph, Cody Gray,

		Community, cshu, DaBler, Daniel Jour, DarkDust, FedeWar, Firas Moalla, Giorgi Moniava, gsamaras, haccks, hmijail, honk, Jacob H, Jean-Baptiste Yunès, Jens Gustedt, John, John Bollinger, Jonathan Leffler, Kamiccolo, Leandros, Lundin, Magisch, Mark Yisri, Martin, MikeCAT, Nemanja Boric, P.P., Peter, Roland Illig, TimF, Toby, tversteeg, user45891, Vasfed, void
51		alk, Chrono Kitsune, Damien, Elazar, EsmaeelE, Faisal Mudhir, Firas Moalla, gmug, jasoninnn, Jens Gustedt, Jonathan Leffler, Jossi, kamoroso94, Madhusoodan P, OznOg, Paul Kramme, PhotometricStereo, RamenChef, Toby, Vality
52	のパディングとパッキング	EsmaeelE, Jarrod Dixon, Jedi, Jesferman, Jonathan Leffler, Liju Thomas, MayeulC, tilz0R
53	な	Alejandro Caro, alk, Blagovest Buyukliev, immerhart, Jonathan Leffler, manav m-n, Toby
54		202_accepted, 3442, alk, Amani Kilumanga, Andrea Corbelli, Bakhtiar Hasan, BenG, blatinox, cpplearner, Damien, Dariusz, EsmaeelE, Faisal Mudhir, Fantastic Mr Fox, Firas Moalla, gsamaras, hrs, Iwillnotexist Idonotexist, Jens Gustedt, Jonathan Leffler, kdopen, Ken Y-N, L.V.Rao, Leandros, LostAvatar, Magisch, MikeCAT, noamgot, P.P., Paul92, Peter, stackptr, Toby, Will, Wolf, Yu Hao
55		Jossi, RamenChef, Toby, Vality
56		alk, Amani Kilumanga, bevenson, Blacksilver, Firas Moalla, haccks, Ishay Peled, Jean-Baptiste Yunès, Jens Gustedt, Jonathan Leffler, Jossi, jxh, MC93, MikeCAT, nathanielng, P.P., Qrchack, R. Joiny, syb0rg, Toby, tofro, Turtle, Vraj Pandya, Алексей Неудачин
57	リテラル	alk, haccks, Jens Gustedt, Kerrek SB
58	シーケンス	Jonathan Leffler, PassionInfinite, Toby
59	スコープ	embedded_guy, Firas Moalla, Jean-Baptiste Yunès, Jens Gustedt, Jonathan Leffler
60	ステートメント	alk, bevenson, Blagovest Buyukliev, Faisal Mudhir, GoodDeeds, gsamaras, jxh, L.V.Rao, lordjohncena, MikeCAT, NeoR, noamgot, OznOg, P.P., Toby, tofro
61		2501, alk, AnArrayOfFunctions, AShelly, cdrini, cSmout, Dariusz , Elazar, Eli Sadoff, Firas Moalla, Guy, Iskar Jarak, Jasmin Solanki, Jens Gustedt, John Bollinger, Jonathan Leffler,

		L.V.Rao, Leandros, Liju Thomas, lordjohncena, Magisch, mhk, OznOg, Ray, Ryan Haining, Ryan Hilbert, stackptr, Toby, Waqas Bukhary
62	のパラメータ	2501, Alejandro Caro, alk, Chrono Kitsune, ganesh kumar, George Stocker, Jens Gustedt, Jonathan Leffler, Leandros, MikeCAT, Minar Ashiq Tishan, P.P., RamenChef, Richard Chambers, someoneigna, syb0rg, Toby
63	ポインタ	Alejandro Caro, alk, David Refaeli, Filip Allberg, hlovdal, John Burger, Leandros, Malcolm McLean, P.P., Srikar, stackptr, Toby