배우기

C Language

#c

1: C

C , , . C . .

. C . .

C Bell Labs 1969 1973 Dennis Ritchie Unix . C .

C . . .

- GCC, GNU
- clang : LLVM C
- MSVC, Microsoft Visual C / C ++

.

- Microsoft Visual C
- GCC

C

C C99 . , 2015 MSVC C99 (:) C (: `<tgmath.h>`) . " " . Wikipedia .

(GCC) C C . . .

C .

C .

( ) :

C (,) . . . , , , , .

( ). SO . . . .

K & R , Allman , GNU . . , Allman Allman Allman-8 . Wikipedia . . GNU GNU

UpperCamelCase, lowerCamelCase, lower_case_with_underscore, ALL_CAPS . (: ALL_CAPS)

K & R SO Pico .

C API ( ) :

- POSIX API (: PThreads , ,)

		1978-02-22
C89	ANSI X3.159-1989	1989-12-14
C90	ISO / IEC 9899 : 1990	1990-12-20
C95	ISO / IEC 9899 / AMD1 : 1995	1995-03-30
C99	ISO / IEC 9899 : 1999	1999-12-16
C11	ISO / IEC 9899 : 2011	2011 12 15

Examples

"Hello, World" C (: `hello.c` - `.c`) .

hello.c

```
#include <stdio.h>

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

Coliru

.

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

`stdio.h` . , . `stdio.h` `puts()` .

.

```
int main(void)
```

. (`main`), (`void` ,) (`int`) . `main()` .

```
{
    …
}
```

. , .

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

puts() ( ) . .

"Hello, World" . C "…" .

.

C (;) .

```
    return 0;
```

main() int ., . 0 . return 0; return 0; .

Linux vim gedit Windows Notepad . Visual Studio Code Sublime Text .

RTF .

(hello.c) (: Unix / Linux hello , Windows hello.exe) . C .

GCC

GCC (GNU Compiler Collection) C . .

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

(hello.c) -o (hello) . .

-Wall -Wextra -Werror . .

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

clang

clang .

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

clang GCC .

Microsoft C

Visual Studio Windows Microsoft cl.exe C hello.exe (cl /W3 GCC -Wall etc clang).

```
cl hello.c
```

./hello . , Hello, World , .

"Hello, World!" K & R C

"Hello, World!". C (C) "K & R" :

```
#include <stdio.h>

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

C (1978) C90 .

main() return K & R . C89 . C89 main int K & R . C99 C99 5.1.2.2.3 main (main) return . 0 , .

main int main (void) , int main(int argc, char **argv) .

C90 §5.1.2.2.3

> main exit main . main .

C90 §6.6.6.4 **return**

> return . } return .

C99 §5.1.2.2.3

> main int main exit main ; } main 0 . int .

C : https://riptutorial.com/ko/c/topic/213/c--

2: -

Examples

```
#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` , . .

- `int` . `EOF` ( ) .
- (: `isspace`) *`unsigned char`* *`EOF`* . `fgetc` `fgetc` .
- 0 (`false`) 0 ( ) `true` . 1 0 . `!!` .

```
#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` , . .

- (: `isspace`) *unsigned char EOF* .
- `*p` `char` .
- `char` `signed char` `unsigned char` `signed char` .
- `char` `unsigned char` `char` , `char` `unsigned char` .
- `char` `signed char` `unsigned char` . , .
- 0 (`false`) 0 () `true` . 1 0 . `!!` .

`ctype.h` C . .

. `int` *EOF* char .

'is' . (TRUE) . 0 (FALSE) .

C .

```
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. */
```

. 'to' . . 0 0 .

C .

```
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);
```

cplusplus.com 127 ASCII (a 0) .

ASCII		iscntrl	isblank					isdigit	isxdigit		ispunct
0x00 .. 0x08	NUL ( )	•									
0x09	('\ t')	•	•	•							
0x0A .. 0x0D	(: '\ f', '\ v', '\ n', '\ r')	•		•							
0x0E .. 0x1F	( )	•									
0x20	('')		•	•							
0x21 .. 0x2F	! "# $ % & '() * +, -. /										•
0x30 .. 0x39	0123456789							•	•	•	
0x3a .. 0x40	:; <=>? @										•
0x41 .. 0x46	ABCDEF				•		•		•	•	
0x47 .. 0x5A	GHIJKLMNOPQRSTUVWXYZ				•		•			•	
0x5B .. 0x60	[] ^ _`										•
0x61 .. 0x66	abcdef					•	•		•	•	
0x67						•	•			•	

ASCII		iscntrl	isblank					isdigit	isxdigit		ispunct
.. 0x7A											
0x7B .. 0x7E	{} ~										•
0x7F	(DEL)	•									

- : https://riptutorial.com/ko/c/topic/6846/-ctype-h-------

3: 2D

Examples

2D .

2 .

```
#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]);
    }
  }
}
```

GCC 4.9.4 .

```
$ 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);
```

: . . 2 C .

```
#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]);
    }
  }
}
```

,   .

```
#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

C (ISO / IEC 9899 : 1999, ISO / IEC 9899 : 2011) VLA (TODO :)   ,   C   (TODO : MS Visual Studio   ).

```
#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]);
    }
  }
}
```

, :

```
$ 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]);
```

`void fun1(int **a, int rows, int cols)` . , .

```
$ 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);
```

. .

```
#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)));
    }
  }
}
```

fun1 . fun1 . . .

```
#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]);
    }
  }
}
```

.   .

```
#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]);
    }
  }
}
```

2D

2D .

```
/* 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;
        }
    }
}
```

2D : https://riptutorial.com/ko/c/topic/6862/2d---

4: Typedef

typedef . . typedef , ( ) .

C typedef ''. static extern .

- typedef existing_name alias_name;

Typedef

typedef C .

typedef

typedef .

:

```
#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
```

typedefs foo.h FOO_DEF . foo bar bar.h .

Typedef #define

#define typedef C .

- typedef #define .
- #define #define typedef .
- #define cptr char * cptr a, b; cptr a, b; typedef char *cptr; typedef char *cptr; cptr a, b; . #define b char typedef .

Examples

typedef

struct .

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

Person person;
```

, struct   struct   .

Person (struct Person)   .   .

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

:

```
typedef struct Person Person;

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

union typedef   .

```
typedef union Float Float;

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

float   .

typedef

:

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

.

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

/* use whenever needed */
```

```
ll foo;
mystruct object;
```

.

. `int` 2 4 . 4 .

`int` 2 , `long` 4 . `int` 4 , `long` 8 . ,

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

`int` `long` .

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

`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 */
```

`typedef` .

C99

`<stdint.h>` `<inttypes.h>` (`typedef`) `typedef` , . , `uint8_t` 8 . `int64_t` 64 . `uintptr_t` . . `uint_least16_t` (16) `int_fast32_t` (32) . `intmax_t` `uintmax_t` . .

`typedef` . `typedef` .

typedef

`typedef` . , .

```
#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);
}
```

typedef printer .

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

int printer_t . . :

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

.

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

typedef . .

.

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

typedef .

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

typedef .

<signal.h> signal . C .

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

. int int .

SigCatcher :

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

signal() .

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

(C) . signal , int SigCatcher SigCatcher - SigCatcher int .

`typedef`   `typedef` ,   . .

Typedef : https://riptutorial.com/ko/c/topic/2681/typedef

5: Valgrind

- valgrind <

Valgrind C . Valgrind `free()` . .

http://valgrind.org .

Examples

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

. , , malloc .

.

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

() valgrind --help http://valgrind.org/ .

-

malloc .

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

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

    s = malloc(26); // the culprint

    return 0;
}
```

valgrind .

`--leak-check=yes --tool=memcheck` .

```
$ 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==
```

(: GCC `-g`) .

.

Valgrind

Valgrind (file.c:line_no) . valgrind .

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

.

1. *I*

```
==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
```

. .

2.

```
==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)
```

, `valg.c` `main` 7 `printf()` `printf` .

3. .

```
==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 , *10* (a) `valg.c` *9* , *(7)* .

Valgrind : https://riptutorial.com/ko/c/topic/2674/valgrind

6: X-

X- / . , ( ). `X()` , .

X- `X()` . `X()` . "Do not Repeat Yourself"(DRY) `X()` .

`X()` X . (:). .

"X-macro" . .

.

- X .
- X .
- X . X .

X Dr. Dobbs (http://www.drdobbs.com/the-new-cx-macros/184401387) Randy Meyers [X-Macros] .

Examples

printfs X

```
/* 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 */
```

:

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

```
/* 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;
}
```

.

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

: X .

X "X" . "X" .

X . .

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

.

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

/* apply macro to the list of items */
ITEM_LIST(PRINTSTRING)
```

.

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

"X" X , (`PRINTSTRING` .

X-Macros , , .

X 4 enum

.

```
/* 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]());
```

---

enum .

. int enum .

```
/* 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[];
```

.

```
/* 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 */}
```

Chromium GPU .

X- : https://riptutorial.com/ko/c/topic/628/x--

7:

. . . . `struct` .

Examples

.

`int` `struct` :

```
struct point
{
    int x;
    int y;
};
```

`x` `y` `point` ( ).

:

```
struct point p;     // declare p as a point struct
p.x = 5;            // assign p member variables
p.y = 3;
```

. .

```
struct point p = {5, 3};
```

.

.

```
printf("point is (x = %d, y = %d)", p.x, p.y);
```

Typedef

`struct` `typedef` . :

```
typedef struct
{
    int x, y;
} Point;
```

:

```
struct Point
{
    int x, y;
};
```

:

```
Point point;
```

:

```
struct Point point;
```

.

```
typedef struct Point Point;

struct Point
{
    int x, y;
};
```

point . C ++ . struct .

typedef . , struct . POSIX ' stat

```
int stat(const char *pathname, struct stat *buf);
```

struct stat stat .

typedef struct . struct .

:

```
#include "bar.h"

struct foo
{
    bar *aBar;
};
```

typedef struct bar.h bar bar .

```
typedef struct bar bar;
```

bar.h bar .

Typedef .

struct (.) . struct . (->) . struct (" ").

```
#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);
    }
}
```

C99

.   .

```
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[];
};
```

.

```
/* 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));
```

sizeof .

. .

, . .

```
/* 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 */
```

malloc , calloc realloc . .

```
/* 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'

C99 . 'struct hack' 1 .

```
struct ex1
{
    size_t foo;
    int flex[1];
};
```

.

```
/* 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));
```

`flex` `sizeof(*pe1)` (`sizeof(struct ex1)`) `offsetof(struct ex1, flex)` `malloc` . `sizeof(*pe1)-sizeof(pe1->flex)` . 0 "" 1 . .

`FLEXMEMB_SIZE` .

```
#if __STDC_VERSION__ < 199901L
#define FLEXMEMB_SIZE 1
#else
#define FLEXMEMB_SIZE /* nothing */
#endif

struct ex1
{
    size_t foo;
    int flex[FLEXMEMB_SIZE];
};
```

`offsetof(struct ex1, flex)` ( ). :

```
struct ex1 *pe10 = malloc(offsetof(struct ex1, flex) + n * sizeof(pe10->flex[0]));
```

1 . .

```
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);
```

C . . .

```
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;
}
```

. . . .

```
/* 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);
```

C .

```
/* 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");
    }
}
```

.

```
/* 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;
}
```

: https://riptutorial.com/ko/c/topic/1119/

8:

C , DEC Alpha RISC ARM CPU .

CPU . .

.

(Eric Raymond) C .

Examples

C . . GCC `__attribute__((__packed__))` . 64 .

```
struct foo {
    char *p;  /* 8 bytes */
    char c;   /* 1 byte  */
    long x;   /* 8 bytes */
};
```

`8-byte` :

```
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)` 17 24 . 64 8 / `char c;` `char c;` 1 8 (,) 7 .

`packed` .

```
struct __attribute__((__packed__)) foo {
    char *p;  /* 8 bytes */
    char c;   /* 1 byte  */
    long x;   /* 8 bytes */
};
```

`sizeof(struct foo)` 17 .

.

- .
- .

ARM Cortex-M0 . CPU .

struct 32 struct .

```
struct test_32 {
    int a;      // 4 byte
    short b;    // 2 byte
    int c;      // 4 byte
} str_32;
```

struct 10 , sizeof(str_32) 12 .

. N (N 1, 2, 4, 8, 16 2) N N).

sizeof(int) == 4 sizeof(short) == 2 .

- int a; 0 ; 4.
- short b; 4 ; 2.
- 6 ; 2.
- int c; 8 ; 4.

struct test_32 12 . .

struct test_32 4 struct test_32 . malloc() , calloc() realloc() calloc() .

32 , , - (OS Mac OS X , i7) 64 x86_64 double A 4 ; 64 8 double .

: https://riptutorial.com/ko/c/topic/4590/---

9:

C , C ( ) C . . . . . C , ( ) .

. , . . .

C2011 . . 8 . . C .

- **1** (3.6 / 3) . 8 , `CHAR_BIT` .
- " "(3.10 / 1)
- (5.1.1.2/1) .
- 3 (5.1.1.2/1)
- (5.1.1.2/1) 5 .
- (5.1.1.3 / 1).
- (5.1.2.1/1) .
- (5.1.2.1 / 1)
- (5.1.2.1/2).
- `int main(int argc, char *arg[])` `int main(void)` (5.1.2.2.1 / 1) `main()` .
- `main()` (5.1.2.2.1 / 2) .
- 5.1.2.3 ( ) 7.21.3 () (5.1.2.3/7) " " ?
- (5.1.2.3 / 10) .
- (5.1.2.4/1).
- (5.2.1 / 1).
- (5.2.2 / 3) `char` .
- (5.2.4.2/1).
- (5.2.4.2.2 / 6).
- (5.2.4.2.2 / 8) `FLT_ROUNDS` .
- `FLT_ROUNDS` 3 -1 (5.2.4.2.2 / 8).
- (5.2.4.2.2 / 9) `FLT_EVAL_METHOD` .
- `FLT_EVAL_METHOD` -1 (5.2.4.2.2 / 9).

- (5.2.4.2.2 / 10) FLT_HAS_SUBNORM , DBL_HAS_SUBNORM LDBL_HAS_SUBNORM
- thread thread () (6.2.4 / 4)
- char (6.2.5 / 3).
- ( ) (6.2.5 / 4) .
- **char signed char unsigned char** (6.2.5 / 15) char unsigned signed 0 SCHAR_MIN CHAR_MIN .
- , ((6.2.6.1/2))
- **3** , (6.2.6.2/2).
- (6.2.8 / 1).
- (6.2.8 / 3)
- (6.2.8 / 4).
- (6.3.1.1/1).
- (6.3.1.3/3) .
- (6.3.1.4/2 ; 6.3.1.5/1 ; 6.4.4.2 / 3).
- 0 (6.3.2.3/5) .
- #pragma (6.4 / 4).
- (6.4.2.1/1) , , , .
- (6.4.2.1/5).
- (6.4.4.4/2 ; 6.4.4.4/10) .
- (6.4.4.4/11).
- , (6.4.5 / 5)
- 7 (6.4.5 / 6) .
- (6.4.7 / 2) .
- FP_CONTRACT (6.5 / 8)
- **sizeof _Alignof** (6.5.3.4/5)
- (6.5.6 / 9).
- (6.5.7 / 5) .

- `register` (6.7.1 / 6).
- `int` `unsigned int` `signed int` (6.7.2 / 5)
- `_Bool` , `signed int` `unsigned int` . (6.7.2.1/5).
- (6.7.2.1 / 11).
- (6.7.2.1/14).
- (6.7.2.2/4) .
- `volatile` -qualifed (6.7.3 / 7) "" ?
- `inline` (6.7.4 / 6).
- , (6.10.1 / 4) .
- `#include` (6.10.2 / 2-3) .
- `#include` (6.10.2 / 4) .
- `#include` (6.10.2 / 6).
- `#` (6.10.3.2/2) `\` `\` .
- `STDC` (6.10.6 / 1) pragma `#pragma` preprocessing .
- `__DATE__` `__TIME__` (6.10.8.1/1) .
- `__STDC_ISO_10646__` `wchar_t` (6.10.8.2/1).
- `__STDC_UTF_32__` `char32_t` (6.10.8.2/1).
- (7.2.1.1/2).
- (7.6 / 6).
- (7.6 / 8).
- (7.6 / 10).
- (7.6.1 / 2) .
- `fegetexceptflag()` (7.6.2.2/1) .
- `feraiseexcept()` "" " " (7.6.2.3/2) " " .
- `setlocale()` (7.11.1.1/3) `"C"` .
- `FLT_EVAL_METHOD` `0` , `1` `2` (7.12 / 2) `double_t` `float_t` `double_t` .
-

(7.12 / 6) .

- (7.12.1 / 2) `math.h` .
- (7.12.1 / 3) `math.h` .
- `math.h` `errno` `ERANGE` (7.12.1 / 6) .
- FP (7.12.2 / 2) .
- `fmod()` 0 0 (7.12.10.1/3)
- `remainder()` 0 0 (7.12.10.2/3)
- `remquo()` (7.12.10.3 / 2) .
- `remquo()` 0 0 (7.12.10.3/3)
- , (7.14 / 4).
- (7.14.1.1/3) .
- `SIGFPE` , `SIGILL` `SIGSEGV` (7.14.1.1/3).
- (7.14.1.1/6).
- `NULL` (7.19 / 3).
- (7.21.2 / 2)
- (7.21.2 / 3) null .
- (7.21.3 / 1) .
- (7.21.3 / 2)
- (7.21.3 / 3) .
- 0 (7.21.3 / 4).
- (7.21.3 / 8).
- (7.21.3 / 8).
- (7.21.3 / 10)
- `remove()` (7.21.4.1/2).
- `rename()` (7.21.4.2/2).
- `tmpfile()` (7.21.4.3/2).
- `freopen()`

(7.21.5.4/3) ?

I / O

- FP printf () - family (7.21.6.1/8) .
- `printf()` -family (7.21.6.1/8) .
- `- [` (7.21.6.2/12) `scanf()` .
- `scanf()` -family `p` (7.21.6.2/12).
- (7.21.9.1/2) `fgetpos()` `errno` .
- `fsetpos()` `errno` (7.21.9.3/2).
- `ftell()` `errno` (7.21.9.4/3).
- `strtod()` - NaN (7.22.1.3p4) .
- `strtod()` -family `errno` `ERANGE` (7.22.1.3/10).
- 0 (7.22.3 / 1) .
- `abort()` ( ) OS (7.22.4.1/2).
- `exit()` (7.22.4.4/5).
- _Exit `_Exit()` (7.22.4.5/2).
- `getenv()` (7.22.4.6/2) .
- `system()` (7.22.4.8/3).
- (7.27.1 / 1).
- `clock_t` `time_t` (7.27.1 / 4) .
- `clock()` (7.27.2.1/3) .
- `timespec_get()` (`TIME_UTC` , 7.27.2.5/3).
- "C" (7.27.3.5/7) `%Z` `strftime()` .

I / O

- FP `wprintf()` -family (7.29.2.1/8) .
- `wprintf()` -family (7.29.2.1/8) .
- `wscanf()` -family `- [` (7.29.2.2/12) .

- `wscanf()` -family `p` (7.29.2.2/12).
- `wstrtod()` - NaN (7.29.4.1.1 / 4) .
- `wstrtod()` -family `errno` `ERANGE` (7.29.4.1.1 / 10).

Examples

```
int signed_integer = -1;

// The right shift operation exhibits implementation-defined behavior:
int result = signed_integer >> 1;
```

```
// 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

```
// The allocation functions have implementation-defined behavior when the requested size
// of the allocation is zero.
void *p = malloc(0);
```

. . `int` `-1` .

```
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"); }
}
```

`&` " " .

: https://riptutorial.com/ko/c/topic/4832/--

10:

`rand()` . :

- `arc4random()` (OS X BSD)
- `random()` (Linux)
- `drand48()` (POSIX)

Examples

`rand()` 0 `RAND_MAX` (0 `RAND_MAX`).

`srand(int)` . `rand()` . `rand()` . .

`time(NULL)` `time(NULL)` . . .

`srand(1)` .

```
#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;
}
```

:

```
Random value between [0, 2147483647]: 823321433
```

:

C . `rand()` . **`rand()`** .

`rand()` .

? .

pcg-random.org PCG32 , RNG. .

```
#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);
}
```

.

```
#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;
}
```

0.0 1.0 p . N RAND_MAX .

```
#define uniform() (rand() / (RAND_MAX + 1.0))
```

0.0 ~ 1.0 ρ -

```
i = (int)(uniform() * N)
```

`i` 0 ~ N - 1 .

RAND_MAX `double` . `RAND_MAX + 1.0` `RAND_MAX + 1.0` . .

Xorshift

`rand()` *xorshift* George Marsaglia . xorshift . xorshift Wikipedia .

```
#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;
}
```

: https://riptutorial.com/ko/c/topic/365/-

11:

Examples

struct union

.

```
#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;
}
```

C ( ) .

C ( ) .

IEEE 754 "Fast Inverse Square Root". ( ) union ( ).

```
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);
}
```

.

. ,   .   . http://www.riptutorial.com/c/example/9399/using-unions-to-reinterpret-values .

.   m_1   m_2   m_2   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
```

: https://riptutorial.com/ko/c/topic/7645/

12:

trigraph . . .

(GCC `-trigraphs` , `-Wtrigraphs` `-Wall` trigraph).

C , , .

trigraph (`puts("What happened??!!");`).

Examples

`[ ] { } ^ \ | ~ #` C 1980 (: ISO 646) ASCII (: `£ #` , `{ } { } | \ Æ Å æ å ø Ø` , EBCDIC `~`). , C .

C . , .

`#` , `{ }` .

```
??=include <stdio.h>

int main()
??<
    printf("Hello World!\n");
??>
```

C .

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
}
```

?? =	#
?? /	\
?? '	^
?? (	[
??)	]
??!	|
?? <	{

??>	}
?? -	~ ~

trigraph `??/` , `??/` (: `'??/??/'` . ,).

Digraphs

C99

1994 . . . (:) .

.

```
#include <stdio.h>

int main()
<%
    printf("Hello %> World!\n"); /* Note that the string contains a digraph */
%>
```

:

```
#include <stdio.h>

int main()
{
    printf("Hello %> World!\n"); /* Note the unchanged digraph within the string. */
}
```

<:	[
:>	]
<%	{
%>	}
% :	#

: https://riptutorial.com/ko/c/topic/7111/---

13:

- char 1 , 8 1 8 ( ) . char <limits.h> CHAR_BIT . POSIX 1 8 .
- C (:) .

Examples

(int short long).

```
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 */
```

.

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

char signed unsigned unsigned signed . char signed char unsigned char ( ) .

(C) .

```
/* 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) */
```

signed . 16 0x 0X 8 0 . signed unsigned .

```
/* 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 = 65536l; /* l or L represent long int */
```

1 , , INT_MAX long , long long .

<limits.h> . ( ) .

CHAR_BIT	(byte)	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 / - (2 ^{15} - 1)
`SHRT_MAX`	`short int`	32,767 / ^{2} 15 1
`USHRT_MAX`	`unsigned short int`	2 65,535 ^{16} - 1
`INT_MIN`	`int`	-32767 / - (2 ^{15} - 1)
`INT_MAX`	`int`	32,767 / ^{2} 15 1
`UINT_MAX`	`unsigned int`	2 65,535 ^{16} - 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

`LLONG_MIN`	`long long int`	-9223372036854775807 / - $^{(263}$ - 1)
`LLONG_MAX`	`long long int`	9223372368547758 7 / 2 $^{63-1}$
`ULLONG_MAX`	`unsigned long long int`	18446744073709551615 / 2 ^{64} - 1

`char` `CHAR_MIN` `SCHAR_MIN` `SCHAR_MIN` `CHAR_MAX` `SCHAR_MIN` `SCHAR_MAX` . `char` `CHAR_MIN` 0 `CHAR_MAX` `UCHAR_MAX` `UCHAR_MAX` .

C99

C99 `<stdint.h>` . .

C 0 .

```
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' */
```

(.rodata). .

```
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! */
```

. ,   .

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 */
```

.

```
/* 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

`<stdint.h>` .   2 .

.

```
/* 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 float , double long double .

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

<float.h> .

. (arm, x86, x86_64, MIPS) IEEE 754 .

C 3 .

C .

.

- `*` " ";
- binary `[]` "array subscription";
- (1 + n) -ary `()` " ";
- `()`

.

`[]` ( )	1	
`()` ( )	1	
`*` ( )	2	

, . .

`thing[X]`	`X`
`thing(t1, t2, t3)`	`t1` , `t2` , `t3` ...
`*thing`	...

.

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

[] * . names char 20 .

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

* . place char 10 .

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

. fn long , short int .

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

() . fn void int .

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

() : fp void int .

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

. [] . arr int 8 5 .

```
int **ptr;
```

. . ptr int .

(* *) . .

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

.

- fn : void int .
- ptr : int .
- fp : int int ;
- arr : int 20 10 .
- num : int .

. .

```
/*
 * 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];
```

: https://riptutorial.com/ko/c/topic/309/-

14:

C11 `<threads.h>` . C `pthread.h` POSIX (pthreads) .

- thrd_t //
- int thrd_create (thrd_t * thr, thrd_start_t func, void * arg); // .
- int thrd_equal (thrd_t thr0, thrd_t thr1); //
- thr_t thrd_current (void); // .
- int thrd_sleep (const struct timespec * duration, struct timespec * remaining); // .
- void thrd_yield (void); //
- _Noreturn void thrd_exit (int res); // .
- int thrd_detatch (thrd_t thr; // .
- int thrd_join (thrd_t thr, int * res); // .

http://www.riptutorial.com/c/example/2622/data-race . C11 `mtx_lock()` ( ) http://www.riptutorial.com/c/topic/4924/atomics . `stdatomic.h` .

Examples

C11

```
#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);
}
```

: https://riptutorial.com/ko/c/topic/10489/-

15:

C `malloc()` , `calloc()` , `realloc()` `free()` . C99 `aligned_alloc()` . `alloca()` .

- void * aligned_alloc (size_t alignment, size_t size); / * C11 * /
- void * calloc (size_t nelements, size_t size);
- void free (void * ptr);
- void * malloc (size_t size);
- void * realloc (void * ptr, size_t size);
- void * alloca (size_t size); / * alloca.h . * /

(`malloc` , `realloc` `aligned_alloc`)	(). `aligned_alloc` , .
(`calloc`)	
nelements	
	`malloc` , `calloc` , `realloc` `aligned_alloc`

C11

`aligned_alloc()` C11 .

POSIX (: `posix_memalign()`) `mmap()` .

Examples

free () .

```
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` `free()` (libc OS) , `p` . . C . `p` .

`malloc()` , `calloc()` , `realloc()` `aligned_alloc()` `free()` . `strdup ()`). ,

- &
- ,

. .

.

. p p .

```
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);
}
```

(, } free() p . p .

.

```
free(p);
p = NULL;     // you may also use 0 instead of NULL
```

:

- : . .

  NULL . . .

  fail-fast .

- null . C free(NULL) .

  free ptr . , . ptr (NULL) . calloc , malloc realloc free realloc .

- (:)

C <stdlib.h> . .

```
int *p = malloc(10 * sizeof *p);
if (p == NULL)
{
    perror("malloc() failed");
    return -1;
}
```

10 `int` , S `malloc` (, `malloc`) `p` .

`sizeof` `sizeof` (`char` , `signed char` `unsigned char` , `sizeof` `1`).

`malloc` . .

`malloc()` `realloc()` `free()` `free()` .

, `int array[10];` `int array[10];` . `static` ( ). `static` . C .

`malloc` `memset` 0 . `calloc` `0` . 0 .

```
int *p = calloc(10, sizeof *p);
if (p == NULL)
{
    perror("calloc() failed");
    return -1;
}
```

`calloc` : ( ) `calloc()` net `malloc()` , `memset()` .

C11

C11 `aligned_alloc()` . `malloc()` `calloc()` . `malloc()` `calloc()` . `alignof(max_align_t)`). `aligned_alloc()` .

```
/* Allocates 1024 bytes with 256 bytes alignment. */
char *ptr = aligned_alloc(256, 1024);
if (ptr) {
    perror("aligned_alloc()");
    return -1;
}
free(ptr);
```

C11 . 1) ( ) 2) . .

. `void *realloc(void *ptr, size_t size)` `ptr` `size` . `ptr` `malloc` , `calloc` `realloc` ( ) . . . . `ptr` NULL, .

```
#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;
}
```

`*p` . `realloc` .

`realloc` `p` `temporary` . `realloc` null . .

C99

C99 C VLA . VLA `sizeof` VLA ( ) .

```
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);
}
```

`matrix` `double[m]` , `sizeof` `double[n][m]` `n` .

`free` to single call .

VLA . `[]` . `[]` . `sumAll` . C .

```
double sumAll(size_t n, size_t m, double (*A)[m]);
```

, `n` .

, `main` `argc+1` `argv` C .

VLA C11 C11 . `__STDC_NO_VLA__` `__STDC_NO_VLA__` .

realloc (ptr, 0) free (ptr) .

`realloc` `malloc + memcpy + free` .

0 `realloc` . `0` `size` . null .

`realloc(ptr,0)` `free(ptr)` .

- "" `ptr` .
- `free(ptr)` ,
- `free(ptr)` `0`
- `0` .

.

`realloc(ptr,0)` free / deallocate `free` .

`malloc()` . C (8).

. `free()` . , .

. . .

```
/* 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;
```

. . . 32 :

```
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;
}
```

.

alloca :

: `alloca` . ( ) . C VLA (*Variable Length Arrays)* .

```
#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
}
```

.

`alloca` `free` ( ).

`alloca` .

`free` ( ).

- `malloc` .
- free'd

- free , realloc ( )
- ( ).
- , malloc() .
- alloca() VLA ( )
- alloca() malloc() malloc()

- alloca() .

C99

.

```
void foo(int size) {
    char data[size];
    /*
      function body;
    */
    // data is automatically freed
}
```

alloca() , alloca() (alloca() . C99 __STDC_NO_VLA__ C11 .

: https://riptutorial.com/ko/c/topic/4726/-

16:

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

argc	- .
argv	argument vector - ( ) `char` -pointers () .

' '(- '') AC `main` . .

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

`argv` `char **argv` . . `argc` `argv` .

`main` `int main(void)` .

.

- `argc` .
- `argv` ( ) `char` -pointers () .
- "" . Unix `myprogram *.txt` . Windows " `*.txt` " .

: `argv` `argc` . `argc` `0` `argc` 0 `argv[0]` (`argv[argc]`) . . . `argv[0][0] == '\0'` .

.

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

`argc` `4` .

- `argv[0]` `"./some_program"` ( ) . `""` .
- `argv[1]` `"abba"` .
- `argv[2]` `"banana"` ,
- `argv[3]` `"mamajam"` ,
- `argv[4]` `NULL` .

C C ++ `main()` .

Examples

.

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

1. `argv` `char *argv[]` .
2. `argv[0]` ( ). "" `argv[1]` , `i` 1 .
3. print `argv[i]` `*(argv + i)` . .
4. . , , . ( ).

.

(`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 () .
- strtol

GNU getopt

C . (-) .

Linux Unix glibc getopt , .

GNU , POSIX .

GNU getopt   .

```
#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;
}
```

`gcc` :

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

(`--help` , `--file` `--msg`) . " " (`-h` , `-f` `-m`). "file" "msg" . "msg" .

.

- `--option=value` ( )
- `-ovalue` `-o"value"` ( )

: https://riptutorial.com/ko/c/topic/1285/--

17:

C . C- `'\0'` 1 .

, `"abc"` C `'a'` , `'b'` , `'c'` `'\0'` 4 .

.

- char str1 [] = ", !"; / * * /
- char str2 [14] = ", !"; / * * /
- char * str3 = "Hello, world!"; / * * /

Examples

: 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;
}
```

`len` . . `program_name "Hello, world!"` `program_name "Hello, world!"` , `The length of the second argument is 13.` `Hello, world!` `Hello, world!` 13 .

`strlen` NUL `'\0'` . NUL .

`strlen` ( ). ( ) . .

```
#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
```

: 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
```

NUL !

(: `"foo"`) NUL .

: strcmp (), strncmp (), strcasecmp (), strncasecmp ()

`strcase*` - C POSIX .

`strcmp` null . , 2 , 0, 3 2 .

```
#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` `strcasecmp` .

```
#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 strncasecmp n .

```
#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
```

: strtok (), strtok_r () strtok_s ()

strtok .

```
#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]
```

strtok .

strtok NULL NULL . . , strtok .

strtok . , src strtok . , const ( ). ID (: "," "!").

. .

strtok thread re-entrant . strtok , strtok strtok , strtok .

strtok .

```
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]
```

outer do while 10 ("1.2" , "3.5" , "4.2") . strtok ("1" , "2" , "3" , "5" , "4" , "2").

strtok . strtok "1.2 \ 0" "1" "2" . strtok NULL . src .

C11

C thread-safe re-entrant POSIX strtok_r . MSVC strtok strtok_s .

C11

C11 strtok_s Annex K . __STDC_LIB_EXT1__ . .

strtok_s POSIX strtok_r . strtok_s strtok_r .

strtok_s strtok_s .

```
/* 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);
```

.

```
[1.2]
 [1]
 [2]
[3.5]
 [3]
 [5]
[4.2]
 [4]
 [2]
```

/ : strchr (), strrchr ()

strchr strrchr NUL . strchr strrchr .

```
#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`) :

```
$ ./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` . `C:\Users\eak\myfile.txt` `myfile.txt` .

```
char *getFileName(const char *path)
{
    char *pend;

    if ((pend = strrchr(path, '\')) != NULL)
        return pend + 1;

    return NULL;
}
```

for .

```
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. */
}
```

`strlen()` .

```
size_t length = strlen(string);
size_t i = 0;
for (; i < length; i++) {
    printf("%c\n", string[i]);    /* Print each character of the string. */
}
```

, C null (`strlen()` `strlen()` .-)). .

```
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++;
}
```

C ('\ 0') .

. . , `"hello world"` . null .

. , `char *` .

```
char * string = "hello world";
```

`char *` . `string` `'h'` .

1 . `const`

```
char const * string = "hello world";
```

2 .

```
char const string_arr[] = "hello world";
```

.

```
char modifiable_string[] = "hello world";
```

.

```
char modifiable_string[] = {'h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\0'};
```

`'\0'` null .

1 string ( ).

2 . string char | string_arr | .

.

1. char * s
2. char

.

```
char * string_array[] = {
    "foo",
    "bar",
    "baz"
};
```

: char * . string_array / . , .

C main argv ( ) char * : char * argv[] .

. .

```
char modifiable_string_array_literals[][4] = {
    "foo",
    "bar",
    "baz"
};
```

.

```
char modifiable_string_array[][4] = {
    {'f', 'o', 'o', '\0'},
    {'b', 'a', 'r', '\0'},
    {'b', 'a', 'z', '\0'}
};
```

4 . null 4 .

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 haystack ( ) needle . strstr . needle NULL . zbpos . zbpos . findnext .

```
/*
    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" */
}
```

null char . . .

```
const char *get_hello() {
    return "Hello, World!";  /* safe */
}
```

const . . .

```
char *foo = "hello";
foo[0] = 'y';  /* Undefined behavior - BAD! */
```

- - const .

```
const char *foo = "hello";
/* GOOD: can't modify the string pointed to by foo */
```

. :

```
char *foo = "hello";
foo = "World!"; /* OK - we're just changing what foo points to */
```

char . . . const :

```
char foo[] = "hello";
foo[0] = 'y';  /* OK! */
```

0

memset ( ) 0 .

str 0 n .

```
#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'
''
```

:

```
#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 strcspn

strspn () . strcspn strcspn .

```
/*
  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;
}
```

wcsspn wcscspn . .

▪

= = C . C null = ( ).

```
#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;
}
```

`char *d` `char d[3]` . . C .

```
#include <stdio.h>

int main(void) {
    char a[] = "abc";
    char b[8];

    b = a; /* compile error */
    printf("%s\n", b);

    return 0;
}
```

`strcpy()`

, `strcpy()` `string.h` . .

```
#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()`

`snprintf()` . .

```
#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() strncat() . .

```
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 */
```

sizeof(dest) - 1 ; strncat() null () ( ) .

. :

```
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);
```

.

```
23: [Clownfish: Marvin and N]
```

, . . . strncat() null .

```
    strcpy(dst, "Clownfish: ");
    assert(len < sizeof(dst) - 1);
    strncat(dst + len, src, sizeof(dst) - len - 1);
    printf("%zu: [%s]\n", strlen(dst), dst);
```

strncat() dst .

strncpy()

strncpy() .   ,   :

1. strncpy()   null .
2. strncpy()   . null .

(UNIX   )

.

```
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 */
```

, strncpy() .

: atoi (), atof () (,)

: atoi , atol , atoll atof .   . (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;
}
```

10 .

```
$ ./atoi 100
String value = 100, Int value = 100
$ ./atoi 200
String value = 200, Int value = 200
```

.

```
$ ./atoi 0x200
0
$ ./atoi 0123x300
123
```

.

```
$ ./atoi hello
Formatting the hard disk...
```

atoi .

- long int atol() strtol() .
- double atof() strtod() .

C99

- long long int atoll() strtoll() .

/

.

```
int sprintf ( char * str, const char * format, ... );
```

sprintf float .

```
#include <stdio.h>
int main ()
{
  char buffer [50];
  double PI = 3.1415926;
  sprintf (buffer, "PI = %.7f", PI);
  printf ("%s\n",buffer);
  return 0;
}
```

.

```
int sscanf ( const char * s, const char * format, ...);
```

sscanf .

```
#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;
}
```

Number : strtoX

C99

C99 C . strtoX strtoX , X l , ul , d .

```
double strtod(char const* p, char** endptr);
long double strtold(char const* p, char** endptr);
```

.

```
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 */
```

strtod 0.0 .

endptr . . 0 NULL .

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 */
```

:

```
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);
```

nbase nbase .

```
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 0 C . 0x 16 , 0 8 10 .

```
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;
}
```

, 8 , 10 16 .

: https://riptutorial.com/ko/c/topic/1990/

18: / : for, while, do-while

- / * * /
- for ([], [], []) one_statement
- for ([expression]; [expression]; [expression]) {0 }
- while (expression) one_statement
- while (expression) {0 }
- while () while one_statement;
- while (expression) while { };
- // C99
- for (; [expression]; [expression]) one_statement;
- for (; [expression]; [expression]) {0 }

/ .

- /
- /

/

```
for ([<expression>]; [<expression>]; [<expression>]) <statement>
while (<expression>) <statement>
```

C99

```
for ([declaration expression]; [expression] [; [expression]]) statement
```

/

```
do <statement> while (<expression>);
```

Examples

For

. `for` . `n` `scanf()` `n` .

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]);
}
```

scanf() n (10) .

, i , . size_t ( ) .

for C99 . for .

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

while . while . 0 . 0 while true .

```
int num = 1;

while (num != 0)
{
    scanf("%d", &num);
}
```

Do-While

for while do-while . do while . do-while .

do-while 50 .

```
int num, sum;
num = sum = 0;

do
{
  scanf("%d", &num);
  sum += num;

} while (sum < 50);
```

do-while .

for

```
for ([declaration-or-expression]; [expression2]; [expression3])
{
    /* body of the loop */
}
```

for .

- declaration-or-expression . .

C99

. , for .

C99

C for for .

- expression2 . . true . . , update body . true . . , .
- expression3 *update* . . .

.

:

C99

```
for(int i = 0; i < 10 ; i++)
{
    printf("%d", i);
}
```

.

```
0123456789
```

i = 0 i . i < 10 true . i . i++ i 0 1, , . i 10 . i < 10 false .

. false .

:

C99

```
for (int i = 0; i >= 0; )
{
    /* body of the loop where i is not changed*/
}
```

i , iterator 0 . true . , i . , i 0 , false , .

true .

```
while (true)
{
    /* body of the loop */
}
```

for . , true , .

```
for (;;)
{
    /* body of the loop */
}
```

break true .

```
while (true)
{
    /* statements */
    if (condition)
    {
         /* more statements */
         break;
    }
}
```

. ***B*** . , ***A*** ***A*** ***B*** .

```
do_B();
while (condition) {
    do_A();
    do_B();
}
```

B / Duff 's Device switch fall through while .

```
switch (true) while (condition) {
case false: do_A(); /* FALL THROUGH */
default:    do_B(); /* FALL THROUGH */
}
```

. n .

```
do {
    *ptr++ ^= mask;
} while (--n > 0);
```

n 4 , :

```
do {
    *ptr++ ^= mask;
    *ptr++ ^= mask;
    *ptr++ ^= mask;
    *ptr++ ^= mask;
} while ((n -= 4) > 0);
```

Duff 's Device n 4 .

```
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);
```

.

/ : for, while, do-while : https://riptutorial.com/ko/c/topic/5151/------for--while--do-while

19:

("") . C . C .

C . C99 VLA .

- []; / * 'name' 'length' 'type' . * /
- int arr [10] = {0}; / * 0 . * /
- int arr [10] = {42}; / * 42 0 . * /
- int arr [] = {4, 2, 3, 1}; / * 4 . * /
- arr [n] = ; / * n . * /
- = arr [n]; / * n . * /

?

. , C (`char` s) "Hello, World!" . . , . . .

.

`sizeof` , `_Alignof` (C2011) `&` (address-of) () ("") . subscripting (`[]`) . expression `arr[idx]` `*(arr + idx)` . `*(arr + idx)` `*(idx + arr)` `idx[arr]` . `idx` `arr` ( ) .

, `&(arr[0])` `&*(arr + 0)` `arr` . . .

, `T[N]` (`&arr`) `T (*)[N]` . pointed-to x h .

.

```
void foo(int a[], int n);
void foo(int *a, int n);
```

`foo` `a` . `foo()` . . , . .

Examples

1 .

```
type arrName[size];
```

`type` `arrName` `size` .

(10 int) :

```
int array[10];
```

. 0 .

```
int array[10] = {0};
```

. 10 `int` . 3 `int` `1` , `2` , `3` 0.

```
int array[10] = {1, 2, 3};
```

. 0 . (ISO C99) . ,

```
int array[5] = {[2] = 5, [1] = 2, [4] = 9}; /* array is {0, 2, 5, 0, 9} */
```

, . .

```
int array[] = {1, 2, 3}; /* an array of 3 int's */
int array[] = {[3] = 8, [0] = 9}; /* size is 4 */
```

0 .

C99 C11

Variable Length Array ( ) (VLA) C99 C11 . . . . VLA . VLA .

```
size_t m = calc_length(); /* calculate array length at runtime */
int vla[m];               /* create array with calculated length */
```

:

VLA . `vla` . VLA .

()

0 .

```
#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;
}
```

`<string.h>` `memset()` . `array` .

```
memset(array, 0, ARRLEN * sizeof (int)); /* Use size explicitly provided type (int here). */
```

```
memset(array, 0, ARRLEN * sizeof *array); /* Use size of type the pointer is pointing to. */
```

`array` ( ). 0 .

```
 memset(array, 0, sizeof array); /* Use size of the array itself. */
```

. . .

```
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]);
```

(""). `sizeof` . , . .

, `int` . :

```
/* array will decay to a pointer, so the length must be passed separately */
int last = get_last(array, length);
```

.

```
int get_last(int input[], size_t length) {
    return input[length - 1];
}
```

`input` **`input`** **`input`** (`int`). `int *input` `input` . . ( ). .

. .

```
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 */
}
```

, . `clang` `clang` .

```
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[])
                     ^
```

.

```
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];
```

+ (->) .

```
*(array + 4) = 5;
*(4 + array) = 5;
```

:

```
array[4] = 5;
4[array] = 5; /* Weird but valid C ... */
```

.

```
val = array[4];
val = 4[array]; /* Weird but valid C ... */
```

C ( ).

```
int val;
int array[10];

array[4] = 5;    /* ok */
val = array[4];  /* ok */
array[19] = 20;  /* undefined behavior */
val = array[15]; /* undefined behavior */
```

```
#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;
}
```

```
#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;
}
```

`calloc()` `int` `n` . 0 .

`free()` .

C . . . `array[0][0]` 10000x10000 `array[0][1]` `sizeof(type)*10000` `array[1][0]` `array[0][0]` . ?

```
#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;
    }
}
```

.

```
#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;
    }
}
```

, (i j 2) :

```
#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;
    }
}
```

3 i, j k :

```
#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;
        }
    }
}
```

, **N1 x N2 x ... x Nd** , **d** **n1, n2, ..., nd** .

$$n_d + N_d \cdot (n_{d-1} + N_{d-1} \cdot (n_{d-2} + N_{d-2} \cdot (\cdots + N_2 n_1)\cdots))) = \sum_{k=1}^{d} \left(\prod_{\ell=k+1}^{d} N_\ell \right) n_k$$

/ : https://en.wikipedia.org/wiki/Row-major_order

C . .

```
type name[size1][size2]...[sizeN];
```

, 3 (5 x 10 x 4) .

```
int arr[5][10][4];
```

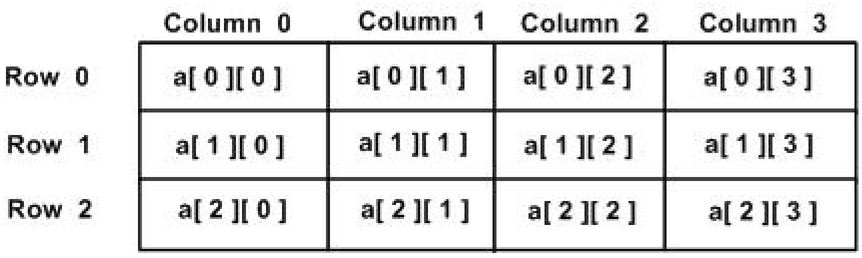
2

2 . 2 1 . mxn 2 .

```
type arrayName[m][n];
```

`type` C (`int` , `float`) `arrayName` C . 2 `m` `n` . : C . `int a[4][3]` `int a[3][4]` . C - .

3 4 2 `a` .



	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]

`a` `a[i][j]` . `a` , `i` `j` . 0 . 2-D .

2

. 4 3 .

```
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 */
};
```

. .

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

.

2

2 . -

```
int val = a[2][3];
```

. 2 .

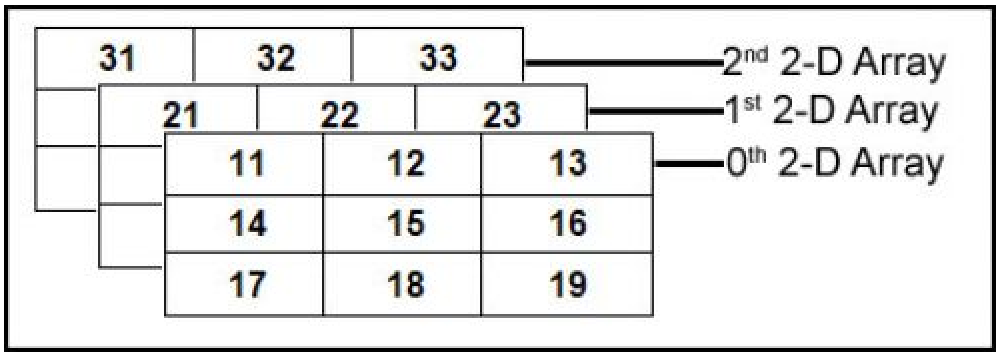
```
#include <stdio.h>

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;
}
```

.

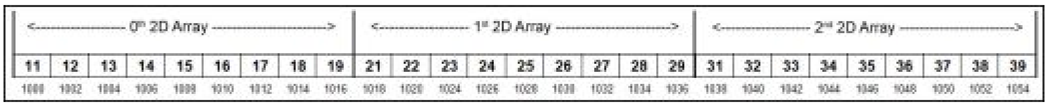
```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

3 :

3D . 2D , 2D 1D .



3D :



<-- 0th 2D Array -->									<-- 1st 2D Array -->									<-- 2nd 2D Array -->								
11	12	13	14	15	16	17	18	19	21	22	23	24	25	26	27	28	29	31	32	33	34	35	36	37	38	39
1000	1002	1004	1006	1008	1010	1012	1014	1016	1018	1020	1024	1026	1028	1030	1032	1034	1036	1038	1040	1042	1044	1046	1048	1050	1052	1054

3D :

```
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}}
};
```

.

```
#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;
}
```

`for` `p` a `a` .

`++p` `p` `*p` .

. decay-to-pointer, ( ) . .

```
#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);
}
```

: https://riptutorial.com/ko/c/topic/322/

20:

printf (printf , fprintf) . *varargs* .

`#include <stdarg.h>` .

, : `void err_exit(const char *format, ...);` .

- void va_start (va_list ap, *last*); / * . *last* ("...") . * /
- va_arg (va_list ap, *type*); / * ; . * /
- void va_end (va_list ap); / * * /
- void va_copy (va_list dst, va_list src); / * C99 : (:). * /

`va_list` ap	,
	. . `register` , .
	(, `int` `short` `int`)
`va_list` src	
`va_list` dst	

`va_start` , `va_arg` , `va_end` `va_copy` .

`va_start` , `va_end` , . .

(`register`) . .

`va_arg` `va_arg` .

- `short` `int` (`unsigned short` `int` `sizeof(unsigned short) == sizeof(int)` , `unsigned int`).
- `float` `double` .
- `signed char` `int` . `unsigned char` `sizeof(unsigned char) == sizeof(int)` `int` .
- `char` `int` .
- `uint8_t` `int16_t` C99 .

(, K & R) `<varargs.h>` . (`<stdarg.h>`) C89 . `va_copy` C99 .

Examples

count va_list .

. `printf()` `scanf()` .

( ) . . .

```
#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;
}
```

va_list .

. "" (`printf`) . .

```
/* 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 `execlp()` . `double` .

```
#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));
}
```

:

- ( ) - `0` `-1`
- - `NAN`
- - `NULL`
- -

`printf ()`

`printf()`   .   .

`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
```

.   . `errmsg()` `warnmsg()` `verrmsg()` `verrmsg()` .   `verrmsg()` `verrmsg()` . (-   YAGNI (- YAGNI)) .

`errmsg.c`

variadic `vfprintf()`   .   (`errno`)   .

```
#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

.

```
#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()` `read()` 1 . `close()` . .

`printf()`

GCC (GNU Compiler Collection GNU C) Clang `printf()` . . .

, GCC GCC Clang . GCC `__GNUC__` `__GNUC__` `__GNUC__` .

- `format` .

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
```

, :

```
errmsg(EXIT_FAILURE, errno, "Failed to open file '%d' for reading", filename);
```

(`%d` `%s`) :

```
$ 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
$
```

.

`printf()` , `char` , `int` `double` (10). `printf()` , . , .

```
#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);
}
```

: https://riptutorial.com/ko/c/topic/455/-

21:

- (type) {initializer-list}

C C11-§6.5.2.5 / 3 .

> . . 99)

99 :

> . **void** .

:

> const . 101)
>
> 101) .

.
C11-§6.5.2.5 / 13 :

> const . ,
>
> ```
> (const char []){"abc"} == "abc"
> ```
>
> 1 .

Examples

/

2 | . C99 . .

C , C11- §6.5.2.5 / 9 :

```
int *p = (int [2]){ 2, 4 };
```

`p` , 2 int .

lvalue. static ( ) ( ) , .

```
void f(void)
{
    int *p;
    /*...*/
    p = (int [2]){ *p };
    /*...*/
```

```
}
```

p 2 int . p , 0. [...]

p .

(C11)

```
  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});
```

drawline drawline struct point . x == 1 y == 1 x == 3 y == 4

```
int *p = (int []){ 1, 2, 3};
```

.

```
int *p = (int [10]){1, 2, 3};
```

0 .

---

lvalue . const (const int[]){1,2} .

C99 .

```
void foo()
{
    int *p;
    int i = 2; j = 5;
    /*...*/
    p = (int [2]){ i+j, i*j };
    /*...*/
}
```

: https://riptutorial.com/ko/c/topic/4135/-

22:

_Bool <stdbool.h> C C99 / C11 .

C89 typedef / define .

Examples

stdbool.h

C99

stdbool.h bool . true 1 false 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 _Bool . .

#define

C 0 true , 0 false . C99 _Bool bool #define C .

```
#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!");
    }
}
```

<stdbool.h> .

() _Bool

C99

C C99 _Bool C . 0 (*false*) 1 (*true*) .

```
#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 . if .

```
_Bool z = X;
```

- X () X == 0 z 0 . z 1 .
- X X z 0 1 .

bool , false true <stdbool.h> .

" " .

```
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 0 , 1 , 2 3 . , 0 "" else . "true" if .

```
double* A = malloc(n*sizeof *A);
if (!A) {
   perror("allocation problems");
   exit(EXIT_FAILURE);
}
```

A .

A == NULL .

```
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]);
}
```

.

```
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]);
}
```

. 0 .

typedef bool

#define enum .

```
#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;
```

C C .

typedef , enum

: https://riptutorial.com/ko/c/topic/3336/

23:

Examples

/ /

C '++' '-' . ++ -- . . ++ ++operand postfix operand++ . 1 . .

```
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 ; */
```

1 . .

```
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; */
```

-- .

.

```
    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;
    }
```

, .

C .

```
    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;
    }
```

. .

.

```
    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;
    }
```

: https://riptutorial.com/ko/c/topic/7094/

24:

C . . . . C . C .

- : size;

	signed unsigned , unsigned , int _Bool

`signed` , `unsigned` `_Bool` . `int` , ... (§6.7.2¶5) *`int`* *`signed int`* *`unsigned int`* .

.

Examples

.

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

23 4 . . FPGA . FPGA 32 .

```
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;
   };
};
```

(FPGA). .

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

C99 6.7.2.1, 10 :

( ) .

. . .

```
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;
```

```
#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;
}
```

. . 8 .

```
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. */
};
```

.

.

. .

'A' 1 .

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

B 2 . `c2` `c3` char `c2` `c3` 3 `c4` 3 2 .

```
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;
};
```

?

. .

```
 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
```

8 5 . 8 (0-255) . .

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

. . .

```
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;
}
```

0 . . . , .

```
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;
```

.

```
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;
}
```

, .

1. , .
2. (&) .
3. .
4. `sizeof()` .
5. `typedef typedef` (`typedef`).

```
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 */
}
```

: https://riptutorial.com/ko/c/topic/1930/-

25:

.

Examples

C

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;
}
```

foo.c main.c . gcc .

```
$ gcc -Wall -c foo.c
$ gcc -Wall -c main.c
```

:

```
$ gcc -o testprogram foo.o main.o
```

. . .

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` ? .

.

(" ") . .

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
```

(.h) .c .

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"
};
```

:

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;
}
```

GCC main . :

```
gcc -Wall -Wextra -pedantic -Wconversion -g  main.c resources.c -o main
```

(-Wall -Wextra -pedantic -Wconversion)

:

```
$ ./main
```

:

```
resource ID: 0, resource: '<unknown>'
resource ID: 1, resource: 'OK'
resource ID: 2, resource: 'Cancel'
resource ID: 3, resource: 'Abort'
```

.

```
int a; /* declaring single identifier of type int */
```

int a .

```
int a1, b1; /* declaring 2 identifiers of type int */
```

a1 b1 2 . int .

. (,) ( ). .). (*), (( )) (- [ ]) ( ). . :

```
/* 1 */ int /* 2 */ (*z) /* 3 */ , /* 4 */ *x , /* 5 */ **c /* 6 */ ;
```

#	
1	.
2	z .

#	
	.
4	x .
5	(*c) .
6	.

.

. ( )

(=) . () = . = .

:

```
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];
```

. :

```
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 b3 .

, b3 (*) ( ). ( ). .

```
int a3(); /* you should be able to call a3 */
```

a3 . a3 . . :

```
void f1()
{
    {
        int f2(); /* 1 refers to some function f2 */
    }

    {
        int f2(); /* refers to the exact same function f2 as (1) */
    }
}
```

2 f2 , ( ) .

```
int (*a3)(); /* you should be able to apply indirection to `a3` and then call it */
```

. * ( ) .

, ([ ]) / 1 .

```
int a4[5]; /* here a4 shouldn't be accessed using the index 5 later on */
```

5 . :

```
a4[0], a4[1]; a4[4];
```

a4[5] UB. .

```
int (*a5)[5](); /* here a4 could be applied indirection
                indexed up to (but not including) 5
                and called */
```

a5 .

Typedef

`typedef` `typedef` . :

```
typedef int (*(*t0)())[5];
```

(*`int typedef (*(*t0)())[5];`*)

typedef . .

```
t0 pf;
```

:

```
int (*(*pf)())[5];
```

typedef "". . `typedef` .

```
t0 (*pf1);
```

:

```
int (*(**pf1)())[5];
```

C

" - " C . .

.

```
*   as "pointer to"           - always on the left side
[]  as "array of"             - always on the right side
()  as "function returning"   - always on the right side
```

1

. . "." . .

2

. , , `()` . " " . `[]` *"identifier is array of"* . `)` . (`(` `()` . .)

3

. ("int"), . . `(` .

2 3 .

:

```
int *p[];
```

.

```
int *p[];
     ^
```

"p"

.

```
int *p[];
      ^^
```

"p "

.

```
int *p[];
    ^
```

"p" .

:

```
int *p[];
^^^
```

"p int ".

(*"p int "*)

:

```
int *(*func())();
```

.

```
int *(*func())();
       ^^^^
```

"func"

.

```
int *(*func())();
           ^^
```

"func "

.

```
int *(*func())();
      ^
```

"func "

.

```
int *(*func())();
              ^^
```

"func   ."

.

```
int *(*func())();
    ^
```

"func   ."

.   .

```
int *(*func())();
^^^
```

"func int   ."

,   .   .

. `[3]` *"array (size 3) of ..."* . `(char *,int)` * "function expecting (char , *int) and returning ..."* .

.

```
int (*(*fun_one)(char *,double))[9][20];
```

.

* "fun_one (char , *double) int (size 20) (size 9)   ."*

.

```
int (*(*fun_one)())[][];
```

.

:

C   . , :

```
int *((*fun_one)())[][];
```

"fun_one int " . .

:

```
[]() - cannot have an array of functions
()() - cannot have a function that returns a function
()[] - cannot have a function that returns an array
```

`()` `[]` `*` .

.

```
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

: https://riptutorial.com/ko/c/topic/3729/

26:

:   ?

( ) : https://www.amazon.com/Computer-Systems-Programmers-Perspective-2nd/dp/0136108040/

Examples

,   .   . .

```
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                         */
```

/ .   .   .

```
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) */
```

.

.   ,   .   .

:

```
extern int i = 0;  /* defines i */
extern int j;  /* declares j */
```

"   "   ( ).   (22).

```
/* 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 */
```

: https://riptutorial.com/ko/c/topic/3104/--

27:

Examples

if ()

`if` . .

C `if` .

```
if(cond)
{
  statement(s);  /*to be executed, on condition being true*/
}
```

,

```
if (a > 1) {
    puts("a is larger than 1");
}
```

`a > 1` `if` `true` . "a 1 " `a > 1` .

`if` `{ }` . .

```
if (a > 1)
    puts("a is larger than 1");
```

.

`if` . `if` .

```
if ((a > 1) && (b > 1)) {
    puts("a is larger than 1");
    a++;
}
```

`a` `b` `1` `printf` `a++` .

if () ... else

`if` `true` , `if` / `else` `true` `false` .

:

```
if (a > 1)
    puts("a is larger than 1");
else
```

```
    puts("a is not larger than 1");
```

if if else if ( ). if else if .

```
if (a > 1)
{
    puts("a is larger than 1");
    a--;
}
else
{
    puts("a is not larger than 1");
    a++;
}
```

switch ()

switch .

switch .

```
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;
}
```

```
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 a 1 a is 1 . a 2, a is 2 . a is neither 1 nor 2 a is neither 1 nor 2 .

case n: switch *n* . *n* *n* switch .

default: case n: . switch default .

break; switch .

: case break case , "fall through" case ( ) (*break* Duff 's Device . C .

break; 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 1 2 a is 1 or 2 , a is 1 or 2 a is 1, 2 or 3 . a 3 a is 1, 2 or 3 a is 1, 2 or 3 . a is neither 1, 2 nor 3 a is neither 1, 2 nor 3 .

default . switch .

enum switch .

```
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;
  }
}
```

.

- (default)
- enum (default)
- " default: default: " , enum " enum " . enum default ? ?
- enum .

.

```
enum msg_type t = (enum msg_type)666; // I'm evil
```

.

```
void f(enum msg_type t)
{
   if (!is_msg_type_valid(t)) {
      // Handle this unlikely error
   }

   switch(t) {
    // Same code than before
   }
}
```

if () ... else   if () ... else

`if ()... else` `if ()` () `if () ... else` `if () ... else` . "default" `else` .

:

```
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");
}
```

if () ... else VS if () .. else Ladder

`if()...else` () `if()...else` `if()...else` `if()...else` `if()` `if()..else` ladder `if()` `else if()` .

`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) && (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;
}
```

`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;
}
```

: https://riptutorial.com/ko/c/topic/3073/-

28: ,

(: `0`) (: `"C"`) C . , " , , , , .

( ) .

Examples

. .

		`5`
8	`0`	`0345`
16	`0x` `0X`	`0x12AB` , `0X12AB` , `0x12ab` , `0x12Ab`

. `-1` (`1`) . `-`

`int` `long` . C99 , `long long` .

8 16 `int` , `unsigned` , `long` `unsigned long` . C99 , `long long` `unsigned long long` .

.

`L` , `l`	`long int`
`LL` , `ll` (C99)	`long long int`
`U` , `u`	`unsigned`

U L / LL . (: `U`) .

. (: `"abcd"` `char*`).

`L` `wchar_t*` . : `L"abcd"` .

C11 `L` .

	`char`	
`L`	`wchar_t`	
`u8`	`char`	UTF-8
`u`	`char16_t`	UTF-16

U	char32_t	UTF-32

UTF .

. .

	double	3.1415926 -3E6
f , F	float	3.1415926f 2.1E-6F
l , L	long double	3.1415926L 1E126L

. 3f 3 3.f 3.0f . long double L .

. (: 'a' int . . .

L wchar_t . C11 u U char16_t char32_t char16_t .

(:) . . , \ . .

\b	
\f	
\n	( )
\r	
\t	
\v	
\\	
\'	
\"	
\?	
\nnn	8
\xnn ...	16

C89

\a	(,)

C99

\unnnn	

`\Unnnnnnnn`	

. . `n` 16 . UTF `char` .

I / O OS .

. , `??/` `'\'` `?\?/` `"??/"` .

8 1, 2 3 8 `n` .

, : https://riptutorial.com/ko/c/topic/3455/------

29: ()

- `#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 . `__STDC__NO_THREAD__` . (2016 7) C11 C .

C11 C .

-

C11 C :

- GNU libc

Examples

```
#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;
}
```

, . .

`once_flag` `call_once` .

```
#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 Big . call_once

- initBig .
- call_once initBig .

, mtx_t cnd_t mtx_init cnd_init .

() : https://riptutorial.com/ko/c/topic/4432/---

30:

. .

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

. .

`static`			. .
`extern`			. .
`auto`			.
`register`			. . & "" .
`typedef`			. .
`_Thread_local`		/	C11 . `extern` `static` .

( ) ( ) .

(`int` , `unsigned` , `short`) (`const` , `volatile`) .

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

(`void` , `char` , `int` , `signed long` , `unsigned long long` , `long double` ...) .

.

```
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;
```

.   .

(static) (static).   .   (data , bss rodata)   .

C11

C11 . C   .   . , gcc _Thread_local C   __thread .

.   static extern .   .   .

(   ) .   . ( )   .   .

.

( )   .   ( ).   .

Examples

typedef

.   .

```
/* 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);
```

typedef   .

typedef #define .

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

,

```
#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
```

. ,   .

`static` .

```
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` .

1. (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. (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;
    }
   ```

   .

   ```
    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
```

.

C99

3. null .

```
/* 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]);
}
```

(null) , . 512 . .

( ) . .

```
/* 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 `inline` .

```
/* 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);
```

. . `auto` .

register . register .

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

.

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

(, array &array[0]). , .

, register sizeof . . register . . register .

register . ,

```
/* 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 register .

_Thread_local

C11

C11 . C .

. _Thread_local . static extern .

```
#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);
    }
}
```

: https://riptutorial.com/ko/c/topic/3597/-

31:

ISO / IEC 9899 : 201x - C

, , .

A B A B .

C 2011 C .

1 5.1.2.3 .

- . (6.5.2.2).
- : AND `&&` (6.5.13); OR `||` (6.5.14); , (6.5.17).
- `? :` (6.5.15).
- : (6.7.6);
- . : (6.7.9) ; (6.8.3) ; (`if` `switch`) (6.8.4); `while` `do` (6.8.5); `for` (6.8.5.3) () ; `return` (6.8.6.4) () .
- (7.1.4).
- (7.21.6, 7.29.2) .
- , (7.22.5) .

Examples

.

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

, `a` `b` `c` . `b` `c` . `b` `c` .

, `a` `b` `c` (, `b` `c`). `a`

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

.

C11

.

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

, a , b , b . a

.

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

a b (unsequenced ,) , f .

.

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

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

.

-
- 1 .

---

1 .

f(a) (f a) . ., .

```
unsigned counter = 0;

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

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

printf counter . . :

0 1

1 0.

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

`counter` .

: https://riptutorial.com/ko/c/topic/1275/-

32:

Examples

( ) .   .

.   .

```
#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
```

```
#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;
}
```

.

```
int function(struct whatever *arg);

struct whatever
{
    int a;
    // ...
};

int function(struct whatever *arg)
{
    return arg->a;
}
```

GCC 6.3.0 (`dc11.c`) .

```
$ 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
$
```

`struct whatever;` `struct whatever;` . , .

```
#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;
}
```

. . *`label`* (`goto` *`label`*). .

```
#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 if , i . goto INSIDE; goto INSIDE; . .

.

```
#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 CLEANUP2 . goto . .

: https://riptutorial.com/ko/c/topic/1804/-

33:

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

	SIGABRT , SIGFPE , SIGILL , SIGTERM , SIGINT , SIGSEGV
	: SIG_DFL , , SIG_IGN SIG_IGN , void foo(int sig); .

C .

- SIGSEGV , SIGFPE , SIGILL C . C (: 0) .
- abort raise , raise .
- .
  - sig_atomic_t ( ) (C11 ,)
  - volatile .
- C . C abort , _Exit (C99) , quick_exit (C11) , signal ( ), (C11) .

C . .

- C . printf .
- C (C11) POSIX . .

Examples

"signal ()"

(: Cntrl-C) (SIGINT -) (SIGSEGV -) .

signal() ISO C

```
#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() . .

POSIX signal() sigaction() sigaction() . POSIX SIGUSR1 SIGUSR2 ISO C .

: https://riptutorial.com/ko/c/topic/453/-

34:

- ("") : ()

"   " "".

Examples

(C11). 6.3.1.3.

> **6.3.1.3**
> _Bool .
>
> [ ] .
>
> . .

. C " " . , .

`int` 32 .

```
#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;
}
```

void*   .   .   .

```
#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;
}
```

: https://riptutorial.com/ko/c/topic/2529/-----

35:

.

- a == b a a b .
- C . , .

, , a b . C .

C ( ) . .

, .

, .

const , a double* b const double* , *a *b .

. a b . . , *b *a .

. C .

void void , .

, const .

. , . .

Examples

▪

, char , unsigned char signed char . , char int , int b .

```
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;
}
```

" " . int .

.

C malloc .

union .

```
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;
}
```

union . int "view" ai .

( ).

```
// 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);
```

, `uint32_t*` . `uint32_t` .

▪

`float` `uint32_t` .

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

`u` `f` . `*f` `a` `b` .

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

, `*f` .

""

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

.

44.

. ,

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

. . .

2 `*e` `*f` .

```
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);
```

.

4 4 ?

. .

4 22 ?

e f . restrict .

```
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);
}
```

e f .

, char , signed char 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);
}
```

.

707406378 .

.

- unsigned char .
- a through *ap ap . a for . a .
- a , uint32_t . (707406378) .

: https://riptutorial.com/ko/c/topic/1301/---

36:

. *assertion* . .

- ()
- static_assert (,)
- _Static_assert (,)

	.
	.

`assert` `static_assert` `assert.h` .

`assert` `NDEBUG` . `NDEBUG` , `assert` no-op :

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

`NDEBUG` .

- `assert` calls `abort` assertion . `if/else` `exit` `quick_exit` . `abort` , (`atexit` `at_quick_exit`) .
- `assert` `assert` , , . . .
- (`abort`) (' - ') .

`static_assert` `_Static_assert` . `condition` . .

Examples

. . 0 0 .

```
#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;
}
```

. . . ., .(.) . .

```
#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` .

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

`NDEBUG` :

```
x = -1
```

`NDEBUG` . `NDEBUG` (: `config.h`) .

C11

true . , _ .

. , . .

assert `_Static_assert` . `static_assert` `<assert.h>` .

```
#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 . C99 false . `_Static_assert` . .

```
#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 .

`assert(0)` `assert(0)` .

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

    default:
        assert(0);
}
```

`assert()` false . `assert()` . `NDEBUG` .

`exit` , `quick_exit` `abort` . `exit` `quick_exit` . `abort()` (`assert`) , .

`assert()` . `abort()` . .

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

`assert()` . `abort()` . `assert()` `abort()` . . .

`abort` `exit` `quick_exit` `assert` .

. .

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

: p! = NULL, main.c, 5

AND (`&&`)

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

.

: p! = NULL && " f : p NULL " main.c, 5

0 (true) . `&& 1` . `&& "error message"` .

: https://riptutorial.com/ko/c/topic/555/

37:

C . C (: GLib) . .

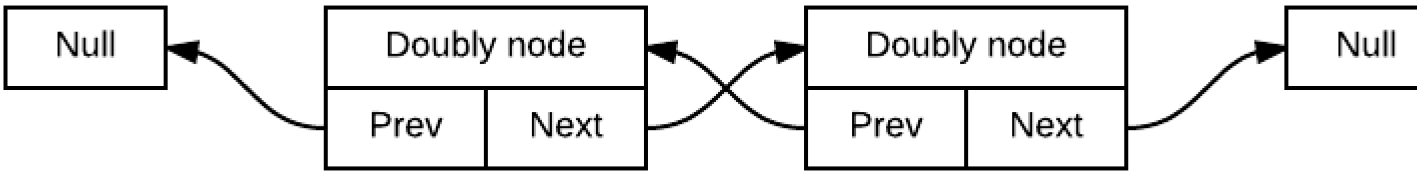
.

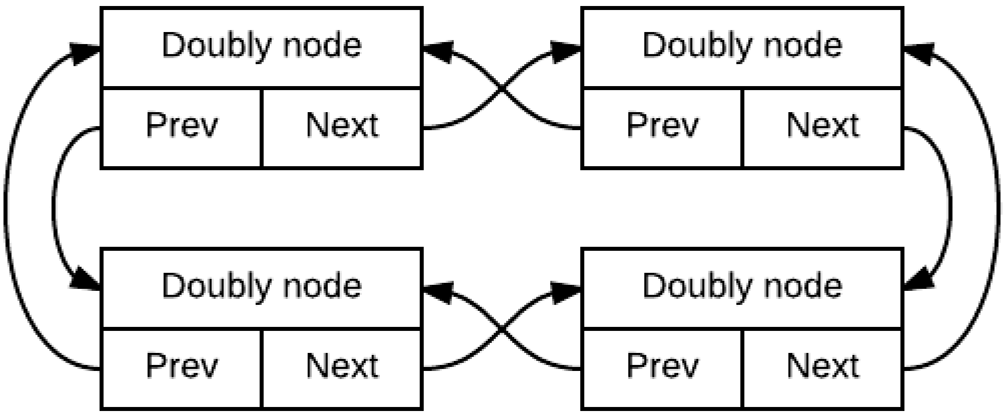
```
struct singly_node
{
  struct singly_node * next;
};
```

. .

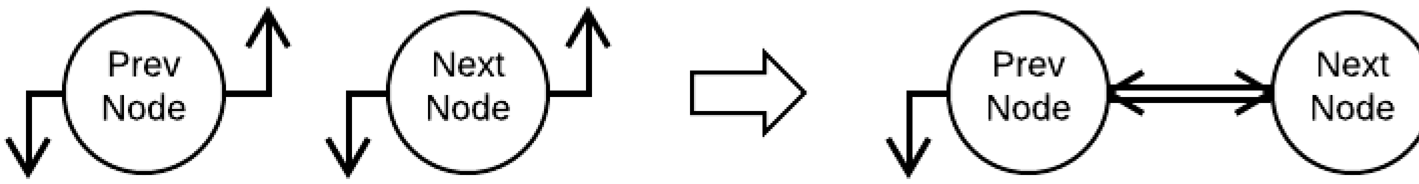
```
struct doubly_node
{
  struct doubly_node * prev;
  struct doubly_node * next;
};
```

Topoliges

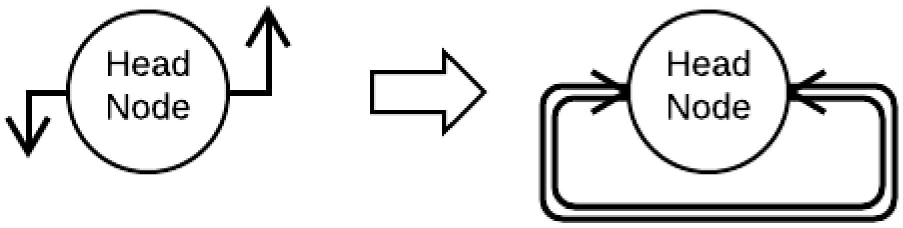




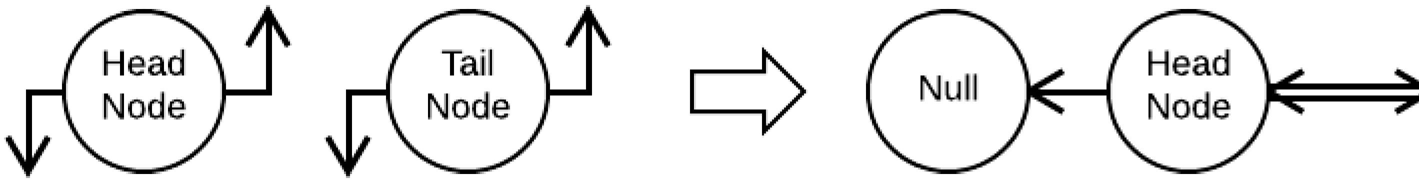
.



```
void doubly_node_bind (struct doubly_node * prev, struct doubly_node * next)
{
  prev->next = next;
  next->prev = prev;
}
```



```
void doubly_node_make_empty_circularly_list (struct doubly_node * head)
{
  doubly_node_bind (head, head);
}
```



```
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);
}
```

NULL . NULL .

```
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

.

```
/* 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;
}
```

.

1. . `HEAD` .

```
| HEAD | --> NULL
```

line `currentNode->next = *headNode;` `currentNode->next` `NULL` `headNode` `NULL` .

.

```
  -----      -------------
 |HEAD | --> |CURRENTNODE| --> NULL /* The head node points to the current node */
  -----      -------------
```

`*headNode = currentNode;`

2. . . 1 .

```
-----    -----------
HEAD --> FIRST NODE --> NULL
-----    -----------
```

`currentNode->next = *headNode` .

```
---------        -----    ---------------------
currentNode -->  HEAD --> POINTER TO FIRST NODE --> NULL
---------        -----    ---------------------
```

`*headNode` `currentNode` .

```
----    -----------    ---------------
HEAD -> currentNode -->     NODE       -> NULL
----    -----------    ---------------
```

`*headNode = currentNode;`

n

.   .   insert()   .

```
#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;
  }
}
```

.   . .

```
#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;
}
```

`previousNode` `NULL` . `NULL` `NULL` . `NULL` `NULL` .

. .

```
Head -> 1 -> 2 -> 3 -> 4 -> 5
```

. .

```
1 <- 2 <- 3 <- 4 <- 5 <- Head
```

.

1 `NULL` . 2 1 , 3 2 .

. .

(`nextNode`) .

, .

```
#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;
  }
}
```

(   ).

```
struct Node *lastNode = NULL;
```

.

.   Node .

```
struct Node {
  int data;
  int key;
  struct Node* next;
  struct Node* previous;
};
```

.

: https://riptutorial.com/ko/c/topic/560/-

38:

, .

C . C 2 . `a / b /` (`a , b`) 2 . (: `~` , `++`) `? :` .

- expr1
- expr2
- expr1 expr2
- expr1? expr2 : expr3

.

- *Arity* . C .
  - (1)
  - 2 (2)
  - 3 (3)
- "" . , . , C .

  ```
  a * b + c
  ```

  .

  ```
  (a * b) + c
  ```

  , .

  ```
  a * (b + c)
  ```

  .

  C . .

`() [] -> .`	
`! ~ ++ -- + - *` ( ) `(type) sizeof`	
`*` () `/ %`	
`+ -`	
`<< >>`	
`< <= > >=`	

`== !=`	
`&`	
`^`	
`|`	
`&&`	
`||`	
`?:`	
`= += -= *= /= %= &= ^= |= <<= >>=`	
`,`	

- *Associativity* . (`-`).

```
a - b - c - d
```

3 .

```
((a - b) - c) - d
```

`-` .

`*` `++` .

```
* ptr ++
```

,

```
* (ptr ++)
```

, (`++`) .

Examples

true . 1 (*true*) 0 (*false*) . (`if` , `while` , `for`) .

"=="

.

```
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.
*/
```

: (=) !

"! ="

.

```
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.
*/
```

equals (==) .

"!"

0 .

! .

```
!someVal
```

.

```
someVal == 0
```

> "

.

```
5 > 4      /* evaluates to 1. */
4 > 5      /* evaluates to 0. */
4 > 4      /* evaluates to 0. */
```

"<"

.

```
5 < 4      /* evaluates to 0. */
4 < 5      /* evaluates to 1. */
4 < 4      /* evaluates to 0. */
```

"> ="

.

```
5 >= 4      /* evaluates to 1. */
4 >= 5      /* evaluates to 0. */
4 >= 4      /* evaluates to 1. */
```

"<="

.

```
5 <= 4      /* evaluates to 0. */
4 <= 5      /* evaluates to 1. */
4 <= 4      /* evaluates to 1. */
```

.

```
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'. */
```

.

```
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 */
```

(`a`) . : `p`

```
*p += 27;
```

dereferences `p` .

```
*p = *p + 27;
```

`a = b` *rvalue* . . .

rvalue `if` (`switch`) . :

```
char *buffer;
if ((buffer = malloc(1024)) != NULL)
{
    /* do something with buffer */
    free(buffer);
}
else
{
    /* report allocation failure */
}
```

.

```
int a = 2;
/* ... */
if (a = 1)
    /* Delete all files on my hard drive */
```

, `a = 1` `1` `if` ( ). (`==`) .

```
int a = 2;
/* ... */
if (a == 1)
    /* Delete all files on my hard drive */
```

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

`c` `b` `b` , . `a` . .

. (, , ,).

(`+`) . :

```
#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;
}
```

(`-`) . :

```
#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;
}
```

(*) .:

```
#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;
}
```

* .

(/) . (%).

.

:

```
#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;
}
```

(%) .:

```
#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;
}
```

/

(a++) (a--) . . / .:

```
#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 d c . (++) (--) . .

/ .

AND

AND-ing . 0 1 . AND `int` .

```
0 && 0  /* Returns 0. */
0 && 1  /* Returns 0. */
2 && 0  /* Returns 0. */
2 && 3  /* Returns 1. */
```

OR

0 1 ORing . OR `int` .

```
0 || 0  /* Returns 0. */
0 || 1  /* Returns 1.  */
2 || 0  /* Returns 1.  */
2 || 3  /* Returns 1.  */
```

NOT

. NOT `int` . NOT 1 , 0 . 1 .

```
!1 /* Returns 0. */
!5 /* Returns 0. */
!0 /* Returns 1. */
```

`&&` `||` .:

- (RHS) (LHS) ,
- ,
- .

.

- LHS "(0) , `||` ('true OR anything' 'true') ,
- LHS "() `&&` RHS (' AND ' ").

.

```
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";
}
```

`value >= 0` `value < NUM_NAMES` `value < NUM_NAMES` `value < NUM_NAMES` .

/

.

```
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 */
```

`++` `--` .

/

0 . .

```
a = b ? c : d;
```

.

```
if (b)
    a = c;
else
    a = d;
```

: `condition ? value_if_true : value_if_false` . .

```
int x = 5;
int y = 42;
printf("%i, %i\n", 1 ? x : y, 0 ? x : y); /* Outputs "5, 42" */
```

. .

```
big= a > b ? (a > c ? a : c)
           : (b > c ? b : c);
```

.

```
#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;
}
```

. :

```
exp1 ? exp2 : exp3 ? exp4 : exp5
```

.

```
exp1 ? exp2 : ( exp3 ? exp4 : exp5 )
```

.

```
int x = 42, y = 42;
printf("%i\n", (x *= 2, y)); /* Outputs "42". */
```

.

, . .

printf() .

```
printf("%i\n", (x *= 2, y)); /* Outputs "42". */
/*            ^        ^ this is a comma operator */
/*            this is a separator */
```

for . :

```
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);
```

.

```
int x = 3;
int y = 4;
printf("%f\n", (double)x / y); /* Outputs "0.750000". */
```

x double , division y double double printf .

sizeof

size_t . .

```
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! */
```

size_t . . . .

```
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. */
```

N N .

```
int arr[] = {1, 2, 3, 4, 5};
printf("*(arr + 3) = %i\n", *(arr + 3)); /* Outputs "4", arr's fourth element. */
```

. 3 + arr . arr[k] k+1 arr+k arr[k] . , arr arr+0 arr[0] arr+1 arr[2] . *(arr+k) arr[k] .

, int 1 4 . + . . .

```
#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;
}
```

.

```
#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 + ++ . ptr - -- .

ptrdiff_t .

```
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". */
```

(. ->) (A) struct .

lvalue .

```
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". */
```

. x->y (*x).y - .

```
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". */
```

& . . lvalue . .

```
int x = 3;
int *p = &x;
printf("%p = %p\n", (void *)&x, (void *)p); /* Outputs "A = A", for some implementation-
defined A. */
```

`*` . lvalue .

```
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". */
```

. `a[i]` `*(a + i)` .

```
int arr[] = { 1, 2, 3, 4, 5 };
printf("arr[2] = %i\n", arr[2]); /* Outputs "arr[2] = 3". */
```

(,). `pointer + integer == integer + pointer` .

`arr[3]` `3[arr]` .

```
printf("3[arr] = %i\n", 3[arr]); /* Outputs "3[arr] = 4". */
```

`arr[3]` `3[arr]` . .

( ). , . .

```
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 */
```

.
C 6 .

&	AND
|	OR
^	(XOR)
~ ~	(1)
<<	
>>	

.

```
#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;
}
```

. .

- 1 .
- (1) .
- .

:

( ) . ( ) .

.

- .
- , 0 (AND)
- , 1 (OR).
- , (OR).

.

```
#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

. 2 . C `size_t` .

. .

`<stdalign.h>` `alignof` .

```
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;}));
}
```

:

```
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

(if / while / ...) . ( ) (: && false ||) . .

:

```
#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;
}
```

.

```
#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!
```

() . , . "forked!" 4 ?

: https://riptutorial.com/ko/c/topic/256/

39:

`enum` .

`int` .

enumerator-list .

`int` "" .

`char` , . . , .

"" `0` 1 .

"" .

Examples

. `enum` .

`int` `string/ char*` `enum` .

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);
}
```

(:) :

```
int main(){
    enum color chosenColor;
    printf("Enter a number between 0 and 2");
    scanf("%d", (int*)&chosenColor);
    printColor(chosenColor);
    return 0;
}
```

C99

2

C99 .

```
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]);
}
```

:

```
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` .

```
enum color
{
    RED,
    GREEN,
    BLUE
};
```

.

```
enum color chosenColor = RED;
```

enum typedef enum type .

```
typedef enum
{
    RED,
    GREEN,
    BLUE
} color;

color chosenColor = RED;
```

enum color . enum . C ++ .

```
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

.

```
#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;
}
```

:

```
Base = 0
One = 1
Two = 2
Negative = -1
AnotherZero = 0
```

typename

.

```
  enum { buffersize = 256, };
  static unsigned char buffer [buffersize] = { 0 };
```

`int` .

: https://riptutorial.com/ko/c/topic/5460/-

40:

- #include <errno.h>
- int errno; / * * /
- #include <string.h>
- char * strerror (int errnum);
- #include <stdio.h>
- void perror (const char * s);

`errno . errno .`

Examples

errno . C errno 3 :

EDOM	
EILSEQ	

strerror

`perror <string.h> strerror .`

```
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;
}
```

stderr <stdio.h> perror .

```
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;
}
```

errno .

: https://riptutorial.com/ko/c/topic/2486/-

41:

- `#ifdef __STDC_NO_ATOMICS__`
- `# error this implementation needs atomics`
- `#endif`
- `#include <stdatomic.h>`
- _Atomic = ATOMIC_VAR_INIT (0);

C Atomics C11 .

. . (`++`) , , . .

- : `_Atomic` .
- : - - (: `++` `*=`) .
- : `atomic_compare_exchange` .
- : .
- : . .
- lock-free `atomic_flag` . .

C11 . mutex `mtx_t` . .

Examples

.

```
/* 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;
}
```

lvalue ( ) , .

- . . .
- `a = a+1;` `a` : , . . `a += 1;` `a++;` .

: https://riptutorial.com/ko/c/topic/4924/-

42:

. . ( ) (,) .

const	( ) .
volatile	() ( ) .
restrict	. .

(static , extern , auto , register), (signed , unsigned , short , long) (int , char , double) .

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

```
/* "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;
```

```
/* "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

(const)

```
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 . .

```
_Bool doIt(double const* a) {
   double rememberA = *a;
   // do something long and complicated that calls other functions

   return rememberA == *a;
}
```

*a false true .

const .

```
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" */
```

. .

volatile .

volatile .

```
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;
```

.

- I / O .
- (:) ,

.

```
int quit = false;

void main()
{
    ...
    while (!quit) {
      // Do something that does not modify the quit variable
    }
    ...
}
```

```
void interrupt_handler(void)
{
  quit = true;
}
```

while quit while (true) . quit SIGINT SIGTERM .

volatile quit .

.

```
uint8_t * pReg = (uint8_t *) 0x1717;

// Wait for register to become non-zero
while (*pReg == 0) { } // Do something else
```

. . . .

```
uint8_t volatile * pReg = (uint8_t volatile *) 0x1717;
```

: https://riptutorial.com/ko/c/topic/2588/-

43:

Examples

. .

(`main()`). (`plusfive()` `timestwo()`) "source1.c" "source2.c" . `main()` .

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
```

`timestwo` `plusfive` `complicated1` `complicated2` " ", . .

gcc .

```
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
```

-O2 .

`inline` . . `inline` `.o` " ".

. . `.c` `extern` . `source1.c` .

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

`static inline` `inline` . .

: https://riptutorial.com/ko/c/topic/7427/

44:

C . ISO C . . .

1. . . C .
2. .

1. . (`#ifdef`) . .
2. x86 ARM . . .
3. . .

Examples

gcc asm

gcc .

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

`AssemblerInstructions` . volatile asm gcc . `AssemblerInstructions` . asm C asm . GCC .

```
/* Note that this code will not compile with -masm=intel */
#define DebugBreak() asm("int $3")
```

`DebugBreak()` `int $3` . gcc asm asm . asm .

gcc asm

gcc asm .

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

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

`AssemblerTemplate` `OutputOperands` C `InputOperands` C `Clobbers` `GotoLabels` goto .

C . ARM 16 32 Linux .

```
/* 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 `x` . C .

asm gcc C asm . asm asm `volatile` .

gcc

.

```
#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 . AES . , AES , C `>>` `Rotate Right` .

'AES256' 'AddRoundKey ()' .

```
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
```

`subkey` `w` .

`Rotate Right` shift + assign assign C .

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

.

: https://riptutorial.com/ko/c/topic/4263/-

45:

- _Generic (, generic-assoc-list)

generic-assoc-list	,
	type-name : **OR** :

1. `_Generic` 1 .
2. `_Generic` 7 . .

Examples

.

```
#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
```

---

generic :

```
#define is_const_int(x) _Generic((x), \
        const int: "a const int",     \
        int:       "a non-const int", \
        default:   "of other type")
```

.

```
i is a non-const int
j is a non-const int
k is of other type
```

`_Generic` .

```
#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
```

`int` `double` . `print(X)` .

`_Generic` .

```
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))
```

`(X)+(Y)` .] .

.

int / string .

```
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` [1] . Add : `( x , y )` .

---

[1] (: ISO : IEC 9899 : 201X 6.5.1.1 3)
.

: https://riptutorial.com/ko/c/topic/571/-

46: C

Examples

.

```
if ( i  == 2) //Bad-way
{
    doSomething;
}
```

== =   .   .

```
if( 2 == i) //Good-way
{
    doSomething;
}
```

" "   .   .

.

. void .

.

- .   .
- ***void***   ( ) ,   . , .

?

ANSWER : ***void***

:   . , `#define` `enum` (`#define`).

C11 6.7.6.3 " ", 10,

> void   .

14   :

> ...   .   .

K & R (pgs- 72-73)   :

> `double atof();` `atof`   .   .   C   .   . . `void` .

:

```
int foo(void);
```

.

```
int foo(void)
{
    ...
    <statements>
    ...
    return 1;
}
```

`int foo()` (, ***void***) `foo(42)` . . .

`main()` :

```
int main(void)
{
    ...
    <statements>
    ...
    return 0;
}
```

. :

```
#include <stdio.h>

static void parameterless()
{
    printf("%s called\n", __func__);
}

int main(void)
{
    parameterless(3, "arguments", "provided");
    return 0;
}
```

`proto79.c` GCC (macOS Sierra 10.12.5 7.1.0) :

```
$ gcc -O3 -g -std=c11 -Wall -Wextra -Werror -Wmissing-prototypes -pedantic proto79.c -o
proto79
$
```

.

```
$ 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
$
```

`static void parameterless(void)` `static void parameterless(void)` .

```
$ 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)
             ^~~~~~~~~~~~~
$
```

- .

C : https://riptutorial.com/ko/c/topic/10543/-c-----

47:

C . .

Examples

`signed` `unsigned` . , ?

```
#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 -1 `a is more than b` .

(, 6.3.1.8).

" " `unsigned int` . ,

.

`int` `b` `unsigned int` .

-1 `unsigned int` `unsigned int` `unsigned int` 1000 `a > b` false.

== =

`=` .

`==` .

.

```
/* assign y to x */
if (x = y) {
     /* logic */
}
```

:

```
/* compare if x is equal to y */
if (x == y) {
    /* logic */
}
```

y x 0 .

```
if ((x = y) != 0) {
    /* logic */
}
```

. .

```
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);
}
```

. :

```
if (x = y)         /* warning */

if ((x = y))       /* no warning */
if ((x = y) != 0)  /* no warning; explicit */
```

(Yoda). rvalues .

```
if (5 = y) /* Error */

if (5 == y) /* No error */
```

C . Yoda .

.

```
if (x > a);
   a = x;
```

.

```
if (x > a) {}
a = x;
```

x a . .

.

```
if (i < 0)
    return
day = date[0];
hour = date[1];
minute = date[2];
```

day = date [0] .

. :

```
if (x > a) {
    a = x;
}
```

\ 0

malloc strlen 1 .

```
char *dest = malloc(strlen(src)); /* WRONG */
char *dest = malloc(strlen(src) + 1); /* RIGHT */

strcpy(dest, src);
```

strlen \0 . WRONG strcpy .

stdin .

```
#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 */
```

( )

strdup() API . ( ) . . . . . .

getline() .

```
#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;
}
```

getline() .

```
#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;
}
```

. " " . , . , . , exit() , main() . .

. getline() . . . C API *( )* .

NULL . (: NULL / NULL). ( ) ( ) / . 0 (NULL) C NOP (:). NULL free() . .

```
char buf[8]; /* tiny buffer, easy to overflow */

printf("What is your name?\n");
scanf("%s", buf); /* WRONG */
scanf("%7s", buf); /* RIGHT */
```

7 (null -1) buf . . .

realloc

realloc NULL NULL . realloc NULL ( ) . NULL , .

```
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");
```

(float , double long double) . 1/3 10 1/3 1/10 2 . . .

```
#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;
}
```

:

```
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)
```

.

```
#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;
}
```

`ptr2` `ptr2` . 32 `sizeof(int)` 4 " `array[0]` " 8 .

`ptr2` `array[0]` 2 `ptr2` , 2 .

```
#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;
}
```

.

```
#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]` `(*((E1)+(E2)))` (N1570 6.5.2.1 2) `&(E1[E2])` `((E1)+(E2))` (N1570 6.5.3.2, 102).

. ' ' .

```
#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;
}
```

▪

. .

```
#include <stdio.h>

#define SQUARE(x) x*x

int main(void) {
    printf("%d\n", SQUARE(1+2));
    return 0;
}
```

`9` (`3*3`) `1+2*1+2` `5` .

.

```
#include <stdio.h>

#define SQUARE(x) ((x)*(x))

int main(void) {
    printf("%d\n", SQUARE(1+2));
    return 0;
}
```

.   .

```
#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;
}
```

`((a++) <= (10) ? (a++) : (10))` . `a++` (`0`) `10`   `a++`   `a`   `MIN`   .

.

```
#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;
}
```

double-evaluation , `min`   `double`   .

.

```
#define OBJECT_LIKE_MACRO     followed by a "replacement list" of preprocessor tokens
#define FUNCTION_LIKE_MACRO(with, arguments) followed by a replacement list
```

`#define`   . *lparen*   .   .   `(`   ).

C99

C99 `static inline int min(int x, int y) { … }` .

C11

C11 `min` '' .

```
#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;
}
```

`double` , `float` , `long long` , `unsigned long` , `long` , `unsigned` `gen_min` .

. .

```
$ 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
$
```

.

```
int foo(void);

int main(int argc, char **argv)
{
    int foo_val;
    foo_val = foo();
    return foo_val;
}
```

foo (`int foo();`) ( ). , `Undefined reference` .
foo .

```
/* 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;
}
```

. foo() foo.c   foo.c undefined_reference.c   foo()   foo.h   . foo.c undefined_reference.c .

```
$ gcc -c undefined_reference.c
$ gcc -c foo.c
$ gcc -o working_program undefined_reference.o foo.o
$
```

:

```
$ gcc -o working_program undefined_reference.c foo.c
$
```

.

```
#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;
}
```

. #include <math.h> pow()   .

```
$ 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
```

```
$
```

pow() . -lm libm . (-lm macOS , .)

. .

```
$ 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
$
```

!

, Type** Type[M][N] .

```
#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;
}
```

:

```
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)
```

main s print_strings , . print_strings print_strings main .

(*array decay*) . char[4][20] (20 4) s &s[0] . char (*)[20] (20 1 char (*)[20] , (3) . . .

char [20]	**(20)**	char *	**(1)**
char [4][20]	**(** 20 **4**)	char (*)[20]	**(** 20 **1**)
char *[4]	**(** 1 **4**)	char **	**(** 1 **1**)

`char [3][4][20]`	**array of (** 20 4 **3**)	`char (*)[4][20]`	**(** 20 4 **1**)
`char (*[4])[20]`	**(** 20 1 **4**)	`char (**)[20]`	**(** 20 1 **1**)

. .

. , . .

, (`char (*)[20]`) (`char **`) . `print_strings` .

```
void print_strings(char (*strings)[20], size_t n)
/* OR */
void print_strings(char strings[][20], size_t n)
```

`print_strings` char : 20 30 ? 50? .

```
#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;
}
```

.

```
Example 1
Example 2
Example 3
Example 4
```

""

`malloc` , `calloc` `realloc` ( ).

```
/* 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]));
```

"" (: `int array[4][16];`) 0 `int array[4][16];` :

```
/* 0x40003c, 0x402000 */
printf("%p, %p\n", (void *)(array[0] + 15), (void *)array[1]);
```

`int` , 8128 (8132-4) , 2032 `int` . , "" .

"" `int *` .

```
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` 2 .

```
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` `array` (VLA) . `int *` `func` `func_vla` `func_vla` `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 C11 . C11 `__STDC_NO_VLA__` 1 C99 .

,

C .

`'a'` . . `'abc'` .

`"abc"` 0 . , `char` . null `"abc"` 4 (`{'a', 'b', 'c', '\0'}`).

. . 2 g g . .

```
#include <stdio.h>

int main(void) {
    const char *hello = 'hello, world'; /* bad */
    puts(hello);
    return 0;
}
```

. `char` . (, `char` .) .

```
#include <stdio.h>

int main(void) {
    char c = "a"; /* bad */
    printf("%c\n", c);
    return 0;
}
```

. .

C . , `malloc` , null . .

:

```
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 */
```

:

```
#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;
}
```

.   .

scanf ()   .

```
#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;
}
```

.

```
42
life
```

`42` `"life"` `42` `""` .

`42` `scanf()` `fgets()` `life` . , `fgets()` `life` .

( ) `scanf()` `fgets()` . `sscanf()` .

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

int main(void) {
    int num = 0;
    char line_buffer[128] = "", str[128], *lf;
```

```c
    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;
}
```

`scanf()` `fgets()` .

```c
#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

C C . ,

```c
/* WRONG */
#define MAX 100;
int arr[MAX];
```

.

```c
int arr[100;];
```

. `#define` . `#define` .

.

C / * * / .

:

```c
/*
 * 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;
}
```

.

```
//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...
*/
```

C99 .

```
// 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;
}
```

.

```
/*

// 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;
}

*/
```

`#ifdef` `#ifndef` . .

```
#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
```

`#if 0` .

#if 0 .

0 . 0 1 . .

```
#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`

.

```
#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`

.   .

-

.

:

```
#include <stdio.h>

int factorial(int n)
{
       return n * factorial(n - 1);
}

int main()
{
    printf("Factorial %d = %d\n", 3, factorial(3));
    return 0;
}
```

: `Segmentation fault: 11`

.   .

:

```
#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;
}
```

```
Factorial 3 = 6
```

`n` 1 (`n` `int` 32 `12` `12`).

:

1. . . .
2. . .
3. .
4. .
5. .
6. .

:

'true'

C `bool` , `true` `false` . `true` 1 `false` 0 .

C99

C99 `_Bool` `_Bool` `bool` (`_Bool`), `false` `true` `<stdbool.h>` . `bool` , `true` `false` .

0 false 0 . :

```
/* 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;
    }
}
```

`true` `true` . `true` `if` `(bitfield & 0x80)` `true` . `1` `0x80` . .

```
/* 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;
    }
}
```

0 true .

```
/* 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;
    }
}
```

double .

`float` . `0.1` `double` . :

```
#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
```

`n` 0.100000001490116112. `n` `0.1` (0.1000000000000000001) .

`float` `double` .

: https://riptutorial.com/ko/c/topic/2006/-

48:

() # . AC #define . C (C) .

. C . , '}' .

.

( ) . POSIX -E . gcc stdout :

```
$ gcc -E cprog.c
```

cpp . . . gcc , -P .

```
$ cpp -P cprog.c
```

Examples

(: #if , #ifdef , #else , #endif) .

```
/* 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 C .

```
#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 C if . defined(identifier) 1 0 .

```
#if defined(DEBUG) && !defined(QUIET)
#define DLOG(x) (printf(x))
#else
#define DLOG(x)
#endif
```

. .

`SHORT SerOpPluAllRead(PLUIF *pPif, USHORT usLockHnd)` .   .

.   . `SerOpPluAllRead()`   `SerOpPluAllRead_Debug()`   ,   .

.

```
#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
```

`SerOpPluAllRead()`   .

▪   .   .

.   `SerOpPluAllRead()` `SerOpPluAllRead()` `SerOpPluAllRead_Debug()`   ,   .

```
#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;
}
```

#include .

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

#include . (<>) ("") .

.

```
#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
```

manifest constant manifest constant .

```
#define ARRSIZE 100
int array[ARRSIZE];
```

10 .

```
#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);
```

. TIMES10 A b . TIMES10 TIMES10 .

```
#define TIMES10(A) ((A) = (A) * 10)
```

A .

. . ( ) .

```
#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 */
```

. C11 `_Generic` .

( ) .

`#error` .

```
#define DEBUG

#ifdef DEBUG
#error "Debug Builds Not Supported"
#endif

int main(void) {
    return 0;
}
```

:

```
$ gcc error.c
error.c: error: #error "Debug Builds Not Supported"
```

#if 0

.

```
/* 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;
}
*/
```

* / .

```
/* 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;
}
*/
```

'* /' . `#if 0` .

```
#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
```

"#if 0" .

`#if 0` . .

`#if 0` # `#defined` . `#if defined(POSSIBLE_DEAD_CODE)` `#if defined(FUTURE_CODE_REL_020201)` . .

. , `front##back` `frontback` . Win32 `<TCHAR.H>` . C `L"string"` . Windows API `#define UNICODE` . `TCHAR.H` .

```
#ifdef UNICODE
#define TEXT(x) L##x
#endif
```

`TEXT("hello, world")` UNICODE C `L` . `L` `"hello, world"` `L"hello, world"` .

C .

- ( ) `__FILE__`
- ( ) `__LINE__` ,
- `__DATE__` ( ),
- `__TIME__` ( ).

`__func__` (ISO / IEC 9899 : 2011 §6.4.2.2) .

> `__func__` .
>
> ```
> static const char __func__[] = "function-name";
> ```
>
> . *function-name* .

`__FILE__` , `__LINE__` `__func__` . :

```
fprintf(stderr, "%s: %s: %d: Denominator is 0", __FILE__, __func__, __LINE__);
```

C99 `__func__` . gcc C89 `__FUNCTION__` .

:

- `__STDC_VERSION__` C . `yyyymmL` (`201112L` C11 `199901L` C99 , C89 / C90)
- `__STDC_HOSTED__` `__STDC_HOSTED__` `1` , `0` .

- `__STDC__` 1 C .

( )

ISO / IEC 9899 : 2011 §6.10.9.2 :

- `__STDC_ISO_10646__` `yyyymmL` (: `yyyymmL`). `wchar_t` `wchar_t` . Unicode ISO / IEC 10646 . .
- `__STDC_MB_MIGHT_NEQ_WC__` `wchar_t` 1.
- `__STDC_UTF_16__` `char16_t` UTF-16 . .
- `__STDC_UTF_32__` `char32_t` UTF-32 . .

ISO / IEC 9899 : 2011 §6.10.8.3

- `__STDC_ANALYZABLE__` L ( ) 1.
- `__STDC_IEC_559__` F (IEC 60559) 1.
- `__STDC_IEC_559_COMPLEX__` G (IEC 60559) 1.
- `__STDC_LIB_EXT1__` K ( ) `201112L` .
- `__STDC_NO_ATOMICS__` (`_Atomic`) `<stdatomic.h>` 1.
- `__STDC_NO_COMPLEX__` `<complex.h>` 1.
- `__STDC_NO_THREADS__` 1. `<threads.h>` .
- `__STDC_NO_VLA__` 1.

.

my-header-file.h

```
#ifndef MY_HEADER_FILE_H
#define MY_HEADER_FILE_H

// Code body for header file

#endif
```

`#include "my-header-file.h"` `#include "my-header-file.h"` , . .

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
}
```

. `MyStruct` `MyStruct` . . :

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
}
```

.

```
#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
}
```

header-1.h `HEADER_1_H` . Ergo, :

```
#define HEADER_1_H

typedef struct {
    …
} MyStruct;

int myFunction(MyStruct *value);

#define HEADER_2_H

int myFunction2(MyStruct *value);

int main() {
    // do something
}
```

.

: . `HEADER_2_H_` , `MY_PROJECT_HEADER_2_H` . .

. . , C `FILE` (I / O). , `header-1.h` .

```
#ifndef HEADER_1_H
#define HEADER_1_H

typedef struct MyStruct MyStruct;
```

```
int myFunction(MyStruct *value);

#endif
```

(`MyStruct` - typedef `MyStruct`) `{ … }` . " `struct MyStruct` `MyStruct` ".

.

```
struct MyStruct {
    …
};
```

C11 `typedef struct MyStruct MyStruct;` `typedef struct MyStruct MyStruct;` C . , C11 .

`#pragma once` .

my-header-file.h

```
#pragma once

// Code for header file
```

`#pragma once` C .

. `<assert.h>` . `NDEBUG` . . . .

FOREACH

. , single-linked doubly-linked, C `foreach` .

.

```
#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);
    }
}
```

FOREACH   .

```
#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);
    }
}
```

.

```
void* (*first)(void *coll);
void* (*last) (void *coll);
void* (*next) (void *coll, CollectionItem *currItem);
```

C ++   C ++ C   __cplusplus - name mangling

C   C ++   .

C   C ++   . C ++   ( )   C ++   C   . C ++ C ++   . C ++   C   .

C   C ++ C ++   (   )   `__cplusplus`   .

C C ++ __cplusplus C ++ C . #ifdef defined() #if C ++ C .

```
#ifdef __cplusplus
printf("C++\n");
#else
printf("C\n");
#endif
```

.

```
#if defined(__cplusplus)
printf("C++\n");
#else
printf("C\n");
#endif
```

C ++ C C __cplusplus extern "C" { /* ... */ }; C ++ C . C extern "C" { */ ... */ }; . extern "C" { /* ... */ }; C ++ C .

```
#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
```

inline .

```
#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;
}
```

(DEBUG) void . if/else .

DEBUG do { ... } while(0) . , (void)0 .

```
#define LOG(LINE) do { /* empty */ } while (0)
```

.

- GCC , GNU . :

```
#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);
}
```

Variadic

C99

:

print .

```
#define debug_print(msg) printf("%s:%d %s", __FILE__, __LINE__, msg)
```

:

somefunc() -1 0 .

```
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()` ? . .

```
debug_printf(retVal);      /* this would obviously fail */
debug_printf("%d",retVal); /* this would also fail */
```

`__VA_ARGS__` . X- .

:

```
 #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);
```

.

```
debug_print("Hey");
```

`debug_print()` . `debug_print("Hey",);` .

`##__VA_ARGS__` . `##__VA_ARGS__` .

:

```
 #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());
```

: https://riptutorial.com/ko/c/topic/447/--

49:

- ; / * . val . * /
- ; / * void . * /
- ; / * (Iteration Statement) () switch . * /
- ; / * Iteration Statement () . * /
- LBL . / * LBL . * /
- LBL : / * . * /

C .

C , `jmp_buf` C `setjmp` `longjmp` *long jump* .

/ : for, while, do-while

Examples

goto .

. `i` `j` , .

```
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 */
     }
}
```

C `goto` . .

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

`return` . " ".

`goto` .

```
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 . " ".

: 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;
}
```

:

1. void (void *) return . , return return; .
2. void return .
3. main() (main()), return (C99) . } , 0 . return . .

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. */
}
```

continue break .

```
#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);
  }
}
```

: https://riptutorial.com/ko/c/topic/5568/-

50:

C . . , .

UB , .

(UB) ?

C . C11 (ISO / IEC 9899 : 2011) .

UB ?

.

( )).

( ) .

" [ ] ."(ANSI C)

UB ?

?

. .

UB ?

.

- . .
- . . , .

: null . , . , . .

null C null Java C # . throw (Java `NullPointerException` , C # `NullReferenceException`) Java C # *C* .

.

- , C .
- . .

C . POSIX C . . C .

.

C (UB).

(: PC-Lint) .

Examples

NULL .

```
int * pointer = NULL;
int value = *pointer; /* Dereferencing happens here */
```

`NULL` C , .

```
int i = 42;
i = i++; /* Assignment changes variable, post-increment as well */
int a = i++ + i--;
```

`i` " " . C . C2011 .

. , .

(C99 , 6.5, 2)

, C99 . C2011 , " " :

. . . , . .

(C2011 , 6.5, 2)

" " . , v , v v v " v ".

.

```
int i = 42;
i = (i++, i+42); /* The comma-operator creates a sequence point */
```

.

```
int i = 42;
printf("%d %d\n", i++, i++); /* commas as separator of function arguments are not comma-
operators */
```

. .

return .

```
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;
}
```

. ( ) 1 .

. ,

```
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()` 0 (2) .

---

1 (*ISO / IEC 9899 : 201x* , 6.9.1 / 12)

} .

2 (*ISO / IEC 9899 : 201x* , 5.1.2.2.3 / 1)

main } 0 .

C99 C11 6.5 / 5 . . 6.2.5 / 9 . . . ,

```
#include <limits.h>      /* to get INT_MAX */

int main(void) {
    int i = INT_MAX + 1; /* Overflow happens here */
    return 0;
}
```

. ( ) , . , .

```
int square(int x) {
    return x * x;  /* overflows for some values of x */
}
```

, . . .

.

```
int zero(int x) {
    return x - x;  /* Cannot overflow */
}
```

. .

```
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 */
}
```

. `int` .

```
int a;
printf("%d", a);
```

`a` `int` . (`a`). . .

: `static` 0 . .

```
static int b;
printf("%d", b);
```

---

0 . . .

:

```
#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
```

. , .

```
int main(void)
{
    int *p;
    p++; // Trying to increment an uninitialized pointer.
}
```

.

```
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;
}
```

. , `gcc` .

```
warning: function returns address of local variable [-Wreturn-local-addr]
```

`clang` .

```
warning: address of stack memory associated with local variable 'baz' returned
[-Wreturn-stack-address]
```

.

(1) `static` .

(2) ISO / IEC 9899 : 2011 6.2.4 §2 " ."

(3) `foo` .

0

```
int x = 0;
int y = 5 / x;  /* integer division */
```

```
double x = 0.0;
double y = 5.0 / x;  /* floating point division */
```

```
int x = 0;
int y = 5 % x;  /* modulo operation */
```

(x) 0 .

(: IEEE 754) , C 0 (: `INFINITY`) .

`n` `memory` `memory + (n - 1)` . . , .

```
int array[3];
int *beyond_array = array + 3;
*beyond_array = 0; /* Accesses memory that has not been allocated. */
```

3 . , .

```
int array[3];
array[3] = 0;
```

(`beyond_array = array + 3`) (`*beyond_array`). (`malloc`) .

. .

, ...

```
#include <string.h> /* for memcpy() */

char str[19] = "This is an example";
memcpy(str + 7, str, 10);
```

... 3 10 . :

```
                overlapping area
                |
                _ _
               |   |
               v   v
T h i s   i s   a n   e x a m p l e \0
^              ^
|              |
|              destination
|
source
```

.

`memcpy()` , `strcpy()` , `strcat()` , `sprintf()` `sscanf()` . .

.

`memmove()` . . `memmove()` .

. . . . C , `memmove()` . .

C11

1 .

-
-
- .

a .

```
void Function( void )
{
    int a;
    int b = a;
}
```

---

[1] (: ISO : IEC 9899 : 201X 6.3.2.1 Lvalues, 2)
lvalue ( ) ( )), .

C11

C11 . [1] , . [2] . ( ) .

.

```
#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 );
}
```

`thrd_create` `Function` . `a` `a` . .

- `a` .
- `thrd_join` `a` .
- `a` .

, `a` . ( ).

---

1 .

2 (ISO : IEC 9889 : 201x, 5.1.2.4 . " ")
. .

(UB) .

```
char *p = malloc(5);
free(p);
if (p == NULL) /* NOTE: even without dereferencing, this may have UB */
{

}
```

ISO / IEC 9899 : 2011 , 6.2.4 §2 :

> [...] .

.

char `p` . .

```
char *p = "hello world";
p[0] = 'H'; // Undefined behavior
```

`char` . .

```
char a[] = "hello, world";
char *p = a;

a[0] = 'H';
p[7] = 'W';
```

(,). .

.

```
int * x = malloc(sizeof(int));
*x = 9;
free(x);
free(x);
```

(7.20.3.2 C99) :

> calloc, malloc realloc free realloc .

printf

`printf` . .

```
long z = 'B';
printf("%c\n", z);
```

.

```
printf("%f\n",0);
```

. `%f` () double . 0 `int` .

(`clang` `gcc` `-Wformat`). :

```
warning: format specifies type 'double' but the argument has type
      'int' [-Wformat]
    printf("%f\n",0);
            ~~    ^
            %d
```

.

```
 char *memory_block = calloc(sizeof(uint32_t) + 1, 1);
 uint32_t *intptr = (uint32_t*)(memory_block + 1);  /* possible undefined behavior */
 uint32_t mvalue = *intptr;
```

. C11 *(6.3.2.3)* . `uint32_t` 2 4 .

`calloc` . `memory_block` `uint32_t` . `uint32_t` 2 4 `memory_block + 1` .

C . `memory_block + 1` .

`char *` .

`memcpy` .

```
memcpy(&mvalue, memory_block + 1, sizeof mvalue);
```

`uint32_t*` .

`mvalue` `mvalue` `mvalue` .

- `calloc` . `0` .
- `uint32_t` .
- .

.

```
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 (6.5.6).

.

```
char buffer[6] = "hello";
char *ptr3 = buffer + 6;  /* OK, pointing to just beyond */
char value = *ptr3;       /* undefined behavior */
```

const

```
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 9899 : 201x , 6.7.3 §2 :

> const-qualified lvalue const . [...]

---

(1) GCC throw : `warning: assignment discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]`

printf % s

`printf` `%s` . .

```
char *foo = NULL;
printf("%s", foo); /* undefined behavior */
```

. . , Glibc .

```
(null)
```

.

```
char *foo = 0;
printf("%s\n", foo); /* undefined behavior */
```

GCC `printf("%s\n", argument);` `printf("%s\n", argument);` `puts` `puts(argument)` `puts` Glibc . .

null . . :

```
char *foo = "";
printf("%s\n", foo);
```

```
extern int var;
```

```
static int var; /* Undefined behaviour */
```

C11, §6.2.2, 7 :

.

. *C11, §6.2.2, 4* :

31) . . .

```
/* 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;
```

fflush

POSIX C `fflush` . `fflush` .

```
#include <stdio.h>

int main()
{
    int i;
    char input[4096];

    scanf("%i", &i);
    fflush(stdin); // <-- undefined behavior
    gets(input);

    return 0;
}
```

. , `stdin` `fflush` . Microsoft `fflush` . `fflush` . POSIX.1-2008 `fflush` .

`fflush(stdin)` .

1 :

```
int x = 5 << -3; /* undefined */
int x = 5 >> -3; /* undefined */
```

.

```
int x = -5 << 3; /* undefined */
```

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;
```

(.eg `-5 >> 3`) .

---

1 *ISO / IEC 9899 : 201x* , 6.5.7 :

> .

getenv, strerror setlocale

`getenv()` , `strerror()` `setlocale()` . .

getenv () , C11, §7.22.4.7, 4 .

> getenv . getenv .

strerror () , C11, §7.23.6.3, 4 .

> strerror , . strerror .

setlocale () , C11, §7.11.1.1, 8 :

> setlocale . , setlocale .

`localeconv()` `struct lconv` .

localeconv () , C11, §7.11.2.1, 8 .

> localeconv . localeconv .

`_Noreturn` `noreturn`

C11

`_Noreturn` C11 . `<stdnoreturn.h>` `noreturn` `_Noreturn` . `<stdnoreturn.h>` `_Noreturn` `noreturn` `_Noreturn` .

`_Noreturn` (`noreturn`) . .

`func()` `noreturn` .

```
#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 clang .

```
$ 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]
}
^
```

---

noreturn :

```
#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;
}
```

: https://riptutorial.com/ko/c/topic/364/--

51:

C (ISO-IEC 9899-2011) . 6 ( ).

ISO-IEC 9899-2011 .

, ,

(C , " " .)

. . C .

. , ( ) . .

Examples

C 1 . .

```
void foo(int bar)
{
    int var;
    double var;
}
```

. .

. .

---

1) :

C99

, 6.7.2.3 ( ) .

`+` `-` . struct . :

```
struct foo
{
    bool bar;
};

void baz(void)
{
   struct foo testStruct;
   -testStruct; /* This breaks the constraint so must produce a diagnostic */
}
```

: https://riptutorial.com/ko/c/topic/7397/-

52:

Examples

C

static 0 . (register) [1] (,) .

.

```
int x = 1;
char squota = '\'';
long day = 1000L * 60L * 60L * 24L; /* milliseconds/day */
```

static , [2] . .

register , . .

.

```
int binsearch(int x, int v[], int n)
{
   int low = 0;
   int high = n - 1;
   int mid;
   ...
}
```

```
   int low, high, mid;

   low = 0;
   high = n - 1;
```

, ...

:

.

, .

```
int days_of_month[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }
```

(1 0 .)

. 12 .

0.

.   GCC   .

C99

C89 / C90   C   .

C99

C99   0 .

:

.   .

```
char chr_array[] = "hello";
```

:

```
char chr_array[] = { 'h', 'e', 'l', 'l', 'o', '\0' };
```

, 6 (5 + `'\0'`).

1 C   ? ?

2   . , `int global_var = f();` .   `const`   . C `const` " " " " . `const int SIZE = 10; int global_arr[SIZE];` `const int SIZE = 10; int global_var = SIZE;` C .

.

```
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
};
```

, (1990 ,)   :

```
struct Date uk_battles[] =
{
    1066, 10, 14,    // Battle of Hastings
    1815,  6, 18,    // Battle of Waterloo
    1805, 10, 21,    // Battle of Trafalgar
};
```

.   .

.

C99

C99   . ,   .

`int`   :

```
int array[] = { [4] = 29, [5] = 31, [17] = 101, [18] = 103, [19] = 107, [20] = 109 };
```

`=`   .   0 .   . . . .   . (ISO / IEC 9899 : 2011, §6.7.9 , ¶19)   ,   .

. 21 .   .

`. element` :

```
struct Date
{
    int year;
    int month;
    int day;
};

struct Date us_independence_day = { .day = 4, .month = 7, .year = 1776 };
```

(0).

.

C89

C `union` . C89 / C90 `union` .

```
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   `du` .

```
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 };
```

.   .

```
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",
          }
 };
```

GCC   .

```
int array[] = { [3 ... 7] = 29, 19 = 107 };
```

( ) .

: https://riptutorial.com/ko/c/topic/4547/

53:

. . C `/* */` `//` . .

- `/*...*/`
- `//...` (C99)

Examples

/ * * /

(`/*`), (`*/`) . ( ) .

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

. `/*` .

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

`/* */` .

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

.

`/*` , :

```
/* 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 */
    }
}
```

. `/*` ( ) `*/` . .

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

. .

//

C99

C99 C ++ . .

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

.

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

.

```
// 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
}
```

`#if 0` `#endif` " " . .

```
#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) */
...
```

C99

`//` , . :

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

`/` `\` . `??/` .

`??/` trigraph \ . , . , .

```
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;
```

: https://riptutorial.com/ko/c/topic/10670/

54:

. . / .

C . , Unity C . C ++ C . C ++ .

:

TDD - :

C :

1.
2.
3.
4.

C ++ C ++ : C C ++ .

Examples

Cpp Testest

CppUTest C C ++ xUnit . C ++ . , Google . Visual Studio Eclipse CDT .

```
#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);
}
```

`setup()` `teardown()` . `setup` `teardown()` . . .

```
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);
    }
}
```

Unity xUnit . C , , , . .

.

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

.

```
#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 , makefile Ruby .

CMocka

CMocka mock C . C ( ) . mock, API .

```
#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;
}
```

: https://riptutorial.com/ko/c/topic/6779/--

55:

- #include <stdio.h> / * . * /
- FILE * fopen (const char * path, const char * mode); / * * /
- FILE * freopen (const char * path, const char * , FILE * stream); / * * /
- int fclose (FILE * stream); / * * /
- size_t fread (void * ptr, size_t size, size_t nmemb, FILE * stream); / * *size* *nmemb* *ptr* . read . * /
- size_t fwrite (const void * ptr, size_t size, size_t nmemb, FILE * stream); / * *size* *nmemb* *ptr* . . * /
- int fseek (FILE * stream, long offset, int whence); / * *whence* , 0 . * /
- long ftell (FILE *); / * (offset) . * /
- void rewind (FILE * stream); / * . * /
- int fprintf (FILE * fout, const char * fmt, ...); / * fout printf * /
- FILE * stdin; / * * /
- FILE * stdout; / * * /
- FILE * stderr; / * * /

const char *	. .
int where	`SEEK_SET` , `SEEK_END` , `SEEK_CUR` . : `SEEK_END` .

:

`fopen()` `freopen()` .

- `"r"` : .
- `"r+"` : - .
- `"w"` : . 0 . .
- `"w+"` : 0 / . .
- `"a"` : .
- `"a+"` : - . - . .

`b` (: `"rb"` `"a+b"` `"ab+"`). `b` 2 L. . Windows . (Windows `fopen` ' ' `b` `t` .)

C11

- `"wx"` : . .
- `"wbx"` : . .

`x` `x` .

Examples

```
#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;
}
```

main , `output.txt` . . `fopen()` .

`fopen()` `NULL` `errno` `errno` . `fopen()` `fopen()` `perror()` .

`fopen()` `FILE` . `fclose()` .

`fputs()` . `fopen()` `fputs()` `errno` . `EOF` .

`fclose()` `FILE *` . `fputs()` `fputs()` '0') `errno` .

fprintf

`printf` `fprintf` . ,

```
/* 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);
}
```

: (Windows) "" . UNIX \ n Windows \ r ( ) \ n ( ) . CRLF. C . \ n AC .

Windows \n \r \n UNIX .

```
#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;
}
```

`popen()`) (`netstat`) .

: `popen()` C POSIX C .

getline ()

POSIX C `getline()` . , .

`example.txt` :

```
#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;
}
```

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:
```

`getline()` . `getline()` . `getline()` . .

`getdelim()` . `getline()` . '\n' . (`"\r\n"`) '\ `"\r\n"`) () `getline()` Windows .

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
```

```
#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;
}
```

`fwrite` `output.bin` `output.bin` .

.

. .

, (16, 32 64) . . C 2 ( ). unsigned 2 . .

```
/* 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;
}
```

.

fscanf ()

.

file.txt :

```
This is just
a test file
to be used by fscanf()
```

.

```
#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++;
    }
}
```

.

```
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()
```

`stdio.h` `fgets()` . . `n - 1` (`'\n'`) (EOF) .

```
#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);
    }
}
```

.

```
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.
```

.

```
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]:
```

. `fgets()` .

POSIX `getline()` . , .

: https://riptutorial.com/ko/c/topic/507/---

56:

C ( ). C ( ). c11 §5.1.1.2 .

`.c`	. .
`.h`	. .
`.o`	. .
`.obj`	.
`.a`	. .
`.dll`	(Windows).
`.so`	().
`.dylib`	OSX (Unix).
`.exe` , `.com`	Windows . . .

POSIX c99	
`-o filename`	. (`bin/program.exe` , `program`)
`-I directory`	`direrctory` `direrctory` .
`-D name`	`name`
`-L directory`	`directory` .
`-l name`	`libname` .

POSIX (Linux, , Mac) `c99` .

- C99 - C .

GCC (GNU)	
`-Wall`	.
`-Wextra`	.
`-pedantic`	.
`-Wconversion`	.

GCC (GNU)	
-c	.
-v	.

- gcc POSIX .
- POSIX (clang ,) .
- GCC .

TCC (Tiny C)	
-Wimplicit-function-declaration	.
-Wunsupported	TCC GCC .
-Wwrite-strings	char * const char * .
-Werror	.
-Wall	-Werror , -Wunusupported -Wwrite strings .

Examples

(.o) . . .

(, , . . , . "" . .

. .a .so .

ld (: Linux collect2). . .

```
% gcc foo.o bar.o baz.o -o myprog
```

3 (foo.o , bar.o baz.o) myprog . myprog . .

, . , Linux Solaris, AIX, macOS, Windows . GCC gcc -v .

. gcc foo.o bar.o ncurses .

```
% gcc foo.o bar.o -o foo -lncurses
```

.

```
% gcc foo.o bar.o /usr/lib/libncurses.so -o foo
```

(libncurses.so libncurses.a ar). (-lname). . .

(`<math.h>`) `-lm` Mac OS X macOS Sierra . macOS . POSIX POSIX , . .

C . `-Wall` .

```
% gcc -Wall -c foo.cc
```

`-Wall` .

(,) , `-Wall` :

```
% gcc -Wall -Wextra -Wfloat-equal -Wundef -Wcast-align -Wwrite-strings -Wlogical-op \
>     -Wmissing-declarations -Wredundant-decls -Wshadow …
```

`clang` `-Weverything` `clang` .

C 5 .

1. : `.c` `.c` `.c` . : `.cc` `.cpp` C ++ . C .
   : `foo.c`
2. : ( ). . `.h` .
   : `foo.h`
3. : . . (: Windows, MS-DOS) `.obj` `.o` .
   : `foo.o` `foo.obj`
4. : "" . . Windows `.exe` Unix .
   : `foo` `foo.exe`
5. : (, `main()`). . . (: `#include <library.h>`). . .
   - : (`.lib` DLL POSIX `.a` Windows `.lib`) . . .
     : `libfoo.a` `foo.lib`
   - : (POSIX `.so` , OSX `.dylib` Windows `.dll`) . . . ( ) .
     : `foo.so` `foo.dylib` `foo.dll`

C . . . . " " " " . , .

("#") . . .

1. :

   `#define` . ,

   ```
   #define BIGNUM 1000000
   int a = BIGNUM;
   ```

   ```
   int a = 1000000;
   ```

   `#define`

. . #define .

#define . :

```
#define ISTRUE(stm) do{stm = stm ? 1 : 0;}while(0)
// in the function:
a = x;
ISTRUE(a);
```

.

```
// in the function:
a = x;
do {
    a = a ? 1 : 0;
} while(0);
```

#define . .

.

2. :

   #include . :

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

   <stdio.h> #include . #include #define . printf scanf #include . stdio.h . C . #include C .

3. :

```
#if defined A || defined B
variable = another_variable + 1;
#else
variable = another_variable * 2;
#endif
```

   :

```
variable = another_variable + 1;
```

   A B , :

```
variable = another_variable * 2;
```

   . / .

4. . // /* */ .

C . C .o . . , #include (D #include). .

C . GCC .

```
% gcc -Wall -c foo.c
```

`%` OS . `foo.c` `foo.o` . `-c` . `-c` POSIX (: Linux macOS) . .

.

```
% gcc -Wall foo.c -o foo
```

`foo.c` `foo` . `-o` 2 () . `-o` (`gcc foo.c`), `a.out` .

`.c` .

1. - `.c` `#include` `#define` .
2. **compilation** - (`-S`).
3. **assembly** - .
4. - .

"GNU C " "GNU " GCC. C . , "C " `cc` . . Linux `cc` GCC . macOS OS-X clang .

POSIX `c99` C . C99 . POSIX `c89` . POSIX `-c` `-o` .

: `gcc` `-Wall` . (: `-Wextra` .

§5.1.1.2 C 2011 (:) 8 .

1. ( ). Trigraphs .
2. (`\`) .
3. .
4. , , pragma . `#include` 1 - 4 ( ) . .
5. .
6. .
7. .
8. .

C .

: https://riptutorial.com/ko/c/topic/1337/

57: /

Examples

p . void -pointer , void void* ("casted *").

```
#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>` `uintptr_t`

C99 `uintptr_t` `<inttypes.h>` .

```
#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;
}
```

`uintptr_t` . . `uintptr_t` . .

`uintptr_t` `intptr_t` . .

K & R C89

:

K & R-C C89 `void*` (`<stdlib.h>` `int main(void)` .) `long unsigned int` `lx` / .

. .

```
#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;
}
```

* 1 . d .

C99 <stddef.h> ptrdiff_t " " . ptrdiff_t t .

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;
}
```

.

```
p1 = 0x7fff6679f430
p2 = 0x7fff6679f434
p2 - p1 = 1
```

(int . int 4.

---

* 1 .

i , d	int	
u		
o		8 .
x		16 , .

X		16 , .
f		float 6 (nan inf infinity)
F		float 6 (NAN INF INFINITY).
e		/ float 6 .
E		/ float 6 .
g		f e *[]*
G		F E *[]*
a		16 , .
A		16 , .
c		.
s	*	NUL , .
p	*	void -pointer ; void -pointer void* (""). , .
%		% .
n	int *	int .

ISO / IEC 9899 : 2011 §7.21.6.1 ¶7 %n (: %hhn *n* *signed char*).

float double . §6.5.2.2 , ¶7 . .) printf() printf() float double .

g G e f (E F) C printf() POSIX .

double f e (G F E) . 0 P , 6, 0 1. E X :

- P> X> = -4 , f (F) P - (X+1) .
- , e (E) P - 1 .

, '#' , 0 .

printf ()

<stdio.h> printf() C .

```
printf("Hello world!");
// Hello world!
```

.

```
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
```

, , , % .

printf() . . .

, int . , .

C99 C11 printf() . .

hh	d, i, o, u, x X	char , signed char unsigned char
h	d, i, o, u, x X	short int unsigned short int
	d, i, o, u, x X	long int unsigned long int
	a, A, e, E, f, F, g G	double (scanf() , C90)
	d, i, o, u, x X	long long int unsigned long long int
j	d, i, o, u, x X	intmax_t uintmax_t
	d, i, o, u, x X	size_t (POSIX ssize_t)
	d, i, o, u, x X	ptrdiff_t
	a, A, e, E, f, F, g G	long double

.

Microsoft hh , j , z t .

I32	d, i, o, x X	__int32
I32	o, u, x X	unsigned __int32
I64	d, i, o, x X	__int64
I64	o, u, x X	unsigned __int64
	d, i, o, x X	ptrdiff_t (32 __int32 , 64 __int64)
	o, u, x X	size_t (32 unsigned __int64 unsigned __int32 , 64 unsigned __int64

		)
L L	a, A, e, E, f, g G	`long double` (Visual C ++ `long double` `long double` `double` .)
l w	C C	`printf` `wprintf` . (`lc` , `lC` , `wc` `wC` `printf` `C` `wprintf` `c` .)
l w	s, S Z	`printf` `wprintf` . (`ls` , `lS` , `ws` `wS` `printf` `S` `wprintf` `s` .)

`C` , `S` `Z` `I` , `I32` , `I64` `w` Microsoft . `l` `long double` `double` , `long double` `double` .

C (C11 C99) `printf()` .

`-`		. .
`+`		('+' '-') . .
`<space>`		`<space>` . , `<space>` ' `+` ' `<space>` .
`#`		. `o` , 0 (0 0). `x` `X` , 0 `0x` (`0X`) . , , `a` `A` `e` , `E` , `f` , `F` , `g` `G` , , . . `g` `G` 0 . .
`0`		d, i, o, u, x, X, a, A, e, E, f, F, g G 0 ( ) NaN . '0' '-' '0' . d, i, o, u, x X , '0' . "0 ' `<apostrophe>` . .

Microsoft .

`printf()` POSIX .

'	i, d, u, f, F, g, G	10 . . .

/ : https://riptutorial.com/ko/c/topic/3750/-----

58:

.

- < > * < >;
- int * ptrToInt;
- void * ptrToVoid; / * C89 + * /
- struct someStruct * ptrToStruct;
- int ** ptrToPtrToInt;
- int arr [length]; int * ptrToFirstElem = arr; / * <C99 'length' ,> = C11 . * /
- int * arrayOfPtrsToInt [length]; / * <C99 'length' ,> = C11 . * /

.

```
/* The * operator binds to right and therefore these are all equivalent. */
int *i;
int * i;
int* i;
```

.

```
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 */
```

.

```
int *foo[2]; /* foo is a array of pointers, can be accessed as *foo[0] and *foo[1] */
```

Examples

, .

NULL . . .

, :

```
struct SomeStruct *s = malloc(sizeof *s);
s->someValue = 0; /* UNSAFE, because s might be a null pointer */
```

:

```
struct SomeStruct *s = malloc(sizeof *s);
if (s)
{
    s->someValue = 0; /* This is safe, we have checked that s is valid */
}
```

sizeof

/ . (`char`) / . `sizeof(int)` 4 . `sizeof(that_type)` `sizeof(*var_ptr_to_that_type)` .

:

```
int *intPtr = malloc(4*1000);    /* allocating storage for 1000 int */
long *longPtr = malloc(8*1000);  /* allocating storage for 1000 long */
```

:

```
int *intPtr = malloc(sizeof(int)*1000);     /* allocating storage for 1000 int */
long *longPtr = malloc(sizeof(long)*1000);  /* allocating storage for 1000 long */
```

:

```
int *intPtr = malloc(sizeof(*intPtr)*1000);     /* allocating storage for 1000 int */
long *longPtr = malloc(sizeof(*longPtr)*1000);  /* allocating storage for 1000 long */
```

`free` `free` . . .

.

1. `malloc` (`calloc`)
2.
3. `free`

`free` (dangling) `malloc` (wild pointer) , `free` ("double free") .

. OS .

Valgrind .

. :

```
int* myFunction()
{
    int x = 10;
    return &x;
}
```

`x` ( ) . `myFunction` . `myFunction` `x` . `x` (`&x`) . .

. .

`malloc` . .

```
#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;
}
```

/

`*p++` `p` , .

/ . p p .

(*p)++ p .

: *p-- p p .

Dereferencing

```
int a = 1;
int *a_pointer = &a;
```

a_pointer a .

```
*a_pointer = 2;
```

.

```
printf("%d\n", a); /* Prints 2 */
printf("%d\n", *a_pointer); /* Also prints 2 */
```

, NULL .

```
int *p1, *p2;

p1 = (int *) 0xbad;
p2 = NULL;

*p1 = 42;
*p2 = *p1 + 1;
```

. p1 0xbad 0xbad . ? . . p2 NULL) .

:

```
struct MY_STRUCT
{
    int my_int;
    float my_float;
};
```

struct MY_STRUCT struct MY_STRUCT . .

```
typedef struct MY_STRUCT MY_STRUCT;
```

```
MY_STRUCT *instance;
```

instance null . . MY_STRUCT . :

```
MY_STRUCT info = { 1, 3.141593F };
MY_STRUCT *instance = &info;
```

.

```
int a = (*instance).my_int;
float b = instance->my_float;
```

, `->` , `*` . .

.

```
MY_STRUCT copy = *instance;
copy.my_int    = 2;
```

`copy` `instance` `instance` . copy my_int `instance` .

```
MY_STRUCT *ref = instance;
ref->my_int    = 2;
```

`ref` `instance` . `my_int` `instance` .

. . .

.

.

```
int my_function(int a, int b)
{
    return 2 * a + 3 * b;
}
```

.

```
int (*my_pointer)(int, int);
```

.

```
return_type_of_func (*my_func_pointer)(type_arg1, type_arg2, ...)
```

.

```
my_pointer = &my_function;
```

.

```
/* 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);
}
```

, & * . .

```
/* Attribution without the & operator */
my_pointer = my_function;

/* Dereferencing without the * operator */
int result = my_pointer(4, 2);
```

typedef .

```
typedef void (*Callback)(int a);

void some_function(Callback callback)
{
    int a = 4;
    callback(a);
}
```

C ( ) . .

```
void some_function(void callback(int))
{
    int a = 4;
    callback(a);
}
```

. .

```
#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 */

    ...
}
```

NULL . .

:

NULL . NULL . . .

NULL , NULL .

:

NULL (void *)0 . 0x0 . NULL C-faq .

NULL .

```
int i1;

int main()
{
    int *p1 = &i1;
    const char *p2 = "A constant string to point to";
    float *p3 = malloc(10 * sizeof(float));
}
```

(&)

(, , , , ,) .

```
int i = 1;
int *p = NULL;
```

p = &i; i p .

i p .

printf("%d\n", *p); i 1 .

:

void * .

void* . malloc . .

```
void* malloc(size_t);
```

void-to-void malloc .

```
int* vector = malloc(10 * sizeof *vector);
```

void . malloc() stdlib.h malloc() . DRY ( ) void . :

```
int* vector = (int*)malloc(10 * sizeof int*);
```

,

```
void* memcpy(void *restrict target, void const *restrict source, size_t size);
```

void * . .

```
unsigned char buffer[sizeof(int)];
int b = 67;
memcpy(buffer, &b, sizeof buffer);
```

Const

- int

  int . int b b b b 100 .

  ```
  int b;
  int* p;
  p = &b;    /* OK */
  *p = 100;  /* OK */
  ```

- const int

  int .

  ```
  int b;
  const int* p;
  p = &b;    /* OK */
  *p = 100;  /* Compiler Error */
  ```

- int const

  int int .

  ```
  int a, b;
  int* const p = &b; /* OK as initialisation, no assignment */
  *p = 100;  /* OK */
  p = &a;    /* Compiler Error */
  ```

- const const int const

  int int .

  ```
  int a, b;
  const int* const p = &b; /* OK as initialisation, no assignment */
  p = &a;   /* Compiler Error */
  *p = 100; /* Compiler Error */
  ```

- int

  p1 p (int* p1 (int int) int) .

  p1 int a int a . a 100 .

```
void f1(void)
{
  int a, b;
  int *p1;
  int **p;
  p1 = &b; /* OK */
  p = &p1; /* OK */
  *p = &a; /* OK */
  **p = 100; /* OK */
}
```

- `const int`

```
 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' */
}
```

- `const int`

```
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   const`

```
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 const`

```
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

```
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 const const

```
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 */
}
```

,

C C ++   (*)   .

*   .

```
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' */
```

( ) *   .

```
*p = 123;
/* 'p' was pointing to 'i', so this changes value of 'i' to 123 */
```

*   .

```
p = &another_variable;
```

C .

```
int *p = &i;
```

`int i = 5;` `int i = 5;` `int i; i = 5;` , `int *p = &i;` `int *p; *p = &i;` . , `int *p; *p = &i;` UB . `*` .

(*) C . , . , , .

P Q . , . .

C .

```
#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;
}
```

.

```
#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! */

  ...
```

(*) .

```
int *pointer; /* inside a function, pointer is uninitialized and doesn't point to any valid
object yet */
```

. ,

```
int *iptr1, *iptr2;
int *iptr3,  iptr4;  /* iptr3 is a pointer variable, whereas iptr4 is misnamed and is an int
*/
```

Z w (&) | .

```
int value = 1;
pointer = &value;
```

(*) .

```
printf("Value of pointed to integer: %d\n", *pointer);
/* Value of pointed to integer: 1 */
```

, `->` .

```
SomeStruct *s = &someObject;
s->someMember = 5; /* Equivalent to (*s).someMember = 5 */
```

C . () .

```
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 */
```

null

```
pointer = 0;     /* or alternatively */
pointer = NULL;
```

. . null .

```
if (!pointer) exit(EXIT_FAILURE);
```

. . .

.

```
*pointer += 1;
printf("Value of pointed to variable after change: %d\n", *pointer);
/* Value of pointed to variable after change: 2 */
```

. , .

```
int value2 = 10;
pointer = &value2;
printf("Value from pointer: %d\n", *pointer);
/* Value from pointer: 10 */
```

. , `long int` `short int` . C .

.

```
#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 C11 Microsoft Visual Studio `%zu` `%Iu` [1] .)

.

C

C . C . . 3 double base , `*ptr` double , `*(ptr + 1)` , `*(ptr + 2)` double . `[]` .

```
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]);
```

ptr . .

```
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])
}
```

. . (+ 1).

---

1 : Microsoft `printf()` .

void

`qsort()` void .

```
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 */
```

void . `qsort()` () .

void . `qsort()` .

. `qsort()` . (polymorphic) void .

```
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;
}
```

qsort float , void float .

"len" "" qsort .

```
qsort(array, len, sizeof(array[0]), compare_floats);
```

: https://riptutorial.com/ko/c/topic/1108/

59:

- #include <math.h>
- double pow (double x, double y);
- float powf (float x, float y);
- long double powl (long double x, long double y);

1. `-lm` gcc .
2. `errno` 0 . `feclearexcept(FE_ALL_EXCEPT);` . , `errno` 0 0 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW);` `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW);` . math_error .

Examples

: fmod ()

`x/y` . x .

```
#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
```

: .

```
#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
```

long : fmodf (), fmodl ()

C99

`x/y` . X .

:

```
#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
```

:

```
#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
```

- pow (), powf (), powl ()

pow () *1 + 4 (3 + 3 ^ 2 + 3 ^ 3 + 3 ^ 4 + ... + 3 ^ N)* .

```
#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
```

: https://riptutorial.com/ko/c/topic/3170/-

60: (IPC)

(IPC) . C IPC . . POSIX IPC . Windows . .

Examples

. POSIX .

1. 'System V `semctl()` , `semop()` , `semget()` .
2. 'POSIX '- `sem_close()` , `sem_destroy()` , `sem_getvalue()` , `sem_init()` , `sem_open()` , `sem_post()` , `sem_trywait()` , `sem_unlink()` .

System V IPC Unix System V .

. POSIX `#include <sys/types.h>` ; POSIX .

```
#include <sys/sem.h>
```

.

```
#define KEY 0x1111
```

IPC . .

.

```
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
};
```

try (`semwait`) *raise* (`semsignal`) . P V .

```
struct sembuf p = { 0, -1, SEM_UNDO}; # semwait
struct sembuf v = { 0, +1, SEM_UNDO}; # semsignal
```

, IPC ID .

```
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 */
}
```

1 .

```
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 */
}
```

,   .   semop()   .

```
if (semop(id, &p, 1) < 0) {
    /* error handling code */
}
```

&p &v .

```
if (semop(id, &v, 1) < 0) {
    /* error handling code */
}
```

0   -1 .   .

Example 1.1 :

fork   .

```
#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
```

.

Example 1.2 :

1.1 .

```
#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
```

.

(IPC) : https://riptutorial.com/ko/c/topic/10564/----ipc-

61:

C . . . .

Examples

C .

```
#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;
}
```

```
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;
  }
}
```

`[]` `static` (, `size`) . `size` .

`getListOfFriends()` :

```
#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;
}
```

▪

C .

```
#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;
}
```

. .

```
#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;
}
```

. .

C . . .

```
#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;
}
```

. ( ) int .

```
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 */
}
```

struct . .

```
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;
}
```

.

```
int usingFunc (int iValue)
{
    RetValue iRet = func (iValue);

    if (iRet.iStat == 1) {
        /* do things with iRet.iValue, the returned value */
    }
    return 0;
}
```

.

```
int usingFunc (int iValue)
{
    RetValue iRet;

    if ( (iRet = func (iValue)).iStat == 1 ) {
        /* do things with iRet.iValue, the returned value */
    }
    return 0;
}
```

: https://riptutorial.com/ko/c/topic/1006/--

62:

. .

- returnType (* name) ( )
- typedef returnType (* name) ( )
- typedef returnType ( );
  * ;
- typedef returnType ( );
  typedef * NamePtr;

Examples

```
#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;
}
```

```
#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;
}
```

typedef

typedef .

typedef .

```
typedef returnType (*name)(parameters);
```

:

compare sort .

- compare - .

"compare" 0 "" . "").

typedef .

```
void sort(int (*compare)(const void *elem1, const void *elem2)) {
    /* inside of this block, the function is named "compare" */
}
```

typedef .

```
typedef int (*compare_func)(const void *, const void *);
```

sort .

```
void sort(compare_func func) {
    /* In this block the function is named "func" */
}
```

sort .

```
int compare(const void *arg1, const void *arg2) {
    /* Note that the variable names do not have to be "elem1" and "elem2" */
}
```

. `typedef struct something_struct *something_type` . API (: stdio.h `FILE`) ( ).

▪

void * .

```
/* 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);
}
```

.

qsort() , . .

---

char int C . ., . , graph() , 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)
```

., ., double double double . sin() , cos() , tan() , exp() graph() exp() !

---

graph() ? .

```
double (*fn)(double); // fn is a pointer-to-function that takes a double and returns one
```

: . ! typedef ( ) .

C . "" .

.

returnType (* name) ( )

.

1.

. `returnType name(parameters)`
2. . `returnType (*name)(parameters)`

int , **char** , **float** , **array / string** , **struct** .

, / .

.

```
int addInt(int n, int m){
    return n+m;
}
```

.

```
int (*functionPtrAdd)(int, int) = addInt; // or &addInt - the & is optional
```

void .

```
void Print(void){
    printf("look ma' - no hands, only pointers!\n");
}
```

.

```
void (*functionPtrPrint)(void) = Print;
```

.

```
sum = (*functionPtrAdd)(2, 3); //will assign 5 to sum
(*functionPtrPrint)(); //will print the text in Print function
```

. ""( ) **typedef** .

```
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;
}
```

.   "" :

```
int (*array[2]) (int x, int y); // can hold 2 function pointers
array[0] = Add;
array[1] = Multiply;
```

.

.   .   .

: https://riptutorial.com/ko/c/topic/250/-

63:

C . .

, , ...

Examples

C .

- (TU) .
- .
- .
- (IWYU)

  C C ++ C . TU (`code.c` `code.c`) (`"headerA.h"`) `code.c` TU `#include "headerA.h"` (`"headerB.h"` ,) `"headerA.h"` .

.

(TU) . ' (idempotence)' . . `#include <stdio.h>` .

`#pragma once` .

C . .

```
#ifndef UNIQUE_ID_FOR_HEADER
#define UNIQUE_ID_FOR_HEADER
```

`#endif` .

```
#endif /* UNIQUE_ID_FOR_HEADER */
```

`#include` .

. `HEADER_H_INCLUDED` . (: `<stdio.h>` `#ifndef BUFSIZ` .

MD5 ( ) . . .

`#pragma once`

`#pragma once` . .

```
#pragma once
```

`#pragma once` MS Visual Studio GCC Clang . . (C89) pragma (' pragma ') GCC .

. , `header.h` (`#include "header.h"`) .

: .

.

AT & T Indian Hill C ,

. `#include` . `#include` .

.

. `header.h` , .

```
#include "header.h"
```

( ) `header.h` .

. 10 2 " " .

NASA (GSFC) C . . . . . .

GSFC .

> §2.1.1
>
> `#include` . `#include` `#include` .
>
> `#include` . `#includes` . #ifdef .
>
> `#include` makefile . . .
>
> `#include` . .
>
> `#ifdef` .

.

- `header.h` `extra.h` `header.h` `extra.h` `extra.h` `extra.h` .
- `header.h` `notneeded.h` `header.h` `notneeded.h` `notneeded.h` (,).
- ( ).

`chkhdr` .

. . .

, `FILE *` (`<stdio.h>`) `<stdio.h>` . `size_t` , `<stddef.h>` . , `size_t` `<stddef.h>` .

.

. . , C `<standard-ch>` . , `<locale.h>` `<tgmath.h>` .

- C .

(IWYU)

Google IWYU .

`source.c` `arbitrary.h` `freeloader.h` , `freeloader.h` . . `arbitrary.h` `freeloader.h` . , `source.c` IWYU . `source.c` `freeloader.h` -, `#include "freeloader.h"` . (Idempotency .)

IWYU . . API .

C ++ . `file.cpp` `header2.h` `header1.h` . `file.cpp` `header2.h` . . `header1.h` `header2.h` . `header1.h` `header2.h` , `file.cpp` .

IWYU `header2.h` `file.cpp` . . C .

C `#include <header.h>` `#include "header.h"` .

[`#include <header.h>`] < > . .

[`#include "header.h"`] "…" `#include "header.h"` . . [`#include <header.h>`] .

. . , POSIX . . ( ) .

`#include` . .

.

```
#include <openssl/ssl.h>
#include <sys/stat.h>
#include <linux/kernel.h>
```

( ). (`sys` `linux`).

, .

`#include "../include/header.h"` .

. .

: . .

`static inline` `static` .

- .
- . .
-

( ).

- (`static`) `inline` .

- C ?
- C C ++
- C99 `static` `extern` `inline` ?
- `extern` ?
- `"../include/header.h"` ?
-
- ?

: https://riptutorial.com/ko/c/topic/6257/----

S. No		Contributors
1	C	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	-	Alejandro Caro, Jonathan Leffler, Roland Illig, Toby
3	2D	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-	Cimbali, Jens Gustedt, John Bollinger, Leandros, MD XF, mpromonet, poolie, RamenChef, technosaurus, templatetypedef , Toby
7		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
8		EsmaeelE, Jarrod Dixon, Jedi, Jesferman, Jonathan Leffler, Liju Thomas, MayeulC, tilz0R
9		Jens Gustedt, John Bollinger, P.P.
10		dylanweber, ganchito55, haccks, hexwab, Jonathan Leffler, Leandros, Malcolm McLean, MikeCAT, Toby
11		Jossi, RamenChef, Toby, Vality
12		Jonathan Leffler, PassionInfinite, Toby

13		2501, alk, Blagovest Buyukliev, Firas Moalla, Jens Gustedt, Keith Thompson, Ken Y-N, Leandros, P.P., Peter, WMios
14		Parham Alvani, Toby
15		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
16		4386427, A B, alk, drov, dvhh, Jonathan Leffler, Malcolm McLean, Shog9, syb0rg, Toby, Woodrow Barlow, Yotam Salmon
17		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
18	/ : for, while, do-while	alk, GoodDeeds, Jens Gustedt, jxh, L.V.Rao, Malcolm McLean, Nagaraj, RamenChef, reshad, Toby
19		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
20		2501, Blacksilver, eush77, Jean-Baptiste Yunès, Jonathan Leffler, Leandros, mirabilos, syb0rg, Toby
21		alk, haccks, Jens Gustedt, Kerrek SB
22		alk, Bob__, Braden Best, Chrono Kitsune, dhein, Insane, Jens Gustedt, Magisch, Mateusz Piotrowski, Peter, Toby
23		EsmaeelE, Jonathan Leffler, L.V.Rao, madD7, RamenChef, Sirsireesh Kodali, Toby
24		alk, EvilTeach, Fantastic Mr Fox, haccks, Ishay Peled, Jens

		Gustedt, John Odom, Jonathan Leffler, Lundin, madD7, Paul Hutchinson, RamenChef, Rishikesh Raje, Toby, vkgade
25		alk, AnArrayOfFunctions, Blacksilver, Firas Moalla, J Wu, Jens Gustedt, Jonathan Leffler, Jonathon Reinhart
26		Ashish Ahuja, foxtrot9, Kerrek SB, Toby
27		alk, bevenson, Blagovest Buyukliev, Faisal Mudhir, GoodDeeds, gsamaras, jxh, L.V.Rao, lordjohncena, MikeCAT, NeoR, noamgot, OznOg, P.P., Toby, tofro
28	,	Jens Gustedt, Jonathan Leffler, Klas Lindbäck, Neui, Paul92, Toby
29	()	alk, Jens Gustedt, P.P.
30		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
31		2501, Armali, bta, Community, haccks, Jens Gustedt, John Bode, Toby
32		embedded_guy, Firas Moalla, Jean-Baptiste Yunès, Jens Gustedt, Jonathan Leffler
33		3442, alk, Dariusz, Jens Gustedt, Leandros, mirabilos
34		alk, Firas Moalla, Jens Gustedt, Jeremy Thien, kdopen, Lundin, Toby
35		2501, 4386427, Jens Gustedt
36		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
37		4386427, alk, Andrea Biondo, bevenson, iRove, Jonathan Leffler, Jossi, Leandros, Mateusz Piotrowski, Ryan, Toby
38		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

39		Alejandro Caro, alk, jasoninnn, Jens Gustedt, Jonathan Leffler, OznOg, Toby
40		Jens Gustedt, stackptr
41		Jens Gustedt
42		alk, Blagovest Buyukliev, Jens Gustedt, Jesferman, madD7, tversteeg
43		Alex, EsmaeelE, Jens Gustedt, Jonathan Leffler, Toby
44		bevenson, EsmaeelE, Jonathan Leffler
45		2501, Jens Gustedt, Sun Qingyao
46	C	Chandrahas Aroori, Jonathan Leffler, Nityesh Agarwal, Shubham Agrawal
47		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
48		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
49		alk, Jens Gustedt, Jonathan Leffler, lordjohncena, Malcolm McLean, Sourav Ghosh, syb0rg, Toby
50		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		Armali, Toby, Vality
52		Jonathan Leffler, Liju Thomas, P.P.

53		Ankush, Chandrahas Aroori, Jonathan Leffler, Toby
54		Community, EsmaeelE, Jonathan Leffler, lordjohncena, Toby, user2314737, vuko_zrno
55		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
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	/	alk, fluter, Jonathan Leffler, Jossi, lardenn, MikeCAT, polarysekt, StardustGogeta
58		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
59		Alejandro Caro, alk, Blagovest Buyukliev, immerhart, Jonathan Leffler, manav m-n, Toby
60	(IPC)	CLDSEED, EsmaeelE, Jonathan Leffler, Toby
61		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
62		Alejandro Caro, alk, David Refaeli, Filip Allberg, hlovdal, John Burger, Leandros, Malcolm McLean, P.P., Srikar, stackptr, Toby
63		4444, Jonathan Leffler, patrick96, Sirsireesh Kodali