

The C Language : **Common Programming** **Concepts**

David Bouchet

david.bouchet.epita@gmail.com

Integer Types

Signed Integers

```
(signed) char           // 8 bits
(signed) short          // 16 bits
(signed) int            // 32 bits
(signed) long           // 64 bits
(signed) long long     // 64 bits
```

Unsigned Integers

```
unsigned char           // 8 bits
unsigned short          // 16 bits
unsigned int            // 32 bits
unsigned long           // 64 bits
unsigned long long     // 64 bits
```

Sizes are given for 64-bit architectures ([LP64 data model on Linux](#)). 2/38

Floating-Point Types

IEEE 754 Standard

float	// 32 bits (single precision)
double	// 64 bits (double precision)

Other Types

Similar to Unsigned Integers

```
size_t // 64 bits
```

Used for size measurement
(e.g. sizes of arrays, array indexes)

Absence of Type

```
void
```

- Used as function return type when no return value is expected.
- Can be used as function parameter type when no parameters are passed into the function.

Variables

General syntax for declaration, definition and initialization

```
<type> <identifier> = <value>;
```

Examples

```
// Declaration only.  
// (Mostly used with global variables and multiple files).  
extern int x;  
  
int main()  
{  
    int a = 3;           // Declaration, definition, initialization.  
    a = 5;              // Reassignment.  
  
    unsigned char b;   // Declaration, definition.  
    b = 5;              // Initialization (first assignment).  
    b = 4;              // Reassignment  
  
    x = 10;             // Initialization (first assignment).  
  
    return 0;  
}  
  
int x;                 // Definition (usually in another file).
```

Constants

General syntax for declaration, definition and initialization

```
const <type> <identifier> = <value>;
```

Example

```
// Declaration only.  
// (Mostly used with global constants and multiple files).  
extern const int x;  
  
int main()  
{  
    const double a = 4.0;    // Declaration, definition, initialization.  
    const int b = 18;       // Declaration, definition, initialization.  
    const int c = x + b;    // Declaration, definition, initialization.  
  
    return 0;  
}  
  
const int x = 10;          // Definition, initialization (usually in another file).
```

Enumerations

Declaration

```
enum <enum_name>
{
    const_1;
    const_2;
    // ...
    const_N;
}
```

```
enum <enum_name>
{
    const_1 = 0;
    const_2 = 15;
    // ...
    const_N = 3;
}
```

Example

```
int main()
{
    enum color
    {
        red,           // 0
        green,         // 1
        blue,          // 2
    };

    enum color c1 = red;
    enum color c2 = green;
    enum color c3 = blue;

    return 0;
}
```

Functions

```
void f1();           // Declare f1()
short f2(void);     // Declare f2()
void f3(short i, float f); // Declare f3()

int main()          // Declare and define main(): the entry point
{
    short r;
    f1();           // Call f1() ; No parameters ; No return value
    r = f2();       // Call f2() ; No parameters ; Return value -> r
    f3(r, 3.5);     // Call f3() ; Two parameters ; No return value

    return 0;      // Must return an 'int' value.
}

void f1()           // Define f1()
{
}

short f2(void)      // Define f2()
{
    return 5;      // Must return a 'short' value.
}

void f3(short i, float f) // Define f3()
{
    short a = i + 1;
    float b = 3 * f;
    return;
}
```


The *main()* Function

The *main()* function is the entry point of the program.

It should return an 'int' value.

- If no error occurred → Should return 0
- If any error occurred → Should return a value different from 0

We can also use labels defined in `<stdlib.h>`:

- `EXIT_SUCCESS`
- `EXIT_FAILURE`

```
#include <stdlib.h>

int main()
{
    // Some instructions.
    // No error occurred.

    return EXIT_SUCCESS;
}
```

```
#include <stdlib.h>

int main()
{
    // Some instructions.
    // An error occurred.

    return EXIT_FAILURE;
}
```

Formatting and Printing Data (1)

```
#include <stdio.h>

int main()
{
    char c = 'A';    // ASCII code of 'A'.
    short h = 100;
    int i = 200;
    long l = 300;
    float f = 400.0;
    double d = 500.0;

    printf("c = %c\n", c);
    printf("c = %hhi\n", c);
    printf("c = 0x%hhx\n", c);
    printf("h = %hi\n", h);
    printf("i = %i\n", i);
    printf("l = %li\n", l);
    printf("f = %f\n", f);
    printf("d = %f\n", d);
    printf("string = %s\n", "hello");

    return 0;
}
```



```
c = A
c = 65
c = 0x41
h = 100
i = 200
l = 300
f = 400.000000
d = 500.000000
string = hello
```

See [printf\(3\)](#)


Formatting and Printing Data (2)

```
#include <stdio.h>

int main()
{
    unsigned char c = 'A';    // ASCII code of 'A'.
    unsigned short h = 100;
    unsigned int i = 200;
    unsigned long l = 300;
    size_t z = 400;

    printf("c = %c\n", c);
    printf("c = %hhu\n", c);
    printf("c = 0x%hhx\n", c);
    printf("h = %hu\n", h);
    printf("i = %u\n", i);
    printf("l = %lu\n", l);
    printf("z = %zu\n", z);

    return 0;
}
```



```
c = A
c = 65
c = 0x41
h = 100
i = 200
l = 300
z = 400
```

See [printf\(3\)](#)

Conditions and Relational Operators

No Boolean Type!

Conditions use integers

- 0 is equivalent to **FALSE**
- $\neq 0$ is equivalent to **TRUE**

```
#include <stdio.h>

int main()
{
    int a = 5, b = 10, c = 0;

    printf("a == 5 => %i\n", a == 5);
    printf("a != 5 => %i\n", a != 5);
    printf(" a > 5 => %i\n", a > 5);
    printf("a >= b => %i\n", a >= b);
    printf(" a < b => %i\n", a < b);
    printf("a <= b => %i\n", a <= b);
    printf("    !a => %i\n", !a);
    printf("    !c => %i\n", !c);

    return 0;
}
```



a	==	5	=>	1
a	!=	5	=>	0
a	>	5	=>	0
a	>=	b	=>	0
a	<	b	=>	1
a	<=	b	=>	1
	!	a	=>	0
	!	c	=>	1

The *if*, *else if* and *else* Statements

```
int a = 10;

if (a > 0)
    printf("a is positive.\n");

else if (a < 0)
    printf("a is negative.\n");

else
    printf("a is null.\n");

if (a)
{
    printf("a is not null.\n");
    printf("The condition is TRUE.\n");
}

if (!a)
{
    printf("a is null.\n");
    printf("The condition is FALSE.\n");
}
```

a is positive.
a is not null.
The condition is TRUE.

```
if (condition)
{
    // ...
}

else if (condition)
{
    // ...
}

else
{
    // ...
}
```

The ***else*** and ***else if*** statements are optional.

The *for* Statement

```
for (init; condition; post)
{
    // ...
}
```

```
for (int n = 0; n < 3; n++)
    printf("n = %i\n", n);

short x;

for (x = -3; x < 4; x++)
{
    if (x < 0)
        printf("(%hi) * (%hi) = %hi\n", x, x, x*x);
    else
        printf("%hi * %hi = %hi\n", x, x, x*x);
}
```

```
n = 0
n = 1
n = 2
(-3) * (-3) = 9
(-2) * (-2) = 4
(-1) * (-1) = 1
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
```

The *while* Statement

```
while (condition)
{
    // ...
}
```

```
short x = -3;
while (x < 4)
{
    if (x < 0)
        printf("(%hi) * (%hi) = %hi\n", x, x, x*x);
    else
        printf("%hi * %hi = %hi\n", x, x, x*x);

    x++;
}
```

```
(-3) * (-3) = 9
(-2) * (-2) = 4
(-1) * (-1) = 1
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
```

The *do...while* Statement

```
do
{
    // ...
} while (condition);
```

```
short x = -3;

do
{
    if (x < 0)
        printf("(%hi) * (%hi) = %hi\n", x, x, x*x);
    else
        printf("%hi * %hi = %hi\n", x, x, x*x);

    x++;
} while (x < 4);
```

```
(-3) * (-3) = 9
(-2) * (-2) = 4
(-1) * (-1) = 1
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
```


The *break* and *continue* Statements

The *break* and *continue* statements can be used in loop bodies (e.g. *for*, *while*, *do...while*)

- *break*: Terminates the loop.
- *continue*: Goes to the next iteration.

The *switch...case* Statement

```
switch (value)
{
    case const_1:
        // ...
        break;

    case const_2:
        // ...
        break;

    // etc.

    default:
        // ...
}
```

```
int a = 10;
switch (a)
{
    case 0:
        printf("a is null.");
        break;

    case 100:
        printf("a is one hundred.\n");
        break;

    default:
        printf("a is not null.\n");
        printf("a is not one hundred.\n");
}
```

```
a is not null.
a is not one hundred.
```

Type Casting

```
int i = 35;
float f1, f2, f3, f4;

f1 = i;           // Implicit
f2 = (float)i;   // Explicit

printf(" i = %i\n", i);
printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("-----\n");

f1 = i / 2;
f2 = (float)(i / 2);
f3 = (float)i / 2;
f4 = i / (float)2;

printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("f3 = %f\n", f3);
printf("f4 = %f\n", f4);
printf("-----\n");

f1 = 325.53421;
i = f1;

printf("f1 = %f\n", f1);
printf(" i = %i\n", i);
```



i = 35

Type Casting

```
int i = 35;
float f1, f2, f3, f4;

f1 = i;           // Implicit
f2 = (float)i;   // Explicit

printf(" i = %i\n", i);
printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("-----\n");

f1 = i / 2;
f2 = (float)(i / 2);
f3 = (float)i / 2;
f4 = i / (float)2;

printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("f3 = %f\n", f3);
printf("f4 = %f\n", f4);
printf("-----\n");

f1 = 325.53421;
i = f1;

printf("f1 = %f\n", f1);
printf(" i = %i\n", i);
```



```
i = 35
f1 = 35.000000
```

Type Casting

```
int i = 35;
float f1, f2, f3, f4;

f1 = i;           // Implicit
f2 = (float)i;   // Explicit

printf(" i = %i\n", i);
printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("-----\n");

f1 = i / 2;
f2 = (float)(i / 2);
f3 = (float)i / 2;
f4 = i / (float)2;

printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("f3 = %f\n", f3);
printf("f4 = %f\n", f4);
printf("-----\n");

f1 = 325.53421;
i = f1;

printf("f1 = %f\n", f1);
printf(" i = %i\n", i);
```



```
i = 35
f1 = 35.000000
f2 = 35.000000
-----
```

Type Casting

```
int i = 35;
float f1, f2, f3, f4;

f1 = i;           // Implicit
f2 = (float)i;   // Explicit

printf(" i = %i\n", i);
printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("-----\n");

f1 = i / 2;
f2 = (float)(i / 2);
f3 = (float)i / 2;
f4 = i / (float)2;

printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("f3 = %f\n", f3);
printf("f4 = %f\n", f4);
printf("-----\n");

f1 = 325.53421;
i = f1;

printf("f1 = %f\n", f1);
printf(" i = %i\n", i);
```



```
 i = 35
f1 = 35.000000
f2 = 35.000000
-----
f1 = 17.000000
```

Type Casting

```
int i = 35;
float f1, f2, f3, f4;

f1 = i;           // Implicit
f2 = (float)i;   // Explicit

printf(" i = %i\n", i);
printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("-----\n");

f1 = i / 2;
f2 = (float)(i / 2);
f3 = (float)i / 2;
f4 = i / (float)2;

printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("f3 = %f\n", f3);
printf("f4 = %f\n", f4);
printf("-----\n");

f1 = 325.53421;
i = f1;

printf("f1 = %f\n", f1);
printf(" i = %i\n", i);
```



```
 i = 35
f1 = 35.000000
f2 = 35.000000
-----
f1 = 17.000000
f2 = 17.000000
```

Type Casting

```
int i = 35;
float f1, f2, f3, f4;

f1 = i;           // Implicit
f2 = (float)i;   // Explicit

printf(" i = %i\n", i);
printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("-----\n");

f1 = i / 2;
f2 = (float)(i / 2);
f3 = (float)i / 2;
f4 = i / (float)2;

printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("f3 = %f\n", f3);
printf("f4 = %f\n", f4);
printf("-----\n");

f1 = 325.53421;
i = f1;

printf("f1 = %f\n", f1);
printf(" i = %i\n", i);
```



```
 i = 35
f1 = 35.000000
f2 = 35.000000
-----
f1 = 17.000000
f2 = 17.000000
f3 = 17.500000
```


Type Casting

```
int i = 35;
float f1, f2, f3, f4;

f1 = i;           // Implicit
f2 = (float)i;   // Explicit

printf(" i = %i\n", i);
printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("-----\n");

f1 = i / 2;
f2 = (float)(i / 2);
f3 = (float)i / 2;
f4 = i / (float)2;

printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("f3 = %f\n", f3);
printf("f4 = %f\n", f4);
printf("-----\n");

f1 = 325.53421;
i = f1;

printf("f1 = %f\n", f1);
printf(" i = %i\n", i);
```



```
 i = 35
f1 = 35.000000
f2 = 35.000000
-----
f1 = 17.000000
f2 = 17.000000
f3 = 17.500000
f4 = 17.500000
-----
```

Type Casting

```
int i = 35;
float f1, f2, f3, f4;

f1 = i;           // Implicit
f2 = (float)i;   // Explicit

printf(" i = %i\n", i);
printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("-----\n");

f1 = i / 2;
f2 = (float)(i / 2);
f3 = (float)i / 2;
f4 = i / (float)2;

printf("f1 = %f\n", f1);
printf("f2 = %f\n", f2);
printf("f3 = %f\n", f3);
printf("f4 = %f\n", f4);
printf("-----\n");

f1 = 325.53421;
i = f1;

printf("f1 = %f\n", f1);
printf(" i = %i\n", i);
```



```
 i = 35
f1 = 35.000000
f2 = 35.000000
-----
f1 = 17.000000
f2 = 17.000000
f3 = 17.500000
f4 = 17.500000
-----
f1 = 325.534210
 i = 325
```

Overflow

```
c = i1; uc = i1;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i2; uc = i2;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i3; uc = i3;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = 129;
printf(" c = %hhi\n", c);
printf("-----\n");

uc = 250;
uc += 10;
printf("uc = %hhu\n", uc);
printf("-----\n");

uc = 0;
uc--;
printf("uc = %hhu\n", uc);
```

```
int i1 = 257, i2 = 128, i3 = -1;
char c;
unsigned char uc;
```



Overflow

```
c = i1; uc = i1;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i2; uc = i2;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i3; uc = i3;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = 129;
printf(" c = %hhi\n", c);
printf("-----\n");

uc = 250;
uc += 10;
printf("uc = %hhu\n", uc);
printf("-----\n");

uc = 0;
uc--;
printf("uc = %hhu\n", uc);
```

```
int i1 = 257, i2 = 128, i3 = -1;
char c;
unsigned char uc;
```

c = 1

Overflow

```
c = i1; uc = i1;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i2; uc = i2;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i3; uc = i3;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = 129;
printf(" c = %hhi\n", c);
printf("-----\n");

uc = 250;
uc += 10;
printf("uc = %hhu\n", uc);
printf("-----\n");

uc = 0;
uc--;
printf("uc = %hhu\n", uc);
```

```
int i1 = 257, i2 = 128, i3 = -1;
char c;
unsigned char uc;
```



```
c = 1
uc = 1
-----
```

Overflow

```
c = i1; uc = i1;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i2; uc = i2;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i3; uc = i3;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = 129;
printf(" c = %hhi\n", c);
printf("-----\n");

uc = 250;
uc += 10;
printf("uc = %hhu\n", uc);
printf("-----\n");

uc = 0;
uc--;
printf("uc = %hhu\n", uc);
```

```
int i1 = 257, i2 = 128, i3 = -1;
char c;
unsigned char uc;
```



```
c = 1
uc = 1
-----
c = -128
```

Overflow

```
c = i1; uc = i1;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i2; uc = i2;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i3; uc = i3;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = 129;
printf(" c = %hhi\n", c);
printf("-----\n");

uc = 250;
uc += 10;
printf("uc = %hhu\n", uc);
printf("-----\n");

uc = 0;
uc--;
printf("uc = %hhu\n", uc);
```

```
int i1 = 257, i2 = 128, i3 = -1;
char c;
unsigned char uc;
```



```
c = 1
uc = 1
-----
c = -128
uc = 128
-----
```

Overflow

```
c = i1; uc = i1;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i2; uc = i2;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i3; uc = i3;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = 129;
printf(" c = %hhi\n", c);
printf("-----\n");

uc = 250;
uc += 10;
printf("uc = %hhu\n", uc);
printf("-----\n");

uc = 0;
uc--;
printf("uc = %hhu\n", uc);
```

```
int i1 = 257, i2 = 128, i3 = -1;
char c;
unsigned char uc;
```



```
c = 1
uc = 1
-----
c = -128
uc = 128
-----
c = -1
```


Overflow

```
c = i1; uc = i1;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i2; uc = i2;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");


c = i3; uc = i3;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = 129;
printf(" c = %hhi\n", c);
printf("-----\n");

uc = 250;
uc += 10;
printf("uc = %hhu\n", uc);
printf("-----\n");

uc = 0;
uc--;
printf("uc = %hhu\n", uc);
```

```
int i1 = 257, i2 = 128, i3 = -1;
char c;
unsigned char uc;
```



```
c = 1
uc = 1
-----
c = -128
uc = 128
-----
c = -1
uc = 255
-----
```

Overflow

```
c = i1; uc = i1;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i2; uc = i2;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i3; uc = i3;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = 129;
printf(" c = %hhi\n", c);
printf("-----\n");

uc = 250;
uc += 10;
printf("uc = %hhu\n", uc);
printf("-----\n");

uc = 0;
uc--;
printf("uc = %hhu\n", uc);
```

```
int i1 = 257, i2 = 128, i3 = -1;
char c;
unsigned char uc;
```



```
c = 1
uc = 1
-----
c = -128
uc = 128
-----
c = -1
uc = 255
-----
c = -127
-----
```

Overflow

```
c = i1; uc = i1;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i2; uc = i2;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i3; uc = i3;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = 129;
printf(" c = %hhi\n", c);
printf("-----\n");

uc = 250;
uc += 10;
printf("uc = %hhu\n", uc);
printf("-----\n");

uc = 0;
uc--;
printf("uc = %hhu\n", uc);
```

```
int i1 = 257, i2 = 128, i3 = -1;
char c;
unsigned char uc;
```



```
c = 1
uc = 1
-----
c = -128
uc = 128
-----
c = -1
uc = 255
-----
c = -127
-----
uc = 4
-----
```

Overflow

```
c = i1; uc = i1;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i2; uc = i2;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = i3; uc = i3;
printf(" c = %hhi\n", c);
printf("uc = %hhu\n", uc);
printf("-----\n");

c = 129;
printf(" c = %hhi\n", c);
printf("-----\n");

uc = 250;
uc += 10;
printf("uc = %hhu\n", uc);
printf("-----\n");

uc = 0;
uc--;
printf("uc = %hhu\n", uc);
```

```
int i1 = 257, i2 = 128, i3 = -1;
char c;
unsigned char uc;
```



```
c = 1
uc = 1
-----
c = -128
uc = 128
-----
c = -1
uc = 255
-----
c = -127
-----
uc = 4
-----
uc = 255
```

Types Matter

```
int facto_int(int n)
{
    int r = 1;

    for (int i = 2; i <= n; i++)
        r *= i;

    return r;
}
```

```
unsigned int facto_uint(unsigned int n)
{
    unsigned int r = 1;

    for (unsigned int i = 2; i <= n; i++)
        r *= i;

    return r;
}
```

```
unsigned long facto_ulong(unsigned long n)
{
    unsigned long r = 1;

    for (unsigned long i = 2; i <= n; i++)
        r *= i;

    return r;
}
```

```
int main()
{
    printf("facto_int(20)    = %i\n", facto_int(20));
    printf("facto_uint(20)   = %u\n", facto_uint(20));
    printf("facto_ulong(20)  = %lu\n", facto_ulong(20));
}
```

```
facto_int(20)    = -2102132736
facto_uint(20)   = 2192834560
facto_ulong(20)  = 2432902008176640000
```

Readable ?

```
unsigned long f(unsigned long n)
{
    unsigned long r = 1;
    for (; n > 0; r *= n--);
    return r;
}
```