

Number Literals

Integers

0b11111111	binary	0B11111111	binary
0377	octal	255	decimal
0xff	hexadecimal	0xFF	hexadecimal

Real Numbers

88.0f / 88.1234567f

single precision float (f suffix)

88.0 / 88.123456789012345

double precision float (no f suffix)

Signage

42 / +42 positive -42 negative

Binary notation 0b... / 0B... is available on GCC and most but not all C compilers.

Variables

Declaring

int x;	A variable.
char x = 'C';	A variable & initialising it.
float x, y, z;	Multiple variables of the same type.

Variables (cont)

const int x = 88; A constant variable: can't assign to after declaration (compiler enforced.)

Naming

johnny5IsAlive; ✓ Alphanumeric, not a keyword, begins with a letter.

~~2001ASpaceOddysey;~~ ✗ Doesn't begin with a letter.

~~while;~~ ✗ Reserved keyword.

~~how-exciting+;~~ ✗ Non-alphanumeric.

~~i am a ve ry l ong var iab len ame ohm ygo shy esiam;~~ ✗

Longer than 31 characters (C89 & C90 only)

Constants are CAPITALISED. Function names usually take the form of a verb eg. plotRobotUprising().

Primitive Variable Types

**applicable but not limited to most ARM, AVR, x86 & x64 installations*

[class] [qualifier] [unsigned] type/void name;

by ascending arithmetic conversion

Integers

Type	Bytes	Value Range
------	-------	-------------



By **Ashlyn Black** (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 1 of 22.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Primitive Variable Types (cont)

char	1	unsigned OR signed
unsigned char	1	0 to 2^8-1
signed char	1	-2^7 to 2^7-1
int	2 / 4	unsigned OR signed
unsigned int	2 / 4	0 to $2^{16}-1$ OR $2^{31}-1$
signed int	2 / 4	-2^{15} to $2^{15}-1$ OR -2^{31} to $2^{32}-1$
short	2	unsigned OR signed
unsigned short	2	0 to $2^{16}-1$
signed short	2	-2^{15} to $2^{15}-1$
long	4 / 8	unsigned OR signed
unsigned long	4 / 8	0 to $2^{32}-1$ OR $2^{64}-1$
signed long	4 / 8	-2^{31} to $2^{31}-1$ OR -2^{63} to $2^{63}-1$

Primitive Variable Types (cont)

long long	8	unsigned OR signed
unsigned long long	8	0 to $2^{64}-1$
signed long long	8	-2^{63} to $2^{63}-1$

Floats

Type	Bytes	Value Range (Normalized)
float	4	$\pm 1.2 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$
double	8 / 4	$\pm 2.3 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$ OR alias to float for AVR.
long double	ARM: 8, AVR: 4, x86: 10, x64: 16	

Qualifiers

const type	Flags variable as read-only (compiler can optimise.)
------------	--



By **Ashlyn Black** (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 2 of 22.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Primitive Variable Types (cont)

`volatile type` Flags variable as unpredictable (compiler cannot optimise.)

Storage Classes

`register` Quick access required. May be stored in RAM **OR** a register. Maximum size is register size.

`static` Retained when out of scope. `static` global variables are confined to the scope of the compiled object file they were declared in.

`extern` Variable is declared by another file.

Typecasting

`(type) a` Returns `a` as data type.

Primitive Variable Types (cont)

`char x = 1, y = 2; float z = (float) x / y;`

Some types (denoted with **OR**) are architecture dependant.

There is no primitive boolean type, only zero (false, 0) and non-zero (true, usually 1.)

Extended Variable Types

`[class] [qualifier] type name;`

by ascending arithmetic conversion

From the `stdint.h` Library

Type	Bytes	Value Range
<code>int8_t</code>	1	-2^7 to 2^7-1
<code>uint8_t</code>	1	0 to 2^8-1
<code>int16_t</code>	2	-2^{15} to $2^{15}-1$
<code>uint16_t</code>	2	0 to $2^{16}-1$
<code>int32_t</code>	4	-2^{31} to $2^{31}-1$
<code>uint32_t</code>	4	0 to $2^{32}-1$
<code>int64_t</code>	8	-2^{63} to $2^{63}-1$
<code>uint64_t</code>	8	0 to $2^{64}-1$

From the `stdbool.h` Library

Type	Bytes	Value Range
------	-------	-------------



By **Ashlyn Black** (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.

Last updated 12th May, 2016.

Page 3 of 22.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

Extended Variable Types (cont)

`bool` `1` `true / false` or `0 / 1`

The `stdint.h` library was introduced in C99 to give integer types architecture-independent lengths.

Structures

Defining

```
struct strctName{ type x; type y; };
```

```
struct item{ struct item *next; };
```

Declaring

```
struct strctName varName;
```

```
struct strctName *ptrName;
```

```
struct strctName{ type a; type b; } varName;
```

Structures (cont)

`struct strctName varName = { a, b };` A variable `varName` of structure type `strctName` and initialising its members.

Accessing

A structure type `strctName` with two members, `x` and `y`. *Note trailing semicolon*

Bit Fields

with a recursive structure pointer inside. Useful for linked lists.

```
struct{char a:4, b:4} x;
```

Member `x` of structure `varName`.

Value of structure pointer `ptrName` member `x`.

Declares `x` with two members `a` and `b`, both four bits in size (0 to 15.)

Array members can't be assigned bit fields.

Type Definitions

Defining

```
typedef unsigned short uint16;
```

```
typedef struct strctName{int a, b;}newType;
```

A variable `varName` as structure type `strctName`.

A structure `strctName` type pointer, `ptrName`.

Shorthand for defining `strctName` and declaring `varName` as that structure type.

Abbreviating a longer type name to `uint16`.

Creating `newType` from a structure.



Type Definitions (cont)

```
typedef enum typeName{false, true}bool;
```

Creating an enumerated bool type.

Declaring

```
uint16 x = 65535;
```

Variable `x` as type `uint16`.

```
newType y = {0, 0};
```

Structure `y` as type `newType`.

Unions

Defining

```
union uName{int x; char y[8];}
```

A union type `uName` with two members, `x` & `y`. Size is same as biggest member size.

Declaring

```
union uN vName;
```

A variable `vName` as union type `uN`.

Accessing

```
vName.y[int]
```

Members cannot store values concurrently. Setting `y` will corrupt `x`.

Unions are used for storing multiple data types in the same area of memory.

Enumeration

Defining

```
enum bool { false, true };
```

A custom data type `bool` with two possible states: `false` or `true`.

Declaring

```
enum bool varName;
```

A variable `varName` of data type `bool`.

Assigning

```
varName = true;
```

Variable `varName` can only be assigned values of either `false` or `true`.

Evaluating

```
if(varName == false)
```

Testing the value of `varName`.

Pointers

Declaring

```
type *x;
```

Pointers have a data type like normal variables.

```
void *v;
```

They can also have an incomplete type. Operators other than assignment cannot be applied as the length of the type is unknown.



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
Last updated 12th May, 2016.
Page 5 of 22.

Sponsored by [Readable.com](https://readable.com)
Measure your website readability!
<https://readable.com>

Pointers (cont)

`struct type *y;` A data structure pointer.

`type z[];` An array/string name can be used as a pointer to the first array element.

Accessing

`x` A memory address.

`*x` Value stored at that address.

`y->a` Value stored in structure pointer `y` member `a`.

`&varName` Memory address of normal variable `varName`.

`*(type *)v` Dereferencing a void pointer as a type pointer.

A pointer is a variable that holds a memory location.

Arrays

Declaring

`type name[int];` You set array length.

`type name[int] = {x, y, z};` You set array length and initialise elements.

Arrays (cont)

`type name[int] = {x};` You set array length and initialise all elements to `x`.

`type name[] = {x, y, z};` Compiler sets array length based on initial elements.

Size cannot be changed after declaration.

Dimensions

`name[int]` One dimension array.

`name[int][int]` Two dimensional array.

Accessing

`name[int]` Value of element `int` in array `name`.

`*(name + int)` Same as `name[int]`.

Elements are contiguously numbered ascending from 0.

`&name[int]` Memory address of element `int` in array `name`.

`name + int` Same as `&name[int]`.

Elements are stored in contiguous memory.

Measuring

`sizeof(array) / sizeof(arrayType)` Returns length of array. (*Unsafe*)



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 6 of 22.

Sponsored by [Readable.com](https://readable.com)
 Measure your website readability!
<https://readable.com>

Arrays (cont)

`sizeof(array) / sizeof(array[0])` Returns length of array. (Safe)

Strings

'A' character Single quotes.

"AB" string Double quotes.

`\0` Null terminator.

Strings are char arrays.

```
char name[4] = "Ash";
```

is equivalent to

```
char name[4] = {'A', 's', 'h', '\0'};
```

```
int i; for(i = 0; name[i]; i++){
```

`\0` evaluates as false.

Strings must include a char element for `\0`.

Escape Characters

<code>\a</code>	alarm (bell/beep)	<code>\b</code>	backspace
<code>\f</code>	formfeed	<code>\n</code>	newline
<code>\r</code>	carriage return	<code>\t</code>	horizontal tab
<code>\v</code>	vertical tab	<code>\\</code>	backslash
<code>\'</code>	single quote	<code>\"</code>	double quote

Escape Characters (cont)

`\?` question mark

`\nnn` Any octal ANSI character code.

`\xhh` Any hexadecimal ANSI character code.

Functions

Declaring

```
type/void funcName([args...]){ [return var;] }
```

*Function names follow the same restrictions as variable names but must **also** be unique.*

`type/void` Return value type (void if none.)

`funcName()` Function name and argument parenthesis.

`args...` Argument types & names (void if none.)

`{ }` Function content delimiters.



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 7 of 22.

Sponsored by [Readable.com](https://readable.com)
 Measure your website readability!
<https://readable.com>

Functions (cont)

`return var;` Value to return to function call origin. Skip for `void` type functions. Functions exit immediately after a `return`.

By Value vs By Pointer

`void f(type x); f(y);` Passing variable `y` to function `f` argument `x` (by value.)

`void f(type *x); f(array);` Passing an array/string to function `f` argument `x` (by pointer.)

`void f(type *x); f(structure);` Passing a structure to function `f` argument `x` (by pointer.)

`void f(type *x); f(&y);` Passing variable `y` to function `f` argument `x` (by pointer.)

`type f(){ return x; }` Returning by value.

`type f(){ type x; return &x; }` Returning a variable by pointer.

Functions (cont)

`type f(){ static type x[]; return &x; }` Returning an array/string/string by pointer. The `static` qualifier is necessary otherwise `x` won't exist after the function exits.

Passing by pointer allows you to change the originating variable within the function.

Scope

```
int f(){ int i = 0; } i++;
```

`i` is declared inside `f()`, it doesn't exist outside that function.

Prototyping

```
type funcName(args...);
```

Place before declaring or referencing respective function (usually before the function definition.)

`type funcName([args...])` Same type, name and `args...` as the function definition.

`;` Semicolon instance of function delimiter.



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
Last updated 12th May, 2016.
Page 8 of 22.

Sponsored by [Readable.com](https://readable.com)
Measure your website readability!
<https://readable.com>

main()

```
int main(int argc, char *argv[]){return int;}
```

Anatomy

int main	Program entry point.
int argc	# of command line arguments.
char *argv[]	Command line arguments in an array of strings. #1 is always the program filename.
return int;	Exit status (integer) returned to the OS upon program exit.

Command Line Arguments

app two 3	Three arguments, " app ", " two " and " 3".
app "two 3"	Two arguments, " app " and "two 3".

main is the first function called when the program executes.

Conditional (Branching)

if, else if, else

if(a) b;	Evaluates b if a is true.
if(a){ b; c; }	Evaluates b and c if a is true.
if(a){ b; }else{ c; }	Evaluates b if a is true, c otherwise.

Conditional (Branching) (cont)

```
if(a){ b; }else if(c){ d; }else{ e; }
```

switch, case, break

```
switch(a){ case b: c; }
```

```
switch(a){ default: b; }
```

```
switch(a){ case b: case c: d; }
```

```
switch(a){ case b: c; case d: e; default: f; }
```

```
switch(a){ case b: c; break; case d: e; break; default: f; }
```



Iterative (Looping)

while

```
int x = 0; while(x < 10){ x += 2; }
```

Loop skipped if test condition initially false.

int x = 0; Declare and initialise integer x.

while() Loop keyword and condition parenthesis.

x < 10 Test condition.

{ } Loop delimiters.

x += 2; Loop contents.

do while

```
char c = 'A'; do { c++; } while(c != 'Z');
```

Always runs through loop at least once.

char c = 'A'; Declare and initialise character c.

do Loop keyword.

{ } Loop delimiters.

c++; Loop contents.

while(); Loop keyword and condition parenthesis. *Note semicolon.*

c != 'Z' Test condition.

for

```
int i; for(i = 0; n[i] != '\0'; i++){ } (C89)
```

Iterative (Looping) (cont)

OR

```
for(int i = 0; n[i] != '\0'; i++){ } (C99+)
```

Compact increment/decrement based loop.

int i; Declares integer i.

for() Loop keyword.

i = 0; Initialises integer i. *Semicolon.*

n[i] != '\0'; Test condition. *Semicolon.*

i++ Increments i. *No semicolon.*

{ } Loop delimiters.

continue

```
int i=0; while(i<10){ i++; continue; i--; }
```

Skips rest of loop contents and restarts at the beginning of the loop.

break

```
int i=0; while(1){ if(x==10){ break; } i++; }
```

Skips rest of loop contents and exits loop.



By **Ashlyn Black** (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 10 of 22.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Console Input/Output

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

Characters

<code>getchar()</code>	Returns a single character's ANSI code from the input stream buffer as an <i>integer</i> . (<i>safe</i>)
<code>putchar(int)</code>	Prints a single character from an ANSI code <i>integer</i> to the output stream buffer.

Strings

<code>gets(strName)</code>	Reads a line from the input stream into a string variable. (<i>Unsafe, removed in C11.</i>)
----------------------------	---

Alternative

<code>fgets(strName, length, stdin);</code>	Reads a line from the input stream into a string variable. (<i>Safe</i>)
<code>puts("string")</code>	Prints a string to the output stream.

Formatted Data

Console Input/Output (cont)

<code>scanf("%d", &x)</code>	Read value/s (type defined by format string) into variable/s (type must match) from the input stream. Stops reading at the first whitespace. & <i>prefix not required for arrays (including strings.) (unsafe)</i>
<code>printf ("I love %c %d!", 'C', 99)</code>	Prints data (formats defined by the format string) as a string to the output stream.

Alternative

C By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 11 of 22.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Console Input/Output (cont)

`fgets(strName, length, stdin); sscanf(strName, "%d", &x);`

`fgets(strName, length, stdin);` **Uses** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

`scanf` to read the input. **Modes** `scanf` to read the input.

The stream buffers must be flushed to reflect changes. String terminator characters can flush the output while newline characters can flush the input.

Safe functions are those that let you specify the length of the input. *Unsafe* functions do not, and carry the risk of memory overflow.

File Input/Output

`#include <stdio.h>`

Opening

`FILE *fptr = fopen(filename, mode);`

`FILE *fptr` Declares `fptr` as a `FILE` type pointer (stores stream location instead of memory location.)

`fopen()` Returns a stream location pointer if successful, 0 otherwise.

File Input/Output (cont)

`String containing file's directory path & name.`

`String specifying the file access mode.`

`Read existing text/binary file.`

`Write new/over existing text/binary file.`

`Write new/append to existing text/binary file.`

`Read and write existing text/binary file.`

`Read and write new/over existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`

`Read and write new/append to existing text/binary file.`



By **Ashlyn Black** (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 12 of 22.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

File Input/Output (cont)

`fclose(fptr);` Flushes buffers and closes stream. Returns 0 if successful, EOF otherwise.

Random Access

`ftell(fptr)` Return current file position as a long integer.

`fseek(fptr, offset, origin);` Sets current file position. Returns *false* is successful, *true* otherwise. The *offset* is a long integer type.

Origins

`SEEK_SET` Beginning of file.

`SEEK_CUR` Current position in file.

`SEEK_END` End of file.

Utilities

`feof(fptr)` Tests end-of-file indicator.

`rename(strOldName, strNewName)` Renames a file.

`remove(strName)` Deletes a file.

Characters

File Input/Output (cont)

`fgetc(fptr)` Returns character read or EOF if unsuccessful. (*safe*)

`fputc(int c, fptr)` Returns character written or EOF if unsuccessful.

Strings

`fgets(char *s, int n, fptr)` Reads *n-1* characters from file *fptr* into string *s*. Stops at EOF and *\n*. (*safe*)

`fputs(char *s, fptr)` Writes string *s* to file *fptr*. Returns non-negative on success, EOF otherwise.

Formatted Data

`fscanf(fptr, format, [...])` Same as `scanf` with additional file pointer parameter. (*unsafe*)

`fprintf(fptr, format, [...])` Same as `printf` with additional file pointer parameter.

Alternative



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
Last updated 12th May, 2016.
Page 13 of 22.

Sponsored by [Readable.com](https://readable.com)
Measure your website readability!
<https://readable.com>

File Input/Output (cont)

```
fgets(strName, length, fptr); sscanf(strName, "%d", &x);
```

Placeholder Types (f/printf And f/scanf) (cont)

%u	Uses fge	42	Unsigned decimal integer.
%o	ts to limit the input length, then uses	52	Unsigned octal integer.
%x or %X	sscanf to	2a or 2A	Unsigned hexadecimal integer.
%f or %F	read the resulting	1.21	Signed decimal float.
%e or %E	string in place of s	1.21e+9 or 1.21E+9	Signed decimal w/ scientific notation.
%g or %G	canf. (safe)	0x1.207c8ap+30 or 0X1.207C8AP+30	Shortest representation of %f/%F or %e/%E.
%a or %A			Signed hexadecimal float.
%c	Reads a n	a	A character.
%s	umber of	A String.	A character string.
%p	elements		A pointer.
%%	from fptr	%	A percent character.

Binary

```
fread(void *ptr, sizeof(element), number, fptr)
```

```
fwrite(void *ptr, sizeof(element), number, fptr)
```

Safe functions are those that let you specify the length of the input.
Unsafe functions do not, and carry the risk of memory overflow.

Placeholder Types (f/printf And f/scanf)

```
printf("%d%d...", arg1, arg2...);
```

Type	Example	Description
%d or %i	-42	Signed decimal integer.



By **Ashlyn Black** (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
Last updated 12th May, 2016.
Page 14 of 22.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Placeholder Types (f/printf And f/scanf) (cont)

`%n` No output, saves # of characters printed so far. Respective printf argument must be an integer pointer.

The pointer format is architecture and implementation dependant.

Placeholder Formatting (f/printf And f/scanf)

`%[Flags][Width][.Precision][Length]Type`

Flags

- Left justify instead of default right justify.
+ Sign for both positive numbers and negative.
Precede with 0, 0x or 0X for %o, %x and %X tokens.
space Left pad with spaces.
0 Left pad with zeroes.

Width

integer Minimum number of characters to print: invokes padding if necessary. Will not truncate.
* Width specified by a preceding argument in printf.

Placeholder Formatting (f/printf And f/scanf) (cont)

Precision

.integer Minimum # of digits to print for %d, %i, %o, %u, %x, %X. Left pads with zeroes. Will not truncate. Skips values of 0.

Minimum # of digits to print after decimal point for %a, %A, %e, %E, %f, %F (default of 6.)

Minimum # of significant digits to print for %g & %G.

Maximum # of characters to print from %s (a string.)

. If no integer is given, default of 0.

. * Precision specified by a preceding argument in printf.

Length

hh Display a char as int.
h Display a short as int.
l Display a long integer.
ll Display a long long integer.
L Display a long double float.
z Display a size_t integer.



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
Last updated 12th May, 2016.
Page 15 of 22.

Sponsored by [Readable.com](https://readable.com)
Measure your website readability!
<https://readable.com>

Placeholder Formatting (f/printf And f/scanf) (cont)

j	Display a <code>intmax_t</code> integer.
t	Display a <code>ptrdiff_t</code> integer.

Preprocessor Directives

<code>#include <inbuilt.h></code>	Replaces line with contents of a standard C header file.
<code>#include "custom.h"</code>	Replaces line with contents of a custom header file. <i>Note dir path prefix & quotations.</i>
<code>#define NAME value</code>	Replaces all occurrences of NAME with value.

Comments

```
// We're single-line comments!
// Nothing compiled after // on these lines.
/* I'm a multi-line comment!
   Nothing compiled between
   these delimi ters. */
```

C Reserved Keywords

<code>_Alignas</code>	<code>break</code>	<code>float</code>	<code>signed</code>
<code>_Alignof</code>	<code>case</code>	<code>for</code>	<code>sizeof</code>
<code>_Atomic</code>	<code>char</code>	<code>goto</code>	<code>static</code>
<code>_Bool</code>	<code>const</code>	<code>if</code>	<code>struct</code>
<code>_Complex</code>	<code>continue</code>	<code>inline</code>	<code>switch</code>
<code>_Generic</code>	<code>default</code>	<code>int</code>	<code>typedef</code>
<code>_Imaginary</code>	<code>do</code>	<code>long</code>	<code>union</code>
<code>_Noreturn</code>	<code>double</code>	<code>register</code>	<code>unsigned</code>
<code>_Static_assert</code>	<code>else</code>	<code>restrict</code>	<code>void</code>
<code>_Thread_local</code>	<code>enum</code>	<code>return</code>	<code>volatile</code>
<code>auto</code>	<code>extern</code>	<code>short</code>	<code>while</code>
<code>_A-Z...</code>	<code>__...</code>		

C / POSIX Reserved Keywords

<code>E[0-9]...</code>	<code>E[A-Z]...</code>	<code>is[a-z]...</code>	<code>to[a-z]...</code>
<code>LC_[A-Z]...</code>	<code>SIG[A-Z]...</code>	<code>SIG_[A-Z]...</code>	<code>str[a-z]...</code>
<code>mem[a-z]...</code>	<code>wcs[a-z]...</code>	<code>..._t</code>	

GNU Reserved Names



By **Ashlyn Black** (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 16 of 22.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Header Reserved Keywords

Name	Reserved By Library
d_...	dirent.h
l_...	fcntl.h
F_...	fcntl.h
O_...	fcntl.h
S_...	fcntl.h
gr_...	grp.h
..._MAX	limits.h
pw_...	pwd.h
sa_...	signal.h
SA_...	signal.h
st_...	sys/stat.h
S_...	sys/stat.h
tms_...	sys/times.h
c_...	termios.h
V...	termios.h
I...	termios.h
O...	termios.h
TC...	termios.h
B[0-9]...	termios.h

Header Reserved Keywords (cont)

GNU Reserved Names

Heap Space

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

Allocating

```
malloc();
```

```
type *x; x = malloc(sizeof(type));
```

```
type *y; y = malloc(sizeof(type) * length );
```

```
struct type *z; z = malloc(sizeof(struct type));
```

Deallocating

```
free(ptrName);
```

Reallocating



By **Ashlyn Black** (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 17 of 22.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Heap Space (cont)

`realloc(ptrName, size);` Attempts to resize the memory block assigned to `ptrName`.

The memory addresses you see are from virtual memory the operating system assigns to the program; they are not physical addresses.

Referencing memory that isn't assigned to the program will produce an OS segmentation fault.

The Standard Library

`#include <stdlib.h>`

Randomicity

`rand()` Returns a (predictable) random integer between 0 and `RAND_MAX` based on the randomiser seed.

`RAND_MAX` The maximum value `rand()` can generate.

`srand(unsigned integer);` Seeds the randomiser with a positive integer.

`(unsigned) time(NULL)` Returns the computer's tick-tock value. Updates every second.

The Standard Library (cont)

Sorting

`qsort(array, length, sizeof(type),`

`qsort()` Sort using the QuickSort algorithm.

`array` Array/string name.

`length` Length of the array/string.

`sizeof(type)` Byte size of each element.

`compFunc` Comparison function name.

compFunc

`int compFunc( const void *a, const void b* ){ return`

`int compFunc()` Function name unimportant.

`const void *a, const void *b` Argument names unimportant.

`return( *(int *)a - *(int *)b );` Negative result swaps `b` for `a`, result of 0 doesn't swap.

C's inbuilt randomiser is cryptographically insecure: DO NOT use it for security applications.



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
Last updated 12th May, 2016.
Page 18 of 22.

Sponsored by [Readable.com](https://readable.com)
Measure your website readability!
<https://readable.com>

The Character Type Library

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

<code>tolower(char)</code>	Lowercase char.
<code>toupper(char)</code>	Uppercase char.
<code>isalpha(char)</code>	True if char is a letter of the alphabet, false otherwise.
<code>islower(char)</code>	True if char is a lowercase letter of the alphabet, false otherwise.
<code>isupper(char)</code>	True if char is an uppercase letter of the alphabet, false otherwise.
<code>isnumber(char)</code>	True if char is numerical (0 to 9) and false otherwise.
<code>isblank</code>	True if char is a whitespace character (' ', '\t', '\n') and false otherwise.

The String Library

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

<code>strlen(a)</code>	Returns # of char in string a as an integer. Excludes \0. (<i>unsafe</i>)
<code>strcpy(a, b)</code>	Copies strings. Copies string b over string a up to and including \0. (<i>unsafe</i>)
<code>strcat(a, b)</code>	Concatenates strings. Copies string b over string a up to and including \0, starting at the position of \0 in string a. (<i>unsafe</i>)
<code>strcmp(a, b)</code>	Compares strings. Returns <i>false</i> if string a equals string b, <i>true</i> otherwise. Ignores characters after \0. (<i>unsafe</i>)
<code>strstr(a, b)</code>	Searches for string b inside string a. Returns a pointer if successful, NULL otherwise. (<i>unsafe</i>)

Alternatives

<code>strncpy(a, b, n)</code>	Copies strings. Copies n characters from string b over string a up to and including \0. (<i>safe</i>)
-------------------------------	---



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 19 of 22.

Sponsored by [Readable.com](https://readable.com)
 Measure your website readability!
<https://readable.com>

The String Library (cont)

<code>strncat(a, b, n)</code>	Concatenates strings. Copies <code>n</code> characters from string <code>b</code> over string <code>a</code> up to and including <code>\0</code> , starting at the position of <code>\0</code> in string <code>a</code> . (<i>safe</i>)
<code>strncmp(a, b, n)</code>	Compares first <code>n</code> characters of two strings. Returns <i>false</i> if string <code>a</code> equals string <code>b</code> , <i>true</i> otherwise. Ignores characters after <code>\0</code> . (<i>safe</i>)

Safe functions are those that let you specify the length of the input.

Unsafe functions do not, and carry the risk of memory overflow.

The Time Library

```
#include <time.h>
```

Variable Types

<code>time_t</code>	Stores the calendar time.
<code>struct tm *x;</code>	Stores a time & date breakdown.
<i>tm structure members:</i>	
<code>int tm_sec</code>	Seconds, 0 to 59.
<code>int tm_min</code>	Minutes, 0 to 59.
<code>int tm_hour</code>	Hours, 0 to 23.
<code>int tm_mday</code>	Day of the month, 1 to 31.

The Time Library (cont)

<code>int tm_mon</code>	Month, 0 to 11.
<code>int tm_year</code>	Years since 1900.
<code>int tm_wday</code>	Day of the week, 0 to 6.
<code>int tm_yday</code>	Day of the year, 0 to 365.
<code>int tm_isdst</code>	Daylight saving time.

Functions

<code>time(NULL)</code>	Returns unix epoch time (seconds since 1/Jan/1970.)
<code>time(&time_t);</code>	Stores the current time in a <code>time_t</code> variable.
<code>ctime(&time_t)</code>	Returns a <code>time_t</code> variable as a string.
<code>x = localtime(&time_t);</code>	Breaks <code>time_t</code> down into struct <code>tm</code> members.

Unary Operators

by descending evaluation precedence

<code>+a</code>	Sum of 0 (zero) and <code>a</code> . (<code>0 + a</code>)
<code>-a</code>	Difference of 0 (zero) and <code>a</code> . (<code>0 - a</code>)
<code>!a</code>	Complement (logical NOT) of <code>a</code> . (<code>~a</code>)



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 20 of 22.

Sponsored by [Readable.com](https://readable.com)
 Measure your website readability!
<https://readable.com>

Unary Operators (cont)

<code>~a</code>	Binary ones complement (bitwise NOT) of <code>a</code> . (<code>~a</code>)
<code>++a</code>	Increment of <code>a</code> by 1. (<code>a = a + 1</code>)
<code>--a</code>	Decrement of <code>a</code> by 1. (<code>a = a - 1</code>)
<code>a++</code>	Returns <code>a</code> then increments <code>a</code> by 1. (<code>a = a + 1</code>)
<code>a--</code>	Returns <code>a</code> then decrements <code>a</code> by 1. (<code>a = a - 1</code>)
<code>(type)a</code>	Typecasts <code>a</code> as <code>type</code> .
<code>&a;</code>	Memory location of <code>a</code> .
<code>sizeof(a)</code>	Memory size of <code>a</code> (or <code>type</code>) in bytes.

Binary Operators

by descending evaluation precedence

<code>a * b;</code>	Product of <code>a</code> and <code>b</code> . (<code>a × b</code>)
<code>a / b;</code>	Quotient of dividend <code>a</code> and divisor <code>b</code> . Ensure divisor is non-zero. (<code>a ÷ b</code>)
<code>a % b;</code>	Remainder of <i>integers</i> dividend <code>a</code> and divisor <code>b</code> .
<code>a + b;</code>	Sum of <code>a</code> and <code>b</code> .
<code>a - b;</code>	Difference of <code>a</code> and <code>b</code> .

Binary Operators (cont)

<code>a << b;</code>	Left bitwise shift of <code>a</code> by <code>b</code> places. (<code>a × 2^b</code>)
<code>a >> b;</code>	Right bitwise shift of <code>a</code> by <code>b</code> places. (<code>a × 2^{-b}</code>)
<code>a < b;</code>	Less than. True if <code>a</code> is less than <code>b</code> and false otherwise.
<code>a <= b;</code>	Less than or equal to. True if <code>a</code> is less than or equal to <code>b</code> and false otherwise. (<code>a ≤ b</code>)
<code>a > b;</code>	Greater than. True if <code>a</code> is greater than <code>b</code> and false otherwise.
<code>a >= b;</code>	Greater than or equal to. True if <code>a</code> is greater than or equal to <code>b</code> and false otherwise. (<code>a ≥ b</code>)
<code>a == b;</code>	Equality. True if <code>a</code> is equal to <code>b</code> and false otherwise. (<code>a ↔ b</code>)
<code>a != b;</code>	Inequality. True if <code>a</code> is not equal to <code>b</code> and false otherwise. (<code>a ≠ b</code>)
<code>a & b;</code>	Bitwise AND of <code>a</code> and <code>b</code> . (<code>a ∧ b</code>)
<code>a ^ b;</code>	Bitwise exclusive-OR of <code>a</code> and <code>b</code> . (<code>a ⊕ b</code>)



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
 Last updated 12th May, 2016.
 Page 21 of 22.

Sponsored by [Readable.com](https://readable.com)
 Measure your website readability!
<https://readable.com>

Binary Operators (cont)

<code>a b;</code>	Bitwise inclusive-OR of <code>a</code> and <code>b</code> . (<code>a b</code>)
<code>a && b;</code>	Logical AND. True if both <code>a</code> and <code>b</code> are non-zero. (Logical AND) (<code>a & b</code>)
<code>a b;</code>	Logical OR. True if either <code>a</code> or <code>b</code> are non-zero. (Logical OR) (<code>a b</code>)

Ternary & Assignment Operators

by descending evaluation precedence

`x ? a : b;` Evaluates `a` if `x` evaluates as true or `b` otherwise. (`(if(x){ a; } else { b; })`)

`x = a;` Assigns value of `a` to `x`.

`a *= b;` Assigns product of `a` and `b` to `a`. (`a = a * b`)

`a /= b;` Assigns quotient of dividend `a` and divisor `b` to `a`. (`a = a / b`)

`a %= b;` Assigns remainder of *integers* dividend `a` and divisor `b` to `a`. (`a = a % b`)

`a += b;` Assigns sum of `a` and `b` to `a`. (`a = a + b`)

Ternary & Assignment Operators (cont)

`a -= b;` Assigns difference of `a` and `b` to `a`. (`a = a - b`)

`a <<= b;` Assigns left bitwise shift of `a` by `b` places to `a`. (`a = a << 2b`)

`a >>= b;` Assigns right bitwise shift of `a` by `b` places to `a`. (`a = a >> 2b`)

`a &= b;` Assigns bitwise AND of `a` and `b` to `a`. (`a = a & b`)

`a ^= b;` Assigns bitwise exclusive-OR of `a` and `b` to `a`. (`a = a ^ b`)

`a |= b;` Assigns bitwise inclusive-OR of `a` and `b` to `a`. (`a = a | b`)

C Cheatsheet by Ashlyn Black

ashlynblack.com



By Ashlyn Black (Ashlyn Black)
cheatography.com/ashlyn-black/

ashlynblack.com

Published 28th January, 2015.
Last updated 12th May, 2016.
Page 22 of 22.

Sponsored by [Readable.com](https://readable.com)
Measure your website readability!
<https://readable.com>