C Programming Cheatsheet

String Format Specifiers

%[flags][width][.precision][length]specifier

`%c`	char
`%hhd||%hhi`	signed char
`%hhu`	unsigned char
`%hhn`	signed char*
`%lc`	wint_t
`%ls`	wchar_t*
`%s`	string
`%d || %i`	signed int
`%u`	unsigned int
`%hi`	short int
`%hu`	unsigned short int
`%hn`	short int*
`%l`	signed long int
`%ln`	long int*
`%ll`	signed long long int
`%lln`	long long int*
`%llu`	unsigned long long int
`%f || %F`	float or double (%F is uppercase)
`%Lf || %Le`	long double
`%e || %E`	scientific notation (mantissa/exponent)
`%g || %G`	shortest representation of %e||%E
`%o`	octal unsigned int
`%x`	lowercase hex unsigned int
`%X`	uppercase hex unsigned int
`%a || %A`	hexadecimal float-point
`%ji`	intmax_t
`%ju`	uintmax_t
`%jn`	intmax_t*
`%zi || %zu`	size_t || ssize_t
`%zn`	size_t*
`%ti || %tu`	ptrdiff_t
`%tn`	ptrdiff_t*
`%p`	pointer address
`%n`	NULL
`%%`	literal %
Width and Precision	
`%.3f`	float precision of 3 (like 3.141)
`%4d`	4 digit wide int (like 2015)
`%2.2f`	2 digits wide and 2 precise (19.95)
Flags	
`-`	Left-justify
`+`	Right-justify
`SPACE`	Blank space
`#`	Preceded hex & octal with "0x" || "0"
`0`	Left-pad with zeros

Integer from variable - printf("%d", num);
Save integer to variable - scanf("%d", &num);
Save string to variable - scanf("%s", str_var);

Character Escapes

- **\0** - NULL
- **\b** - backspace
- **\f** - form feed (new page)
- **\n** - newline
- **\r** - carriage return
- **\t** - tab
- **\v** - vertical tab

Arithmetic Operators

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`%`	Modulus/Remainder
`++`	Increment by 1
`--`	Decrement by 1
`++>`	Pre-increment and compare
`-->`	Pre-decrement and compare

Equality Operators

`==`	Equal to
`!=`	Not equal to
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to

Logical Operators

Operand	Meaning	Example
`&&`	And	(x && y)
`||`	Or	(x || y)
`!`	Not	!(x < y)

Bitwise Operators

`&`	AND
`|`	OR
`^`	Exclusive OR (XOR)
`~`	Ones Complement (NOT)
`<<`	Left-shift
`>>`	Right-shift

Assignment Operators

Operand	Meaning	Equivalent
`=`	Assign	None
`+=`	Add	`X = X + Y`
`-=`	Subtract	`X = X - Y`
`*=`	Multiply	`X = X * Y`
`/=`	Divide	`X = X / Y`
`%=`	Modulus	`X = X % Y`
`<<=`	Left-shift	`X = X << Y`
`>>=`	Right-shift	`X = X >> Y`
`&=`	AND	`X = X & Y`
`|=`	OR	`X = X | Y`
`^=`	XOR	`X = X ^ Y`

Constructs

Do-While Loop

```
i=0;
do { printf("%d\n", i); ++i;}
while(i<10);
```

For Loop

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

While Loop

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

If, else if, else

```
if (0 == 1) {
    register signed int ZERO = 0;
} else if (8 <= 4) {
    const float PIf = 3.14F;
} else {
    static char SYM[3] = "π\0";
}
```

Macros if

```
#ifdef __linux__
#   include "custom_header.h"
#   include <system_header.h>
#endif
```

Switch-case

```
switch (INPUT) {
    case 0: break;
    default: break;
}
```

Ternary Operator

```
int out = (input == 7 ? 5 : 3);
```

Goto

```
label:
goto label;
```

Define Datatype

```
typedef struct { int x, y; } point_t;
typedef union __number {
    int i; double d;
} number_t;
```

Define Enum

```
enum cardsuit {
    CLUBS = 0,
    DIAMONDS, HEARTS, SPADES
};
```

Variable Aliases and Constants

```
const double PI = 3.14159;
const double *ARCHIMEDES_NUM = &PI;
extern const double PI;  // In Header
char PI_SYM[3] = "π\0";  // Unicode
char PI_UTF8[] = u8"π\0";
char16_t PI_UTF16[] = u"π\0";
char32_t PI_UTF32[] = U"π\0";
```

Arrays

```
double num[2] = { 3.14, 5.0 };
unsigned int LargeArray[2][4] = {
    { 0, 1, 2, 3 }, { 4, 5, 6, 7 } };
char words[2][] = { "BSD", "AIX" };
```

Order of Operations

`() [] -> . ::`	`! ~ - + * & ++ --`
`* / %`	`+ -`
`<< >>`	`< <= > >=`
`!= ==`	`& (Bitwise)`
`^ (Bitwise)`	`| (Bitwise)`
`&& || (Logical)`	`Ternary operator`
`Assignment`	`Comma Operator`

C Programming Cheatsheet

Datatypes

NULL	void
_Bool	bool - <stdbool.h>
char16_t - <uchar.h>	char32_t - <uchar.h>
char	double
enum	EOF - <stdio.h>
FILE - <stdio.h>	fpos_t - <stdio.h>
float	imaxdiv_t - <inttypes.h>
int	long
long double	long int
long long	long long int
nullptr_t - <stddef.h>	ptrdiff_t - <stddef.h>
sig_atomic_t - <signal,h>	short
short char	short int
size_t - <stddef.h>	ssize_t - <stddef.h>
struct	union
wctrans_t - <wchar.h>	wctype_t - <wctype.h>
wchar_t - <wchar.h>	__ibm128
WEOF - <wchar.h>	wint_t - <wchar.h>
signed	unsigned
signed char	unsigned char
signed int	unsigned int
signed long	unsigned long
signed long int	unsigned long int
signed long long	unsigned long long
signed long long int	unsigned long long int
signed short	unsigned short
signed short int	unsigned short int
__float80	__float128
<complex.h>	
complex	_Complex
float complex	float _Complex
double complex	double _Complex
long double complex	long double _Complex
imaginary	_Imaginary
float imaginary	float _Imaginary
double imaginary	double _Imaginary
long double imaginary	long double _Imaginary
_Complex80	_Complex128
<stdint.h>	
intmax_t	uintmax_t
int8_t	uint8_t
int16_t	uint16_t
int32_t	uint32_t
int64_t	uint64_t
int_least8_t	uint_least8_t
int_least16_t	uint_least16_t
int_least32_t	uint_least32_t
int_least64_t	uint_least64_t
int_fast8_t	uint_fast8_t
int_fast16_t	uint_fast16_t
int_fast32_t	uint_fast32_t
int_fast64_t	uint_fast64_t
intptr_t	uintptr_t
<stdfix.h>	
_Fract	_Accum
_Sat _Fract	_Sat _Accum
<decimal.h>	
_Decimal32	_Decimal64
_Decimal128	_Complex _Decimal32

Literal Constant Suffixes

unsigned	U || u
unsigned long long	ULL
long	L
float	F
double	D
long double	L
__float80	W || w
__float128	Q || q
__ibm128	W
_Imaginary	i
_Complex128	KC
exponent	E
__Decimal32	df || DF
__Decimal64	dd || DD
__Decimal128	dl || DL
short _Fract || _Sat short _Fract	HR || hr
_Fract || _Sat _Fract	R || r
long _Fract || _Sat long _Fract	lr || LR
long long _Fract || _Sat long long _Fract	llr || LLR
unsigned short _Fract || _Sat unsigned short _Fract	uhr || UHR
unsigned _Fract || _Sat unsigned _Fract	ur || UR
unsigned long _Fract and _Sat unsigned long _Fract	ulr || ULR
unsigned long long _Fract || _Sat unsigned long long _Fract	ullr || ULLR
short _Accum || _Sat short _Accum	hk || HK
Accum || _Sat _Accum	k || K
long _Accum || _Sat long _Accum	lk || LK
long long _Accum || _Sat long long _Accum	llk || LLK
unsigned short _Accum || _Sat unsigned short _Accum	uhk || UHK
unsigned _Accum || _Sat unsigned _Accum	uk || UK
unsigned long _Accum || _Sat unsigned long _Accum	ulk || ULK
unsigned long long _Accum || _Sat unsigned long long _Accum	ullk || ULLK

https://gcc.gnu.org/onlinedocs/gcc/Fixed-Point.html

Literal Constant Prefixes

Octal	0
Binary	0b
Hexadecimal	0x
char	\u
wchar_t string	L
UTF-8 string	u8
UTF-16 string	u
UTF-32 string	U
Raw literal string	R"delimiter(STRING)delimiter"

Storage Classes

- **auto** - Default specifier; Local-scope
- **extern** - Lasts the whole program, block, or compilation unit; globally visible
- **register** - Stored in stack or CPU-register during the code block
- **static** - Lasts the whole program, block, or compilation unit; private in program
- **typedef** - Specifies a new datatype
- **_Thread_local** - Thread-local-storage; one instance per thread

Type Qualifiers

- **const** - Value does not change; read-only
- **restrict** - For the lifetime of the pointer, the object can only be accessed via the pointer
- **volatile** - Optimizing-compilers must not change
- **_Atomic** - Objects free from data races

Function Specifiers

- **inline** - Inline the function when compiling
- **_Noreturn** - The function does not return

Function Attributes (__attribute__(()))

GNU-GCC only

Use in function declaration (header)

https://gcc.gnu.org/onlinedocs/gcc/Function-Attributes.html

https://gcc.gnu.org/onlinedocs/gcc/Common-Function-Attributes.html

- **alias** - The function is an alias for another; Example: `void f () __attribute__ ((weak, alias ("__f")));`
- **aligned** - Set alignment
- **always_inline** - Inline the function
- **cold** - Unlikely to execute; used for optimizations
- **constructor** - Call function before main()
- **destructor** - Call function after main()
- **deprecated** - Emit warning msg when called
- **error** - Emit error message when called
- **flatten** - Inline all functions in the function; __attribute__((flatten))
- **hot** - Very likely to execute; used for optimizations
- **nonnull** - None of the input pointers are NULL
- **nothrow** - The function is guaranteed not to throw an exception
- **optimize** - Set specific optimization options for the function
- **pure** - The function accepts arguments, has single return, and has no other effects
- **returns_twice** - Returns two separate values
- **simd** - Create multiple functions that can process arguments using SIMD instructions
- **warning** - Emit warning message when called

C Programming Cheatsheet

Type Attributes

GNU-GCC only
https://gcc.gnu.org/onlinedocs/gcc/Type-Attributes.html

- **aligned** - Set alignment
- **deprecated** - Emit warning msg when called
- **mode** - Set type mode. Example: `typedef _Complex float __attribute((mode(TC))) _Complex128;`
- **packed** - Members of a struct or union are placed to minimize the memory required
- **unused** - Inform the compiler that members of a struct or union may appear unused; i.e. the compiler will not issue warnings

Variable Attributes

GNU-GCC only
https://gcc.gnu.org/onlinedocs/gcc/Variable-Attributes.html

- **aligned** - Set alignment
- **common** - Place variable in "common" storage; the common section of an object-file
- **deprecated** - Emit warning msg when called
- **nocommon** - Allocate space for the variable directly
- **unused** - Inform the compiler that members of a struct or union may appear unused, but that is fine; i.e. the compiler will not issue warnings
- **vector_size** - Set the variable's size in bytes and then divide it into parts; A size of 4 and a type of "char" would make the variable contain four "char" values

Special Macros and Keywords

- **__asm__** - Inline assembly code
- **__attribute__** - Function attribute
- **__auto_type** - Duck typing
- **__extension__** - Inform compiler that the following code is a GCC extension
- **_Generic** - Type-polymorphism mechanism
- **__GNUC__** - GNU-GCC compiler
- **__label__** - Create a local label by declaring it in the beginning of the scope (__label__ label;); then, place the actual label where needed (label:;)
- **__restrict__** - There is only one pointer to the referenced object; Example: `int FUNC(char *__restrict__ DATA) {}`

https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html
&&label - Address of *label*
typeof(*x) y - Declare y with x's type

Machine Modes

- **BI** - 1 Bit
- **QI** - Quarter Integer; 1 byte
- **HI** - Half Integer; 2 bytes
- **PSI** - Partial Single Integer; 4 bytes; not all bits used
- **SI** - Single Integer; 4 bytes
- **PDI** - Partial Double Integer; 8 bytes; not all bits used
- **DI** - Double Integer; 8 bytes
- **TI** - Tetra Integer; 16 bytes
- **OI** - Octa Integer; 32 bytes
- **QF** - Quarter Floating; 1 byte quarter-precision float-point
- **HF** - Half Floating; 2 byte half-precision float-point
- **TQF** - Three Quarter Floating; 3 byte three-quarter-precision float-point
- **SF** - Single Floating; 4 byte single-precision float-point
- **DF** - Double Floating; 8 byte double-precision float-point
- **XF** - Extended Floating; 12 byte extended-precision float-point
- **TF** - Tetra Floating; 16 byte tetra-precision float-point
- **CQI** - Complex Quarter Integer; 1 byte
- **CHI** - Complex Half Integer; 2 bytes
- **CSI** - Complex Single Integer; 4 bytes
- **CDI** - Complex Double Integer; 8 bytes
- **CTI** - Complex Tetra Integer; 16 bytes
- **COI** - Complex Octa Integer; 32 bytes
- **QC** - Quarter Complex; 1 byte quarter-precision complex float-point
- **HC** - Half Complex; 2 byte half-precision complex float-point
- **SC** - Single Complex; 4 byte single-precision complex float-point
- **DC** - Double Complex; 8 byte double-precision complex float-point
- **XC** - Extended Complex; 12 byte extended-precision complex float-point
- **TC** - Tetra Complex; 16 byte tetra-precision complex float-point
- **QQ** - Quarter-Fractional; 1-byte
- **HQ** - Half-Fractional; 2-byte
- **SQ** - Single-Fractional; 4-byte
- **DQ** - Double-Fractional; 8-byte
- **TQ** - Tetra-Fractional; 16-byte
- **UQQ** - Unsigned Quarter-Fractional; 1-byte
- **UHQ** - Unsigned Half-Fractional; 2-byte
- **USQ** - Unsigned Single-Fractional; 4-byte
- **UDQ** - Unsigned Double-Fractional; 8-byte
- **UTQ** - Unsigned Tetra-Fractional; 16-byte
- **HA** - Half-Accumulator; 2-byte
- **SA** - Single-Accumulator; 4-byte
- **DA** - Double-Accumulator; 8-byte
- **TA** - Tetra-Accumulator; 16-byte
- **UHA** - Unsigned Half-Accumulator; 2-byte
- **USA** - Unsigned Single-Accumulator; 4-byte
- **UDA** - Unsigned Double-Accumulator; 8-byte
- **UTA** - Unsigned Tetra-Accumulator; 16-byte
- **CC** - Condition Code
- **BLK** - Block
- **VOID** - Void
- **P** - Address mode
- **V4SI** - Vector; 4 single integers
- **V8QI** - Vector; 8 single-byte integers
- **BND32** - 32-bit pointer bound
- **BND64** - 32-bit pointer bound

https://gcc.gnu.org/onlinedocs/gccint/Machine-Modes.html

Printing Width-based Integrals

Datatype	Print Macros
int8_t	PRId8
uint8_t	PRIu8
int16_t	PRId16
uint16_t	PRIu16
uint64_t	PRIu64
intmax_t	PRIdMAX
int_least32_t	PRIdLEAST32
u int_fast32_t	PRIuFAST32
intptr_t	PRIdPTR

Replace "PRI" with "SCN" in scanf()

C POSIX Library

- <aio.h> - Asynchronous I/O
- <arpa/inet.h> - Functions for manipulating numeric IP addresses (part of Berkeley sockets)
- <assert.h> - Macros assertions
- <complex.h> - Arithmetic with complex numbers
- <cpio.h> - Magic numbers for the cpio archive format
- <dirent.h> - Functions for opening and listing directories
- <dlfcn.h> - Dynamic linking
- <errno.h> - Retrieving Error Number
- <fcntl.h> - File opening, locking, and other file operations
- <fenv.h> - Floating-Point environment
- <float.h> - Floating Types
- <fmtmsg.h> - Message display structures
- <fnmatch.h> - Filename matching
- <ftw.h> - File tree traversal
- <glob.h> - Pathname pattern-matching (globbing)
- <grp.h> - User group information and control
- <iconv.h> - Codeset conversion facility
- <inttypes.h> - Fixed-size integer data-types
- <iso646.h> - Alternative spellings
- <langinfo.h> - Language information constants
- <libgen.h> - Pathname manipulation
- <limits.h> - Implementation-defined constants
- <locale.h> - Category macros
- <math.h> - Mathematical and trigonometric functions
- <monetary.h> - Monetary unit string formatting
- <mqueue.h> - Message queue
- <ndbm.h> - NDBM database operations
- <net/if.h> - List local network interfaces
- <netdb.h> - Translating protocol and hostnames into numeric addresses
- <netinet/in.h> - Internet protocol and address family definitions
- <netinet/tcp.h> - Additional TCP control options
- <nl_types.h> - Localization message catalog functions
- <poll.h> - Asynchronous file descriptor multiplexing
- <pthread.h> - API for creating and manipulating POSIX threads
- <pwd.h> - passwd and user information access and control
- <regex.h> - Regular expression matching
- <sched.h> - Execution scheduling
- <search.h> - Search tables
- <semaphore.h> - POSIX semaphores
- <setjmp.h> - Stack environment declarations
- <signal.h> - Signals
- <spawn.h> - Process spawning
- <stdarg.h> - Handle Variable Argument List
- <stdbool.h> - Boolean type and values
- <stddef.h> - Standard Type Definitions
- <stdint.h> - Integer Types
- <stdio.h> - Standard Buffered I/O
- <stdlib.h> - Standard Library Definitions
- <string.h> - Several String Operations
- <strings.h> - Case-insensitive string comparisons
- <stropts.h> - Stream manipulation and ioctl
- <sys/ipc.h> - Inter-process communication (IPC)
- <sys/mman.h> - Memory management, POSIX Shared Memory, and Memory-mapped files
- <sys/msg.h> - POSIX message queues
- <sys/resource.h> - Resource usage, priorities, and limiting
- <sys/select.h> - Synchronous I/O multiplexing
- <sys/sem.h> - XSI (SysV style) semaphores
- <sys/shm.h> - XSI (SysV style) Shared Memory
- <sys/socket.h> - Main Berkley sockets header
- <sys/stat.h> - File information
- <sys/statvfs.h> - Filesystem information
- <sys/time.h> - Time and date functions and structures
- <sys/times.h> - File access and modification times
- <sys/types.h> - Various data-types
- <sys/uio.h> - Vectored I/O operations
- <sys/un.h> - Unix domain sockets
- <sys/utsname.h> - Operating system info and uname
- <sys/wait.h> - Status of terminated child processes
- <syslog.h> - System error logging
- <tar.h> - Magic numbers for the tar archive format
- <termios.h> - Terminal I/O interfaces
- <tgmath.h> - Type-Generic math macros
- <time.h> - Time macros
- <trace.h> - Tracing of runtime behavior
- <ulimit.h> - Resource limiting (DEPRECATED; use <sys/resource.h>)
- <unistd.h> - Various POSIX functions and constants
- <utime.h> - Inode access and modification times
- <utmpx.h> - User accounting database functions
- <wchar.h> - Wide-Character handling
- <wctype.h> - Wide-Character classification and mapping utilities
- <wordexp.h> - Word-expansion

C Programming Cheatsheet

Trigraphs

??=	#
??/	\
??'	^
??(	[
??)	]
??!	|
??<	{
??>	}
??-	~

Digraphs

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

Inline Assembly

```
asm [volatile] (
            { dialect0 | dialect1 | dialect2... }
             : OutputOperands
             [ : InputOperands [ : Clobbers ] ]
);  // Supported x86 dialects - ("att", "intel")

asm [volatile] goto (
            { dialect0 | dialect1 | dialect2... }
            : InputOperands
            : Clobbers
            : GotoLabels
);  // volatile disables some optimizations
```

Specify the assembler name for data: `int var_name asm ("asm_name") = 2;`
Specify the assembler name for functions: `int func(int x, int y) asm ("asm_func");`

```
uint32_t Mask = 1234;
uint32_t Index;
asm ("bsfl %1, %0;"
        : "=r"(Index)
        : "r"(Mask)
        : "cc");
```

Clobber Arguments

cc - Indicates that the assembler code modifies the flags register
memory - Informs the compiler that the assembly code performs memory reads or writes to items other than those listed in the input and output operands

Inline Assembly Modifiers

- **=** - Write
- **+** - Read & write
- **&** - Early clobber read & write
- **%** - Commutative; Only read-only operands can use "%"
- **#** - Ignored as a constraint
- ***** - Ignored as a register preference
- **?** - Slightly disparage constraint
- **!** - Severely disparage constraint
- **^** - Like "?", but only if operand needs reload
- **$** - Like "!", but only if operand needs reload
- **m** - Memory operand
- **o** - Offsetable memory operand
- **V** - Non-offsetable memory operand
- **<** - Autodecrement addressing memory operand
- **>** - Autoincrement addressing memory operand
- **r** - General register
- **i** - Immediate integer operand (constant)
- **n** - Immediate integer operand (static constant)
- **I-P** - Machine-dependent immediate integers
- **E** - Immediate float operand
- **F** - Immediate double or vector operand
- **G, H** - Machine-dependent float-operand
- **s** - Non-explicit immediate integer
- **g** - Register, memory, or immediate integer operand
- **X** - Any operand
- **0-9** - Specific operand (i.e. r12)
- **p** - Memory address operand

Inline x86 Assembly Modifiers

- **R** - Legacy register
- **q** - Any register accessible as rl
- **Q** - Any register accessible as rh
- **a** - The a register
- **b** - The b register
- **c** - The c register
- **d** - The d register
- **S** - The si register
- **D** - The di register
- **A** - The a & d registers
- **f** - Any 80387 floating-point (stack) register
- **t** - Top of 80387 floating-point stack (%st(0))
- **u** - %st(1)
- **y** - Any MMX register
- **x** - Any SSE register
- **Yz** - First SSE register (%xmm0)
- **I** - Integer constant in the range 0-31, for 32-bit shifts
- **J** - Integer constant in the range 0-63, for 64-bit shifts
- **K** - Signed 8-bit integer constant
- **L** - 0xFF or 0xFFFF, for andsi as a zero-extending move
- **M** - 0, 1, 2, or 3 (shifts for the lea instruction)
- **N** - Unsigned 8-bit integer constant (for in and out instructions)
- **G** - Standard 80387 floating point constant
- **C** - SSE constant zero operand
- **e** - 32-bit signed integer constant, or a symbolic reference known to fit that range (for immediate operands in sign-extending x86-64 instructions)
- **Z** - 32-bit unsigned integer constant, or a symbolic reference known to fit that range (for immediate operands in zero-extending x86-64 instructions)

https://gcc.gnu.org/onlinedocs/gcc/Machine-Constraints.html

Datatype Limits

char	Smallest addressable unit	[-128, 127] [0, 255]
short int	at least 16-bits	[-32768, 32767] [0, 65535]
int	at least 16-bits	[-32768, 32767] [0, 65535]
long int	at least 32-bits	[-2147483648, 2147483647] [0, 4294967295]
long long int	at least 64-bits	[-9223372036854775808, 9223372036854775807] [0, 18446744073709551615]
int128_t	128-bits	[-170141183460469231731687303715884105728, 170141183460469231731687303715884105727]
uint128_t	128-bits	[0, 340282366920938463463374607431768211456]

Floating-Point Datatypes

Type	Size	Exponent	Significand
float	32-bits	8	24
double	64-bits	11	53
long double	80-bits	15	64
Quadruple	128-bits	15	113
Octuple	256-bits	19	237
decimal32	32-bits	6	25
decimal64	64-bits	8	55
decimal128	128-bits	12	115

NOTE: The number of significand bits is implicit

Powers of Two

2^X	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536