

# C Reference Card (ANSI)

## Program Structure/Functions

<i>type fnc</i> ( <i>type</i> <sub>1</sub> , ...);	function prototype
<i>type name</i> ;	variable declaration
int main(void) {	main routine w/o args
<i>declarations</i>	local variable declarations
<i>statements</i>	
}	
<i>type fnc</i> ( <i>arg</i> <sub>1</sub> , ...) {	function definition
<i>declarations</i>	local variable declarations
<i>statements</i>	
return <i>value</i> ;	
}	
/* */	comments
int main(int argc, char *argv[])	main with args
exit( <i>arg</i> );	terminate execution

## C Preprocessor

include library file	#include <filename>
include user file	#include "filename"
replacement text	#define <i>name text</i>
replacement macro	#define <i>name</i> ( <i>var text</i> )
<i>Example. #define</i> max(A,B) ((A)>(B) ? (A) : (B))	
undefine	#undef <i>name</i>
quoted string in replace	#
<i>Example. #define</i> msg(A) printf("%s = %d", #A, (A))	
concatenate args and rescan	##
conditional execution	#if, #else, #elif, #endif
is <i>name</i> defined, not defined?	#ifdef, #ifndef
<i>name</i> defined?	defined( <i>name</i> )
line continuation char	\

## Data Types/Declarations

character (1 byte)	char
integer	int
real number (single, double precision)	float, double
short (16 bit integer)	short
long (32 bit integer)	long
double long (64 bit integer)	long long
positive or negative	signed
non-negative modulo 2 <sup><i>m</i></sup>	unsigned
pointer to int, float,...	int*, float*,...
enumeration constant	enum tag { <i>name</i> <sub>1</sub> = <i>value</i> <sub>1</sub> ,...};
constant (read-only) value	type const <i>name</i> ;
declare external variable	extern
internal to source file	static
local persistent between calls	static
no value	void
structure	struct tag {...};
create new name for data type	typedef type <i>name</i> ;
size of an object (type is <i>size_t</i> )	sizeof <i>object</i>
size of a data type (type is <i>size_t</i> )	sizeof ( <i>type</i> )

## Initialization

initialize variable	type <i>name</i> = <i>value</i> ;
initialize array	type <i>name</i> []={ <i>value</i> <sub>1</sub> ,...};
initialize char string	char <i>name</i> []="string";

## Constants

suffix: long, unsigned, float	65536L, -1U, 3.0F
exponential form	4.2e1
prefix: octal, hexadecimal	0, 0x or 0X
<i>Example. 031</i> is 25, <i>0x31</i> is 49 decimal	
character constant (char, octal, hex)	'a', '\ooo', '\xhh'
newline, cr, tab, backspace	\n, \r, \t, \b
special characters	\\, \?, \', \"
string constant (ends with '\0')	"abc...de"

## Pointers, Arrays & Structures

declare pointer to <i>type</i>	type * <i>name</i> ;
declare function returning pointer to <i>type type</i>	*f();
declare pointer to function returning <i>type type</i>	(*pf)();
generic pointer type	void *
null pointer constant	NULL
object pointed to by <i>pointer</i>	* <i>pointer</i>
address of object <i>name</i>	& <i>name</i>
array	<i>name</i> [ <i>dim</i> ]
multi-dim array	<i>name</i> [ <i>dim</i> <sub>1</sub> ][ <i>dim</i> <sub>2</sub> ]...

### Structures

struct tag {	structure template
<i>declarations</i>	declaration of members
};	
create structure	struct tag <i>name</i>
member of structure from template	<i>name.member</i>
member of pointed-to structure	<i>pointer -&gt; member</i>
<i>Example. (*p).x</i> and <i>p-&gt;x</i> are the same	
single object, multiple possible types	union
bit field with <i>b</i> bits	unsigned <i>member</i> : <i>b</i> ;

## Operators (grouped by precedence)

struct member operator	<i>name.member</i>
struct member through pointer	<i>pointer-&gt;member</i>
increment, decrement	++, --
plus, minus, logical not, bitwise not	+, -, !, ~
indirection via pointer, address of object	* <i>pointer</i> , & <i>name</i>
cast expression to type	( <i>type</i> ) <i>expr</i>
size of an object	sizeof
multiply, divide, modulus (remainder)	*, /, %
add, subtract	+, -
left, right shift [bit ops]	<<, >>
relational comparisons	>, >=, <, <=
equality comparisons	==, !=
and [bit op]	&
exclusive or [bit op]	^
or (inclusive) [bit op]	
logical and	&&
logical or	
conditional expression	<i>expr</i> <sub>1</sub> ? <i>expr</i> <sub>2</sub> : <i>expr</i> <sub>3</sub>
assignment operators	+=, -=, *=, ...
expression evaluation separator	,

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

## Flow of Control

statement terminator	;
block delimiters	{ }
exit from switch, while, do, for	break;
next iteration of while, do, for	continue;
go to	goto <i>label</i> ;
label	<i>label</i> : statement
return value from function	return <i>expr</i>

### Flow Constructions

if statement	if ( <i>expr</i> <sub>1</sub> ) <i>statement</i> <sub>1</sub> else if ( <i>expr</i> <sub>2</sub> ) <i>statement</i> <sub>2</sub> else <i>statement</i> <sub>3</sub>
while statement	while ( <i>expr</i> ) <i>statement</i>
for statement	for ( <i>expr</i> <sub>1</sub> ; <i>expr</i> <sub>2</sub> ; <i>expr</i> <sub>3</sub> ) <i>statement</i>
do statement	do <i>statement</i> while( <i>expr</i> );
switch statement	switch ( <i>expr</i> ) { case <i>const</i> <sub>1</sub> : <i>statement</i> <sub>1</sub> break; case <i>const</i> <sub>2</sub> : <i>statement</i> <sub>2</sub> break; default: <i>statement</i> }

## ANSI Standard Libraries

<assert.h>	<ctype.h>	<errno.h>	<float.h>	<limits.h>
<locale.h>	<math.h>	<setjmp.h>	<signal.h>	<stdarg.h>
<stddef.h>	<stdio.h>	<stdlib.h>	<string.h>	<time.h>

## Character Class Tests <ctype.h>

alphanumeric?	isalnum(c)
alphabetic?	isalpha(c)
control character?	iscntrl(c)
decimal digit?	isdigit(c)
printing character (not incl space)?	isgraph(c)
lower case letter?	islower(c)
printing character (incl space)?	isprint(c)
printing char except space, letter, digit?	ispunct(c)
space, formfeed, newline, cr, tab, vtab?	isspace(c)
upper case letter?	isupper(c)
hexadecimal digit?	isxdigit(c)
convert to lower case	tolower(c)
convert to upper case	toupper(c)

## String Operations <string.h>

s is a string; cs, ct are constant strings	
length of s	strlen(s)
copy ct to s	strcpy(s,ct)
only first n chars	strncpy(s,ct,n)
concatenate ct after s	strcat(s,ct)
only first n chars	strncat(s,ct,n)
compare cs to ct	strcmp(cs,ct)
only first n chars	strncmp(cs,ct,n)
pointer to first c in cs	strchr(cs,c)
pointer to last c in cs	strrchr(cs,c)
copy n chars from ct to s	memcpy(s,ct,n)
copy n chars from ct to s (may overlap)	memmove(s,ct,n)
compare n chars of cs with ct	memcmp(cs,ct,n)
pointer to first c in first n chars of cs	memchr(cs,c,n)
put c into first n chars of s	memset(s,c,n)

# C Reference Card (ANSI)

## Input/Output <stdio.h>

### Standard I/O

standard input stream	<code>stdin</code>
standard output stream	<code>stdout</code>
standard error stream	<code>stderr</code>
end of file (type is int)	<code>EOF</code>
get a character	<code>getchar()</code>
print a character	<code>putchar(<i>chr</i>)</code>
print formatted data	<code>printf("format", <i>arg</i><sub>1</sub>, ...)</code>
print to string <i>s</i>	<code>sprintf(<i>s</i>, "format", <i>arg</i><sub>1</sub>, ...)</code>
read formatted data	<code>scanf("format", &amp;<i>name</i><sub>1</sub>, ...)</code>
read from string <i>s</i>	<code>sscanf(<i>s</i>, "format", &amp;<i>name</i><sub>1</sub>, ...)</code>
print string <i>s</i>	<code>puts(<i>s</i>)</code>

### File I/O

declare file pointer	<code>FILE *<i>fp</i>;</code>
pointer to named file	<code>fopen("name", "mode")</code>
modes: <i>r</i> (read), <i>w</i> (write), <i>a</i> (append), <i>b</i> (binary)	
get a character	<code>getc(<i>fp</i>)</code>
write a character	<code>putc(<i>chr</i>, <i>fp</i>)</code>
push one char back to <i>fp</i>	<code>ungetc(<i>chr</i>, <i>fp</i>)</code>
only one push back is guaranteed	

write to file	<code>fprintf(<i>fp</i>, "format", <i>arg</i><sub>1</sub>, ...)</code>
read from file	<code>fscanf(<i>fp</i>, "format", <i>arg</i><sub>1</sub>, ...)</code>
read and store <i>n</i> elts to * <i>ptr</i>	<code>fread(*<i>ptr</i>, eltsize, <i>n</i>, <i>fp</i>)</code>
write <i>n</i> elts from * <i>ptr</i> to file	<code>fwrite(*<i>ptr</i>, eltsize, <i>n</i>, <i>fp</i>)</code>
close file	<code>fclose(<i>fp</i>)</code>
non-zero if error	<code>ferror(<i>fp</i>)</code>
non-zero if already reached EOF	<code>feof(<i>fp</i>)</code>
read line to string <i>s</i> (< max chars)	<code>fgets(<i>s</i>, max, <i>fp</i>)</code>
write string <i>s</i>	<code>fputs(<i>s</i>, <i>fp</i>)</code>

### Codes for Formatted I/O: "%-+ 0w.pmc"

-	left justify
+	print with sign
<i>space</i>	print space if no sign
0	pad with leading zeros
<i>w</i>	min field width
<i>p</i>	precision
<i>m</i>	conversion character:
<i>h</i>	short, <i>l</i> long, <i>L</i> long double
<i>c</i>	conversion character:
<i>d, i</i>	integer <i>u</i> unsigned
<i>c</i>	single char <i>s</i> char string
<i>f</i>	double ( <code>printf</code> ) <i>e, E</i> exponential
<i>f</i>	float ( <code>scanf</code> ) <i>lf</i> double ( <code>scanf</code> )
<i>o</i>	octal <i>x, X</i> hexadecimal
<i>p</i>	pointer <i>n</i> number of chars written
<i>g, G</i>	same as <i>f</i> or <i>e, E</i> depending on exponent

## Variable Argument Lists <stdarg.h>

declaration of pointer to arguments	<code>va_list <i>ap</i>;</code>
initialization of argument pointer	<code>va_start(<i>ap</i>, <i>lastarg</i>);</code>
<i>lastarg</i> is last named parameter of the function	
access next unnamed arg, update pointer	<code>va_arg(<i>ap</i>, <i>type</i>)</code>
call before exiting function	<code>va_end(<i>ap</i>);</code>

## Standard Utility Functions <stdlib.h>

absolute value of int <i>n</i>	<code>abs(<i>n</i>)</code>
absolute value of long <i>n</i>	<code>labs(<i>n</i>)</code>
quotient and remainder of ints <i>n, d</i>	<code>div(<i>n</i>, <i>d</i>)</code>
returns structure with <code>div_t.quot</code> and <code>div_t.rem</code>	
quotient and remainder of longs <i>n, d</i>	<code>ldiv(<i>n</i>, <i>d</i>)</code>
returns structure with <code>ldiv_t.quot</code> and <code>ldiv_t.rem</code>	
pseudo-random integer [0, RAND_MAX]	<code>rand()</code>
set random seed to <i>n</i>	<code>srand(<i>n</i>)</code>
terminate program execution	<code>exit(status)</code>
pass string <i>s</i> to system for execution	<code>system(<i>s</i>)</code>

### Conversions

convert string <i>s</i> to double	<code>atof(<i>s</i>)</code>
convert string <i>s</i> to integer	<code>atoi(<i>s</i>)</code>
convert string <i>s</i> to long	<code>atol(<i>s</i>)</code>
convert prefix of <i>s</i> to double	<code>strtod(<i>s</i>, &amp;endp)</code>
convert prefix of <i>s</i> (base <i>b</i> ) to long	<code>strtoul(<i>s</i>, &amp;endp, <i>b</i>)</code>
same, but unsigned long	<code>strtoul(<i>s</i>, &amp;endp, <i>b</i>)</code>

### Storage Allocation

allocate storage	<code>malloc(size), calloc(nobj, size)</code>
change size of storage	<code>newptr = realloc(ptr, size);</code>
deallocate storage	<code>free(ptr);</code>

### Array Functions

search array for key	<code>bsearch(key, array, <i>n</i>, size, cmpf)</code>
sort array ascending order	<code>qsort(array, <i>n</i>, size, cmpf)</code>

## Time and Date Functions <time.h>

processor time used by program	<code>clock()</code>
<i>Example.</i> <code>clock()/CLOCKS_PER_SEC</code> is time in seconds	
current calendar time	<code>time()</code>
<i>time</i> <sub>2</sub> - <i>time</i> <sub>1</sub> in seconds (double)	<code>difftime(<i>time</i><sub>2</sub>, <i>time</i><sub>1</sub>)</code>
arithmetic types representing times	<code>clock_t, time_t</code>
structure type for calendar time comps	<code>struct tm</code>

<i>tm_sec</i>	seconds after minute
<i>tm_min</i>	minutes after hour
<i>tm_hour</i>	hours since midnight
<i>tm_mday</i>	day of month
<i>tm_mon</i>	months since January
<i>tm_year</i>	years since 1900
<i>tm_wday</i>	days since Sunday
<i>tm_yday</i>	days since January 1
<i>tm_isdst</i>	Daylight Savings Time flag

convert local time to calendar time	<code>mktime(<i>tp</i>)</code>
convert time in <i>tp</i> to string	<code>asctime(<i>tp</i>)</code>
convert calendar time in <i>tp</i> to local time	<code>ctime(<i>tp</i>)</code>
convert calendar time to GMT	<code>gmtime(<i>tp</i>)</code>
convert calendar time to local time	<code>localtime(<i>tp</i>)</code>
format date and time info	<code>strftime(<i>s</i>, smax, "format", <i>tp</i>)</code>
<i>tp</i> is a pointer to a structure of type <code>tm</code>	

## Mathematical Functions <math.h>

Arguments and returned values are double

trig functions	<code>sin(x), cos(x), tan(x)</code>
inverse trig functions	<code>asin(x), acos(x), atan(x)</code>
<code>arctan(<i>y/x</i>)</code>	<code>atan2(<i>y</i>, <i>x</i>)</code>
hyperbolic trig functions	<code>sinh(x), cosh(x), tanh(x)</code>
exponentials & logs	<code>exp(x), log(x), log10(x)</code>
exponentials & logs (2 power)	<code>ldexp(x, <i>n</i>), frexp(x, &amp;<i>e</i>)</code>
division & remainder	<code>modf(x, ip), fmod(x, <i>y</i>)</code>
powers	<code>pow(x, <i>y</i>), sqrt(x)</code>
rounding	<code>ceil(x), floor(x), fabs(x)</code>

## Integer Type Limits <limits.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system, followed by minimum required values (if significantly different).

<code>CHAR_BIT</code>	bits in <code>char</code>	(8)
<code>CHAR_MAX</code>	max value of <code>char</code>	( <code>SCHAR_MAX</code> or <code>UCHAR_MAX</code> )
<code>CHAR_MIN</code>	min value of <code>char</code>	( <code>SCHAR_MIN</code> or 0)
<code>SCHAR_MAX</code>	max signed <code>char</code>	(+127)
<code>SCHAR_MIN</code>	min signed <code>char</code>	(-128)
<code>SHRT_MAX</code>	max value of <code>short</code>	(+32,767)
<code>SHRT_MIN</code>	min value of <code>short</code>	(-32,768)
<code>INT_MAX</code>	max value of <code>int</code>	(+2,147,483,647)
<code>INT_MIN</code>	min value of <code>int</code>	(-2,147,483,648)
<code>LONG_MAX</code>	max value of <code>long</code>	(+2,147,483,647)
<code>LONG_MIN</code>	min value of <code>long</code>	(-2,147,483,648)
<code>UCHAR_MAX</code>	max unsigned <code>char</code>	(255)
<code>USHRT_MAX</code>	max unsigned <code>short</code>	(65,535)
<code>UINT_MAX</code>	max unsigned <code>int</code>	(4,294,967,295) (65,535)
<code>ULONG_MAX</code>	max unsigned <code>long</code>	(4,294,967,295)

## Float Type Limits <float.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system.

<code>FLT_RADIX</code>	radix of exponent rep	(2)
<code>FLT_ROUNDS</code>	floating point rounding mode	
<code>FLT_DIG</code>	decimal digits of precision	(6)
<code>FLT_EPSILON</code>	smallest <i>x</i> so $1.0f + x \neq 1.0f$	( $1.1E - 7$ )
<code>FLT_MANT_DIG</code>	number of digits in mantissa	
<code>FLT_MAX</code>	maximum <code>float</code> number	( $3.4E38$ )
<code>FLT_MAX_EXP</code>	maximum exponent	
<code>FLT_MIN</code>	minimum <code>float</code> number	( $1.2E - 38$ )
<code>FLT_MIN_EXP</code>	minimum exponent	
<code>DBL_DIG</code>	decimal digits of precision	(15)
<code>DBL_EPSILON</code>	smallest <i>x</i> so $1.0 + x \neq 1.0$	( $2.2E - 16$ )
<code>DBL_MANT_DIG</code>	number of digits in mantissa	
<code>DBL_MAX</code>	max <code>double</code> number	( $1.8E308$ )
<code>DBL_MAX_EXP</code>	maximum exponent	
<code>DBL_MIN</code>	min <code>double</code> number	( $2.2E - 308$ )
<code>DBL_MIN_EXP</code>	minimum exponent	

March 2025 v2.3. Copyright © 2025 Joseph H. Silverman

Permission is granted to make and distribute copies of this card provided the copyright notice and this permission notice are preserved on all copies.

Send comments and corrections to J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. (jhs@math.brown.edu)