

C Reference Card (ANSI)

Program Structure/Functions

<code>type fnc(type₁,...)</code>	function declarations
<code>type name</code>	external variable declarations
<code>main() {</code>	main routine
<code>declarations</code>	local variable declarations
<code>statements</code>	
<code>}</code>	
<code>type fnc(arg₁,...) {</code>	function definition
<code>declarations</code>	local variable declarations
<code>statements</code>	
<code>return value;</code>	
<code>}</code>	
<code>/* */</code>	comments
<code>main(int argc, char *argv[])</code>	main with args
<code>exit(arg)</code>	terminate execution

C Preprocessor

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

Data Types/Declarations

character (1 byte)	<code>char</code>
integer	<code>int</code>
float (single precision)	<code>float</code>
float (double precision)	<code>double</code>
short (16 bit integer)	<code>short</code>
long (32 bit integer)	<code>long</code>
positive and negative	<code>signed</code>
only positive	<code>unsigned</code>
pointer to <code>int, float,...</code>	<code>*int, *float,...</code>
enumeration constant	<code>enum</code>
constant (unchanging) value	<code>const</code>
declare external variable	<code>extern</code>
register variable	<code>register</code>
local to source file	<code>static</code>
no value	<code>void</code>
structure	<code>struct</code>
create name by data type	<code>typedef typename</code>
size of an object (type is <code>size_t</code>)	<code>sizeof object</code>
size of a data type (type is <code>size_t</code>)	<code>sizeof(type name)</code>

Initialization

initialize variable	<code>type name=value</code>
initialize array	<code>type name[]={value₁,...}</code>
initialize char string	<code>char name[]="string"</code>

Constants

long (suffix)	L or l
float (suffix)	F or f
exponential form	e
octal (prefix zero)	0
hexadecimal (prefix zero-ex)	0x or 0X
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 <code>type</code>	<code>type *name</code>
declare function returning pointer to <code>type</code>	<code>type *f()</code>
declare pointer to function returning <code>type</code>	<code>type (*pf)()</code>
generic pointer type	<code>void *</code>
null pointer	<code>NULL</code>
object pointed to by <code>pointer</code>	<code>*pointer</code>
address of object <code>name</code>	<code>&name</code>
array	<code>name[dim]</code>
multi-dim array	<code>name[dim₁][dim₂]...</code>

Structures

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

Operators (grouped by precedence)

structure member operator	<code>name.member</code>
structure pointer	<code>pointer->member</code>
increment, decrement	<code>++, --</code>
plus, minus, logical not, bitwise not	<code>+, -, !, ~</code>
indirection via pointer, address of object	<code>*pointer, &name</code>
cast expression to type	<code>(type) expr</code>
size of an object	<code>sizeof</code>
multiply, divide, modulus (remainder)	<code>*, /, %</code>
add, subtract	<code>+, -</code>
left, right shift [bit ops]	<code><<, >></code>
comparisons	<code>>, >=, <, <=</code>
comparisons	<code>==, !=</code>
bitwise and	<code>&</code>
bitwise exclusive or	<code>^</code>
bitwise or (incl)	<code> </code>
logical and	<code>&&</code>
logical or	<code> </code>
conditional expression	<code>expr₁ ? expr₂ : expr₃</code>
assignment operators	<code>+=, -=, *=, ...</code>
expression evaluation separator	<code>,</code>

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

Flow of Control

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

Flow Constructions

<code>if statement</code>	<code>if (expr) statement</code> <code>else if (expr) statement</code> <code>else statement</code>
<code>while statement</code>	<code>while (expr)</code> <code>statement</code>
<code>for statement</code>	<code>for (expr₁; expr₂; expr₃)</code> <code>statement</code>
<code>do statement</code>	<code>do statement</code> <code>while(expr);</code>
<code>switch statement</code>	<code>switch (expr) {</code> <code>case const₁: statement₁ break;</code> <code>case const₂: statement₂ break;</code> <code>default: statement</code> <code>}</code>

ANSI Standard Libraries

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

Character Class Tests <ctype.h>

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

String Operations <string.h>

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

C Reference Card (ANSI)

Input/Output <stdio.h>

Standard I/O
 standard input stream `stdin`
 standard output stream `stdout`
 standard error stream `stderr`
 end of file `EOF`
 get a character `getchar()`
 print a character `putchar(chr)`
 print formatted data `printf("format", arg1, ...)`
 print to string `s` `sprintf(s, "format", arg1, ...)`
 read formatted data `scanf("format", &name1, ...)`
 read from string `s` `sscanf(s, "format", &name1, ...)`
 read line to string `s` (< max chars) `gets(s, max)`
 print string `s` `puts(s)`

File I/O
 declare file pointer `FILE *fp`
 pointer to named file `fopen("name", "mode")`
 modes: `r` (read), `w` (write), `a` (append)
 get a character `getc(fp)`
 write a character `putc(chr, fp)`
 write to file `fprintf(fp, "format", arg1, ...)`
 read from file `fscanf(fp, "format", arg1, ...)`
 close file `fclose(fp)`
 non-zero if error `ferror(fp)`
 non-zero if EOF `feof(fp)`
 read line to string `s` (< max chars) `fgets(s, max, fp)`
 write string `s` `fputs(s, fp)`

Codes for Formatted I/O: "%-+ 0w.pmc"
 - left justify
 + print with sign
 space print space if no sign
 0 pad with leading zeros
 w min field width
 p precision
 m conversion character:
 h short, l long, L long double
 c conversion character:
 d,i integer u unsigned
 c single char s char string
 f double e,E exponential
 o octal x,X hexadecimal
 p pointer n number of chars written
 g,G same as f or e,E depending on exponent

Variable Argument Lists <stdarg.h>

declaration of pointer to arguments `va_list name;`
 initialization of argument pointer `va_start(name, lastarg)`
lastarg is last named parameter of the function
 access next unnamed arg, update pointer `va_arg(name, type)`
 call before exiting function `va_end(name)`

Standard Utility Functions <stdlib.h>

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

Conversions
 convert string `s` to double `atof(s)`
 convert string `s` to integer `atoi(s)`
 convert string `s` to long `atol(s)`
 convert prefix of `s` to double `strtod(s, endp)`
 convert prefix of `s` (base `b`) to long `strtol(s, endp, b)`
 same, but unsigned long `strtoul(s, endp, b)`

Storage Allocation
 allocate storage `malloc(size), calloc(nobj, size)`
 change size of object `realloc(pts, size)`
 deallocate space `free(ptr)`

Array Functions
 search array for key `bsearch(key, array, n, size, cmp())`
 sort array ascending order `qsort(array, n, size, cmp())`

Time and Date Functions <time.h>

processor time used by program `clock()`
Example. `clock()/CLOCKS_PER_SEC` is time in seconds
 current calendar time `time()`
`time2-time1` in seconds (double) `difftime(time2, time1)`
 arithmetic types representing times `clock_t, time_t`
 structure type for calendar time comps `tm`
 `tm_sec` seconds after minute
 `tm_min` minutes after hour
 `tm_hour` hours since midnight
 `tm_mday` day of month
 `tm_mon` months since January
 `tm_year` years since 1900
 `tm_wday` days since Sunday
 `tm_yday` days since January 1
 `tm_isdst` Daylight Savings Time flag

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

Mathematical Functions <math.h>

Arguments and returned values are double

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

Integer Type Limits <limits.h>

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

`CHAR_BIT` bits in char (8)
`CHAR_MAX` max value of char (127 or 255)
`CHAR_MIN` min value of char (-128 or 0)
`INT_MAX` max value of int (+32,767)
`INT_MIN` min value of int (-32,768)
`LONG_MAX` max value of long (+2,147,483,647)
`LONG_MIN` min value of long (-2,147,483,648)
`SCHAR_MAX` max value of signed char (+127)
`SCHAR_MIN` min value of signed char (-128)
`SHRT_MAX` max value of short (+32,767)
`SHRT_MIN` min value of short (-32,768)
`UCHAR_MAX` max value of unsigned char (255)
`UINT_MAX` max value of unsigned int (65,535)
`ULONG_MAX` max value of unsigned long (4,294,967,295)
`USHRT_MAX` max value of unsigned short (65,536)

Float Type Limits <float.h>

`FLT_RADIX` radix of exponent rep (2)
`FLT_ROUNDS` floating point rounding mode
`FLT_DIG` decimal digits of precision (6)
`FLT_EPSILON` smallest x so $1.0 + x \neq 1.0$ (10^{-5})
`FLT_MANT_DIG` number of digits in mantissa
`FLT_MAX` maximum floating point number (10^{37})
`FLT_MAX_EXP` maximum exponent
`FLT_MIN` minimum floating point number (10^{-37})
`FLT_MIN_EXP` minimum exponent
`DBL_DIG` decimal digits of precision (10)
`DBL_EPSILON` smallest x so $1.0 + x \neq 1.0$ (10^{-9})
`DBL_MANT_DIG` number of digits in mantissa
`DBL_MAX` max double floating point number (10^{37})
`DBL_MAX_EXP` maximum exponent
`DBL_MIN` min double floating point number (10^{-37})
`DBL_MIN_EXP` minimum exponent

May 1999 v1.3. Copyright © 1999 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)