

C Reference Card (ANSI)

Program Structure/Functions

| | |
|---|-----------------------------|
| <code>type fnc(type₁, ...);</code> | function prototype |
| <code>type name;</code> | variable declaration |
| <code>int main(void) {</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>int 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>Example. #define msg(A) printf("%s = %d", #A, (A))</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> |
| real number (single, double precision) | <code>float, double</code> |
| short (16 bit integer) | <code>short</code> |
| long (32 bit integer) | <code>long</code> |
| double long (64 bit integer) | <code>long long</code> |
| positive or negative | <code>signed</code> |
| non-negative modulo 2 ^m | <code>unsigned</code> |
| pointer to int, float,... | <code>int*, float*,...</code> |
| enumeration constant | <code>enum tag {name₁=value₁,...};</code> |
| constant (read-only) value | <code>type const name;</code> |
| declare external variable | <code>extern</code> |
| internal to source file | <code>static</code> |
| local persistent between calls | <code>static</code> |
| no value | <code>void</code> |
| structure | <code>struct tag {...};</code> |
| create new name for data type | <code>typedef type name;</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)</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

| | |
|---|---------------------|
| suffix: long, unsigned, float | 65536L, -1U, 3.0F |
| exponential form | 4.2e1 |
| prefix: octal, hexadecimal | 0, 0x or 0X |
| <code>Example. 031 is 25, 0x31 is 49 decimal</code> | |
| 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> | <code>type *name;</code> |
| declare function returning pointer to <i>type</i> | <code>type *f();</code> |
| declare pointer to function returning <i>type</i> | <code>type (*pf)();</code> |
| generic pointer type | <code>void *</code> |
| null pointer constant | <code>NULL</code> |
| object pointed to by <i>pointer</i> | <code>*pointer</code> |
| address of object <i>name</i> | <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 object, multiple possible types | <code>union</code> |
| bit field with <i>b</i> bits | <code>unsigned member: b;</code> |

Operators (grouped by precedence)

| | |
|--|---|
| struct member operator | <code>name.member</code> |
| struct member through 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> |
| relational comparisons | <code>>, >=, <, <=</code> |
| equality comparisons | <code>==, !=</code> |
| and [bit op] | <code>&</code> |
| exclusive or [bit op] | <code>^</code> |
| or (inclusive) [bit op] | <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 switch, while, do, for | <code>break;</code> |
| next iteration of while, do, for | <code>continue;</code> |
| go to | <code>goto label;</code> |
| label | <code>label: statement</code> |
| return value from function | <code>return expr</code> |

Flow Constructions

| | |
|------------------|---|
| if statement | <code>if (expr₁) statement₁</code> <code>else if (expr₂) statement₂</code> <code>else statement₃</code> |
| while statement | <code>while (expr)</code> <code>statement</code> |
| for statement | <code>for (expr₁; expr₂; expr₃)</code> <code>statement</code> |
| do statement | <code>do statement</code> <code>while(expr);</code> |
| switch statement | <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>isctrl(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</code> is a string; <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> |
| concatenate <code>ct</code> after <code>s</code> | <code>strcat(s,ct)</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>s</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 (type is int) | EOF |
| get a character | getchar() |
| print a character | putchar(<i>chr</i>) |
| print formatted data | printf("format", <i>arg1</i> , ...) |
| print to string <i>s</i> | sprintf(<i>s</i> , "format", <i>arg1</i> , ...) |
| read formatted data | scanf("format", & <i>name1</i> , ...) |
| read from string <i>s</i> | sscanf(<i>s</i> , "format", & <i>name1</i> , ...) |
| print string <i>s</i> | puts(<i>s</i>) |

File I/O

| | |
|---|---|
| declare file pointer | FILE * <i>fp</i> ; |
| pointer to named file | fopen("name", "mode") modes: r (read), w (write), a (append), b (binary) |
| get a character | getc(<i>fp</i>) |
| write a character | putc(<i>chr</i> , <i>fp</i>) |
| write to file | fprintf(<i>fp</i> , "format", <i>arg1</i> , ...) |
| read from file | fscanf(<i>fp</i> , "format", <i>arg1</i> , ...) |
| read and store <i>n</i> elts to * <i>ptr</i> | fread(* <i>ptr</i> , <i>eltsize</i> , <i>n</i> , <i>fp</i>) |
| write <i>n</i> elts from * <i>ptr</i> to file | fwrite(* <i>ptr</i> , <i>eltsize</i> , <i>n</i> , <i>fp</i>) |
| close file | fclose(<i>fp</i>) |
| non-zero if error | ferror(<i>fp</i>) |
| non-zero if already reached EOF | feof(<i>fp</i>) |
| read line to string <i>s</i> (< max chars) | fgets(<i>s</i> , <i>max</i> , <i>fp</i>) |
| write string <i>s</i> | fputs(<i>s</i> , <i>fp</i>) |

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 (printf) e,E exponential f float (scanf) lf double (scanf) 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 <i>ap</i> ; |
| initialization of argument pointer | va_start(<i>ap</i> , <i>lastarg</i>); <i>lastarg</i> is last named parameter of the function |
| access next unnamed arg, update pointer | va_arg(<i>ap</i> , <i>type</i>) |
| call before exiting function | va_end(<i>ap</i>); |

Standard Utility Functions <stdlib.h>

| | |
|---|--|
| absolute value of int <i>n</i> | abs(<i>n</i>) |
| absolute value of long <i>n</i> | labs(<i>n</i>) |
| quotient and remainder of ints <i>n,d</i> | div(<i>n</i> , <i>d</i>) returns structure with <i>div_t.quot</i> and <i>div_t.rem</i> |
| quotient and remainder of longs <i>n,d</i> | ldiv(<i>n</i> , <i>d</i>) returns structure with <i>ldiv_t.quot</i> and <i>ldiv_t.rem</i> |
| pseudo-random integer [0, RAND_MAX] | rand() |
| set random seed to <i>n</i> | srand(<i>n</i>) |
| terminate program execution | exit(<i>status</i>) |
| pass string <i>s</i> to system for execution | system(<i>s</i>) |
| Conversions | |
| convert string <i>s</i> to double | atof(<i>s</i>) |
| convert string <i>s</i> to integer | atoi(<i>s</i>) |
| convert string <i>s</i> to long | atol(<i>s</i>) |
| convert prefix of <i>s</i> to double | strtod(<i>s</i> , & <i>endp</i>) |
| convert prefix of <i>s</i> (base <i>b</i>) to long | strtoul(<i>s</i> , & <i>endp</i> , <i>b</i>) |
| same, but unsigned long | strtoul(<i>s</i> , & <i>endp</i> , <i>b</i>) |

Storage Allocation

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

Array Functions

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

Time and Date Functions <time.h>

| | |
|---|--|
| processor time used by program | clock() |
| <i>Example.</i> clock()/CLOCKS_PER_SEC is time in seconds | |
| current calendar time | time() |
| time ₂ -time ₁ in seconds (double) | difftime(time ₂ , time ₁) |
| arithmetic types representing times | clock_t, time_t |
| structure type for calendar time comps | struct 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(<i>tp</i>) |
| convert time in <i>tp</i> to string | asctime(<i>tp</i>) |
| convert calendar time in <i>tp</i> to local time | ctime(<i>tp</i>) |
| convert calendar time to GMT | gmtime(<i>tp</i>) |
| convert calendar time to local time | localtime(<i>tp</i>) |
| format date and time info | strftime(<i>s</i> , <i>smax</i> , "format", <i>tp</i>) <i>tp</i> 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) |
| arctan(<i>y/x</i>) | atan2(<i>y</i> , <i>x</i>) |
| hyperbolic trig functions | sinh(x), cosh(x), tanh(x) |
| exponentials & logs | exp(x), log(x), log10(x) |
| exponentials & logs (2 power) | ldexp(x, <i>n</i>), frexp(x, & <i>e</i>) |
| division & remainder | modf(x, <i>ip</i>), fmod(x, <i>y</i>) |
| powers | pow(x, <i>y</i>), 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, followed by minimum required values (if significantly different).

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

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

January 2007 v2.2. Copyright © 2007 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)