C Reference Card (ANSI)

Program Structure/Functions

type fnc(*type*$_1$, ...);	function prototype
type name;	variable declaration
`int main(void) {`	main routine
declarations	local variable declarations
statements	
`}`	
type fnc(*arg*$_1$, ...) `{`	function definition
declarations	local variable declarations
statements	
`return` *value*;	
`}`	
`/*  */`	comments
`int main(int argc, char *argv[])`	main with args
`exit(`*arg*`);`	terminate execution

C Preprocessor

include library file	`#include <`*filename*`>`
include user file	`#include "`*filename*`"`
replacement text	`#define` *name text*
replacement macro	`#define` *name*(*var*) *text*

    Example. `#define max(A,B) ((A)>(B) ? (A) : (B))`

undefine	`#undef` *name*
quoted string in replace	`#`

    Example. `#define msg(A) printf("%s = %d", #A, (A))`

concatenate args and rescan	`##`
conditional execution	`#if, #else, #elif, #endif`
is *name* defined, not defined?	`#ifdef, #ifndef`
name defined?	`defined(`*name*`)`
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^m	`unsigned`
pointer to `int`, `float`,...	`int*, float*,...`
enumeration constant	`enum` *tag* `{`*name*$_1$`=`*value*$_1$`,...};`
constant (read-only) value	*type* `const` *name*;
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 name*;
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type*`)`

Initialization

initialize variable	*type name*`=`*value*;
initialize array	*type name*`[]={`*value*$_1$`,...};`
initialize char string	`char` *name*`[]="`*string*`";`

Constants

suffix: long, unsigned, float	`65536L, -1U, 3.0F`
exponential form	`4.2e1`
prefix: octal, hexadecimal	`0, 0x` or `0X`

    Example. `031` is 25, `0x31` is 49 decimal

character constant (char, octal, hex)	`'a', '\`*ooo*`', '\x`*hh*`'`
newline, cr, tab, backspace	`\n, \r, \t, \b`
special characters	`\\, \?, \', \"`
string constant (ends with `'\0'`)	`"abc...de"`

Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*;
declare function returning pointer to *type*	*type* `*f();`
declare pointer to function returning *type*	*type* `(*pf)();`
generic pointer type	`void *`
null pointer constant	`NULL`
object pointed to by *pointer*	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[`*dim*`]`
multi-dim array	*name*`[`*dim*$_1$`][`*dim*$_2$`]...`

Structures

`struct` *tag* `{`	structure template
declarations	declaration of members
`};`	

create structure	`struct` *tag name*
member of structure from template	*name*`.`*member*
member of pointed-to structure	*pointer* `->` *member*

    Example. `(*p).x` and `p->x` are the same

single object, multiple possible types	`union`
bit field with *b* bits	`unsigned` *member*`:` *b*;

Operators (grouped by precedence)

struct member operator	*name*`.`*member*
struct member through pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*, `&`*name*
cast expression to type	`(`*type*`)` *expr*
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	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
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` *label*;
label	*label*`:` `statement`
return value from function	`return` *expr*

Flow Constructions

`if` statement	`if (`*expr*$_1$`)` *statement*$_1$ `else if (`*expr*$_2$`)` *statement*$_2$ `else` *statement*$_3$

`while` statement	`while (`*expr*`)`
	  statement

`for` statement	`for (`*expr*$_1$`;` *expr*$_2$`;` *expr*$_3$`)`
	  statement

`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	   `case` *const*$_1$`:` *statement*$_1$ `break;`
	   `case` *const*$_2$`:` *statement*$_2$ `break;`
	   `default:` *statement*
	`}`

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

String Operations `<string.h>`

`s` is a string; `cs`, `ct` are constant strings

length of `s`	`int strlen(s)`
copy `ct` to `s`	`char *strcpy(s, ct)`
concatenate `ct` after `s`	`char *strcat(s,ct)`
compare `cs` to `ct`	`int strcmp(cs,ct)`
only first `n` chars	`int strncmp(cs,ct,n)`
pointer to first `c` in `cs`	`char *strchr(cs,c)`
pointer to last `c` in `cs`	`char *strrchr(cs,c)`
copy `n` chars from `ct` to `s`	`void *memcpy(s,ct,n)`
copy `n` chars from `ct` to `s` (may overlap)	`void *memmove(s,ct,n)`
compare `n` chars of `cs` with `ct`	`int memcmp(cs,ct,n)`
pointer to first `c` in first `n` chars of `cs`	`void *memchr(cs,c,n)`
put `c` into first `n` chars of `s`	`void *memset(s,c,n)`

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	`int getchar(void)`
print a character	`int putchar(int `*chr*`)`
print formatted data	`int printf("`*format*`",`arg_1`,...)`
print to string `s`	`int sprintf(s,"`*format*`",`arg_1`,...)`
read formatted data	`int scanf("`*format*`",&`$name_1$`,...)`
read from string `s`	`int sscanf(s,"`*format*`",&`$name_1$`,...)`
print string `s`	`int puts(s)`

File I/O

declare file pointer	`FILE *`*fp*`;`
pointer to named file	`FILE *fopen("`*name*`","`*mode*`")`
modes: `r` (read), `w` (write), `a` (append), `b` (binary)	
get a character	`int getc(`*fp*`)`
write a character	`int putc(`*chr*`,`*fp*`)`
write to file	`int fprintf(`*fp*`,"`*format*`",`arg_1`,...)`
read from file	`int fscanf(`*fp*`,"`*format*`",`arg_1`,...)`
read and store `n` elts to `*ptr`	`size_t fread(*ptr,eltsize,n,`*fp*`)`
write `n` elts from `*ptr` to file	`size_t fwrite(*ptr,eltsize,n,`*fp*`)`
close file	`int fclose(`*fp*`)`
non-zero if error	`int ferror(`*fp*`)`
non-zero if already reached EOF	`int feof(`*fp*`)`
current value file position	`long ftell(`*fp*`)`
set file position	`int fseek(`*fp*`, offset, origin)`
modes: `r` (read), `w` (write), `a` (append), `b` (binary)	
read line to string `s` (< `max` chars)	`char *fgets(s,max,`*fp*`)`
write string `s`	`int fputs(s,`*fp*`)`

Codes for Formatted I/O: `"%-+ 0`*w.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 `*ap*`;`
initialization of argument pointer	`va_start(`*ap*`,`*lastarg*`);`
lastarg is last named parameter of the function	
access next unnamed arg, update pointer	`va_arg(`*ap*`,`*type*`)`
call before exiting function	`va_end(`*ap*`);`

Standard Utility Functions `<stdlib.h>`

absolute value of `int n`	`int abs(n)`
absolute value of `long n`	`long labs(n)`
quotient and remainder of `ints n,d`	`div_t 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]`	`int rand()`
set random seed to `n`	`void srand(n)`
terminate program execution	`void exit(status)`
pass string `s` to system for execution	`int system(s)`

Conversions

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

Storage Allocation

allocate storage	`void *malloc(size), void *calloc(nobj,size)`
change size of storage	`newptr = realloc(ptr,size);`
deallocate storage	`void free(ptr);`

Array Functions

search `array` for `key`	`void *bsearch(key,array,n,size,cmpf)`
sort `array` ascending order	`void qsort(array,n,size,cmpf)`

Time and Date Functions `<time.h>`

processor time used by program	`clock_t clock(void)`
Example. `clock()/CLOCKS_PER_SEC` is time in seconds	
current calendar time	`time_t time(void)`
$\texttt{time}_2$`-`$\texttt{time}_1$ in seconds (`double`)	`double difftime(`$\texttt{time}_2$`,`$\texttt{time}_1$`)`
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	`time_t mktime(tp)`
convert time in `tp` to string	`char *asctime(tp)`
convert calendar time in `tp` to local time	`char *ctime(tp)`
convert calendar time to GMT	`struct tm *gmtime(tp)`
convert calendar time to local time	`struct tm *localtime(tp)`
format date and time info	`size_t strftime(s,smax,"`*format*`",tp)`
`tp` is a pointer to a structure of type `tm`	

Boolean types `<stdbool.h>`

boolean	`bool`
true (1)	`true`
false (0)	`false`

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(y/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, 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 x so $1.0\text{f} + x \neq 1.0\text{f}$	($1.1\text{E} - 7$)
`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	($1.2\text{E} - 38$)
`FLT_MIN_EXP`	minimum exponent	
`DBL_DIG`	decimal digits of precision	(15)
`DBL_EPSILON`	smallest x so $1.0 + x \neq 1.0$	($2.2\text{E} - 16$)
`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	($2.2\text{E} - 308$)
`DBL_MIN_EXP`	minimum exponent	