C Reference Card (ANSI)

Program Structure/Functions

type fnc(*type*$_1$,...)	function declarations
type name	external variable declarations
`main() {`	main routine
declarations	local variable declarations
statements	
`}`	
type fnc(*arg*$_1$,...) {	function definition
declarations	local variable declarations
statements	
`return` *value*;	
`}`	
`/* */`	comments
`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	`#`
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`
float (single precision)	`float`
float (double precision)	`double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
positive and negative	`signed`
only positive	`unsigned`
pointer to `int, float,`...	`*int, *float,`...
enumeration constant	`enum`
constant (unchanging) value	`const`
declare external variable	`extern`
register variable	`register`
local to source file	`static`
no value	`void`
structure	`struct`
create name by data type	`typedef` *typename*
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type name*`)`

Initialization

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

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*`', '\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	`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 value, multiple type structure	`union`
bit field with *b* bits	*member* : *b*

Operators (grouped by precedence)

structure member operator	*name*.*member*
structure 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]	`<<, >>`
comparisons	`>, >=, <, <=`
comparisons	`==, !=`
bitwise and	`&`
bitwise exclusive or	`^`
bitwise or (incl)	`|`
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 delimeters	`{ }`
exit from `switch, while, do, for`	`break`
next iteration of `while, do, for`	`continue`
go to	`goto` *label*
label	*label*:
return value from function	`return` *expr*

Flow Constructions

`if` statement	`if (`*expr*`)` *statement*
	`else if (`*expr*`)` *statement*
	`else` *statement*
`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?	`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,t are strings, cs,ct are constant strings

length of `s`	`strlen(s)`
copy `ct` to `s`	`strcpy(s,ct)`
up to `n` chars	`strncpy(s,ct,n)`
concatenate `ct` after `s`	`strcat(s,ct)`
up to `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 `cs`	`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	EOF
get a character	getchar()
print a character	putchar(*chr*)
print formatted data	printf("*format*",arg_1,...)
print to string s	sprintf(s,"*format*",arg_1,...)
read formatted data	scanf("*format*",&$name_1$,...)
read from string s	sscanf(s,"*format*",&$name_1$,...)
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*",arg_1,...)
read from file	fscanf(*fp*,"*format*",arg_1,...)
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: "%-+ 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
c	single char
f	double
o	octal
p	pointer

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 unamed 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 **int**s n,d	div(n,d)
retursn structure with **div_t.quot** and **div_t.rem**	
quotient and remainder of **long**s 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()
$time_2$-$time_1$ in seconds (**double**)	difftime($time_2$,$time_1$)
arithmetic types representing times	clock_t,time_t
structure type for calendar time comps	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)
$\arctan(y/x)$	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.

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 RI 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	