

C Reference Card (ANSI)

Program Structure/Functions

```
type fnc(type1, ...);           function prototype
type name;                       variable declaration
int main(void) {                 main routine
    declarations                 local variable declarations
    statements
}
type fnc(arg1, ...) {           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 2m         unsigned
pointer to int, float, ...      int*, float*, ...
enumeration constant            enum tag {name1=value1, ...};
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[]={value1, ...};
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', '\xhh'
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[dim₁][dim₂]...
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         expr1 ? expr2 : expr3
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 (expr1) statement1
                                else if (expr2) statement2
                                else statement3
while statement                 while (expr)
                                statement
for statement                   for (expr1; expr2; expr3)
                                statement
do statement                    do statement
                                while(expr);
switch statement                switch (expr) {
                                case const1: statement1 break;
                                case const2: statement2 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 is a string; cs, ct are constant strings

```
length of s                    strlen(s)
copy ct to s                   strcpy(s,ct)
concatenate ct after s         strcat(s,ct)
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	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</i> , <i>d</i>	div(<i>n</i> , <i>d</i>) returns structure with div_t.quot and div_t.rem
quotient and remainder of longs <i>n</i> , <i>d</i>	ldiv(<i>n</i> , <i>d</i>) returns structure with ldiv_t.quot and ldiv_t.rem
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> ,&endp)
convert prefix of <i>s</i> (base <i>b</i>) to long	strtoul(<i>s</i> ,&endp, <i>b</i>)
same, but unsigned long	strtoul(<i>s</i> ,&endp, <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()
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	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(<i>x</i>), cos(<i>x</i>), tan(<i>x</i>)
inverse trig functions	asin(<i>x</i>), acos(<i>x</i>), atan(<i>x</i>)
arctan(<i>y/x</i>)	atan2(<i>y</i> , <i>x</i>)
hyperbolic trig functions	sinh(<i>x</i>), cosh(<i>x</i>), tanh(<i>x</i>)
exponentials & logs	exp(<i>x</i>), log(<i>x</i>), log10(<i>x</i>)
exponentials & logs (2 power)	ldexp(<i>x</i> , <i>n</i>), frexp(<i>x</i> ,& <i>e</i>)
division & remainder	modf(<i>x</i> , <i>ip</i>), fmod(<i>x</i> , <i>y</i>)
powers	pow(<i>x</i> , <i>y</i>), sqrt(<i>x</i>)
rounding	ceil(<i>x</i>), floor(<i>x</i>), fabs(<i>x</i>)

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)