MICRO LOGIC, POB 174
HACKENSACK, NJ 07602

C LANGUAGE

PROGRAMMER'S INSTANT REFERENCE CARD

MICRO CHART®

INTRODUCTION

This card is a concise comprehensive reference for C language programmers and those learning C. It saves you time and lets you avoid cumbersome manuals.

The C programming language is becoming the standard language for developing both system and application programs. There are several reasons for its popularity. C is flexible with few restrictions on the programmer. C compilers produce fast and short machine code. And finally, C is the primary language used in the UNIX (trademark of AT&T Bell Laboratories) operating system (over 90% of the UNIX system is itself written in C). Because it is a popular "high level" language, it allows software to be used on many machines without being rewritten.

This card is organized so that you can keep your train of thought while programming in C (without stopping to flip thru a manual.) The result is fewer interruptions, more error-free code, and higher productivity.

The following notations are used: [ ]--enclosed item is optional; fn--function; rtn--return; ptd--pointed; ptr--pointer; TRUE--non-zero value; FALSE--zero value.

BASIC DATA TYPES

TYPE	DESCRIPTION
char	Single character
double	Extended precision floating pt
float	Floating point
int	Integer
long int	Extended precision integer
short int	Reduced precision integer
unsigned char	Non-negative character
unsigned int	Non-negative integer
void	No type; used for fn declarations and 'ignoring' a value returned from a fn

CONVERSION OF DATA TYPES

Before performing an arithmetic operation, operands are made consistent with each other by converting with this procedure:

1. All float operands are converted to double. All char or short operands are converted to int.
2. If either operand is double, the other is converted to double. The result is double.
3. If either operand is long int, the other is converted to long int. The result is long int.
4. If either operand is unsigned, the other is converted to unsigned. The result is unsigned.
5. If this step is reached, both operands must be of type int. The result will be int

STATEMENT SUMMARY

STATEMENT	DESCRIPTION
`break;`	Terminates execution of for, while, do, or switch
`continue;`	Skips statements that follow in a do, for, or while; then continues executing the loop

`do`
`   statement`
`while ( expr );`
	Executes statement until expr is FALSE; statement is executed at least once

`for ( e1; e2; e3 )` `   statement`	Evaluates expression e1 once; then repeatedly evaluates e2, statement, and e3 (in that order) until e2 is FALSE; eg: for (i=1; i<=10; ++i)...; note that statement might not be executed if e2 is FALSE on first evaluation
`goto label;`	Branches to statement preceded by label:, which must be in same function as the goto
`if ( expr )` `   statement`	If expr is TRUE, then executes statement; otherwise skips it

`if ( expr )`
`   statement1`
`else`
`   statement2`
	If expr is TRUE, then executes statement1; otherwise executes statement2

`; (null statement)`	No effect; satisfies statement requirement in do, for, and while
`return;`	Returns from function back to caller; no value returned
`return expr;`	Returns from function back to caller with value of expr

`switch ( iexpr )`
`  {`
`    case const1:`
`       statement`
`       ...`
`       break;`
`    case const2:`
`       statement`
`       ...`
`       break;`
`    ...`
`    default:`
`       statement`
`       ...`
`       break;`
`  }`
	iexpr is evaluated and then compared against integer constant exprs const1, const2, ...; if a match is found, then the statements that follow the case (up to the break) will be executed; if no match is found, then the statements in the default case (if supplied) will be executed; iexpr must be an integer-valued expression

`while ( expr )` `   statement`	Executes statement as long as expr is TRUE; statement might not be executed if expr is FALSE the first time it's evaluated

NOTES:

expr is any expression; statement is any expression terminated by a semicolon, one of the statements listed above, or one or more statements enclosed by braces {...}.

OPERATORS

OPER	DESCRIPTION	EXAMPLE	ASSOC
( )	Function call	sqrt (x)	
[ ]	Array element ref	vals[10]	L-R
->	Ptr to struc memb	emp_ptr->name	
.	Struc member ref	employee.name	
-	Unary minus	-a	
++	Increment	++ptr	
--	Decrement	--count	
!	Logical negation	! done	R-L
~	Ones complement	~077	
*	Ptr indirection	*ptr	
&	Address of	&x	
sizeof	Size in bytes	sizeof (struct s)	
(type)	Type conversion	(float) total / n	
*	Multiplication	i * j	
/	Division	i / j	L-R
%	Modulus	i % j	
+	Addition	vals + i	L-R
-	Subtraction	x - 100	
<<	Left shift	byte << 4	L-R
>>	Right shift	i >> 2	
<	Less than	i < 100	
<=	Less than or eq to	i <= j	L-R
>	Greater than	i > 0	
>=	Greater or eq to	grade >= 90	
==	Equal to	result == 0	L-R
!=	Not equal to	c != EOF	
&	Bitwise AND	word & 077	L-R
^	Bitwise XOR	word1 ^ word2	L-R
|	Bitwise OR	word | bits	L-R
&&	Logical AND	j > 0 && j < 10	L-R
||	Logical OR	i > 80 || x_flag	L-R
? :	Conditional expr	(a > b) ? a : b	R-L
= *= /= %= += -= &= ^= |= <<= >>=	Assignment opers	count += 2	R-L
,	Comma operator	i = 10, j = 0	L-R

unary operators

NOTES: L-R means left-to-right, R-L right-to-left. Operators are listed in decreasing order of precedence. Ops in the same box have the same precedence. Associativity determines order of evaluation for ops with the same precedence (eg: a = b = c; is evaluated right-to-left as: a = (b = c)).

EXPRESSIONS

An expression is a variable name, function name, array name, constant, function call, array element reference, or structure member reference. Applying an operator (this can be an assignment operator) to one or more of these (where appropriate) is also an expression. Expressions may be parenthesized. An expression is a "constant expression" if each term is a constant.

ESC CHARS

\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\"	Double quote
\'	Single quote
\(CR)	Line continuation
\nnn	Octal character value

PREPROCESSOR STATEMENTS

STATEMENT	DESCRIPTION
`#define id text`	text will be substituted for id wherever it later appears in the program; if construct id(a1,a2,...) is used, args a1, a2, ... will be replaced where they appear in text by corresponding args of macro call

`#if expr`
`  ...`
`#endif`
	If constant expression expr is TRUE, statements up to #endif will be processed, otherwise they will not be.

`#if expr`
`  ...`
`#else`
`  ...`
`#endif`
	If constant expression expr is TRUE, statements up to #else will be processed, otherwise those between the #else and #endif will be processed

`#ifdef id`
`  ...`
`#endif`
	If id is defined (with #define or on the command line) statements up to #endif will be processed; otherwise they will not be; (optional #else)

`#ifndef id`
`  ...`
`#endif`
	If id has not been defined, statements up to #endif will be processed; (optional #else construct)

`#include "file"`
`  -or-`
`#include <file>`
	Inserts contents of file in program; double quotes mean look first in same directory as source prog, then in standard places; brackets mean only standard places

`#line n "file"`	Identifies subsequent lines of the prog as coming from file, beginning at line n; file is optional
`#undef id`	Remove definition of id

NOTES: Preprocessor statements can be continued over multiple lines provided each line to be continued ends with a backslash character (\). Statements can also be nested.

EXAMPLES:

```
#define  BUFSIZE     512
#define  max(a,b)    (((a) > (b)) ? (a) : (b))
#include <stdio.h>
```

typedef

typedef is used to assign a new name to a data type. To use it, make believe you're declaring a variable of that particular data type. Where you'd normally write the variable name, write the new data type name instead. In front of everything, place the keyword typedef. For example:

```
typedef struct   /* define type COMPLEX */
      {
        float real;
        float imaginary;
      } COMPLEX;

COMPLEX c1, c2, sum; /* declare vars */
```

CONSTANTS

TYPE	SYNTAX	EXAMPLES
char	single quotes	'a' '\n'
char string	double quotes	"hello" ""
double	(note 1)	
enumeration	(note 2)	red true
float	(note 3)	7.2 2.e-15 -1E9
hex integer	0X,0x	0xFF 0Xff 0xA000
int		17 -5 0
long int	l or L	251 100L (note 4)
octal int	0 (zero)	0777 0100

1. all float constants are treated as double
2. identifier previously declared for an enumerated type; value treated as int
3. decimal point and/or scientific notation
4. or any int too large for normal int

VARIABLE USAGE

STORAGE CLASS	DECLARED	CAN BE REFERENCED	INIT WITH	NOTES
static	outside fn	anywhere in file	const expr only	1
	inside fn/b	inside fn/b		
extern	outside fn	anywhere in file	cannot be init	2
	inside fn/b	inside fn/b		
auto	inside fn/b	inside fn/b	any expr	3
register	inside fn/b	inside fn/b	any expr	3,4
omitted	outside fn	anywhere in file or other files w/ext declaration	const expr only	5
	inside fn/b	(see auto)	(see auto)	6

NOTES: (fn/b means function or statement block)
1. init at start of prog execution; deflt is zero
2. var must be defined in only 1 place w/o extern
3. cannot init arrays & structures; var is init each time fn is called; no default value
4. reg assignment not guaranteed; restrict. types can be assigned to registers.
5. var can be decl. in only one place; initialized at start of prog execution; default is zero
6. defaults to auto

ARRAYS

A single-dimensional array aname of n elements of a specified type and with specified initial values (optional) is declared with:

```
type aname[n] = { val1, val2, ... };
```

If complete list of initial values is specified, n can be omitted. Only static or global arrays can be initialized. Char arrays can be init by a string of chars in double quotes. Valid subscripts of the array range from 0 through n-1. Multi dimensional arrays are declared with:

```
type aname[n1][n2]... = { init_list };
```

Values listed in the initialization list are assigned in 'dimension order' (i.e. as if last dimension were increasing first). Nested pairs of braces can be used to change this order if desired. Here are some examples:

```
/* array of char */
static char hisname[] = { "John Smith" };

/* array of char ptrs */
static char *days[7] =
  {"Sun","Mon","Tue","Wed","Thu","Fri","Sat"};

/* 3 x 2 array of ints */
int  matrix [3][2] = { { 10, 17 },
                       { -5, 0  },
                       { 11, 21 } };

/* array of struct complex */
struct complex sensor_data[100];
```

POINTERS

A variable name can be declared to be a pointer to a specified type by a statement of the form:

```
type *name;
```

EXAMPLES:

```
/* numptr points to floating number */
float *numptr;

/* pointer to struct complex */
struct complex *cp;

/* if the real part of the complex
   struct pointed to by cp is 0.0 ... */
if ( cp->real == 0.0 )

/* ptr to char; set equal to address of
   buf[25] (i.e. pointing to buf[25]) */
char *sptr = &buf[25];

/* store 'c' into loc ptd to by sptr */
*sptr = 'c';

/* set sptr pointing to next loc in buf */
++sptr;

/* ptr to fn returning int */
int (*fptr) ();
```

FUNCTIONS

Functions follow this format:

```
ret_type name (arg1,arg2,...)
   arg_declarations
   {
       local_var_declarations
       statement
       statement
       ...
       return value;
   }
```

Functions can be declared extern (default) or static. Static fns can be called only from the file in which they are defined. ret_type is the rtn type for the fn and can be void if the fn rtns no value or omitted if it rtns an int.

EXAMPLE:

```
/* fn to find the length
   of a character string */

int strlen (s)
   char *s;
   {
      int length = 0;

      while ( *s++ )
         ++length;
      return (length);
   }
```

To declare the type of value returned by a function you're calling, use a declaration of the form: ret_type name ();

STRUCTURES

A structure sname of specified members is declared with a statement of the form:

```
struct sname
   {
      member_declaration;
      member_declaration;
      ...
   } variable_list;
```

Each member_declaration is a type followed by one or more member names. An n-bit wide field mname is declared with a statement of the form ... type mname:n; ... If mname is omitted, n unnamed bits are reserved; if n is also zero, the next field is aligned on a word boundary. variable_list (optional) declares variables of that structure type. If sname is supplied, variables can also later be declared using the format:

```
struct sname variable_list;
```

EXAMPLE:

```
/* define complex struct */
struct complex
   {
      float real;
      float imaginary;
   };

static struct complex  c1 =
      { 5.0, 0.0 };
struct complex  c2, csum;

c2 = c1; /* assign c1 to c2 */
csum.real = c1.real + c2.real;
```

UNIONS

A union uname of members occupying the same area of memory is declared with a statement of the form:

```
union uname
   {
      member_declaration;
      member_declaration;
      ...
   } variable_list;
```

Each member_declaration is a type followed by one or more member names; variable_list (optional) declares variables of the particular union type. If uname is supplied, then variables can also later be declared using the format:

```
union uname variable_list;
```

NOTE: unions cannot be initialized.

ENUM DATA TYPES

An enumerated data type ename with values enum1, enum2, ... is declared with a statement of the form:

```
enum ename { enum1, enum2, ... }
                  variable_list;
```

The optional variable_list declares variables of the particular enum type. Each enumerated value is an identifier optionally followed by an equals sign and a constant expression. Sequential values starting at 0 are assigned to these values by the compiler unless the enum=value construct is used. If ename is supplied, then variables can also be declared later using the format

```
enum ename variable_list;
```

EXAMPLES:

```
/* define boolean */
enum boolean {true, false};
/* declare var & assign value */
enum boolean done = false;
/* test value */
if ( done == true )
```

MICRO LOGIC, POB 174
HACKENSACK, NJ 07602

C LANGUAGE

PROGRAMMER'S INSTANT REFERENCE CARD

MICRO CHART®

printf

printf is used to write data to standard output (normally, your terminal.) To write to a file, use fprintf; to 'write' data into a character array, use sprintf. The general format of a printf call is:

```
printf (format, arg1, arg2,...)
```

where format is a character string describing how arg1, arg2, ... are to be printed. The general format of an item in the format string is:

```
%[flags][size][.prec][l]type
```

flags:

- `-` left justify value (default is right justify)
- `+` precede value with a + or - sign
- `blank` precede pos value with a blank
- `#` precede octal value with 0, hex value with 0x (or 0X for type X); force display of decimal point for float value, and leave trailing zeroes for type g and G

size: is a number specifying the minimum size of the field; * instead of number means next arg to printf specifies the size

prec: is the minimum number of digits to display for ints; number of decimal places for e and f; max number of significant digits for g; max number of chars for s; * instead of number means next arg to printf specifies the precision

l: indicates a long int is being displayed; must be followed by d, o, u, x or X

type: specifies the type of value to be displayed per the following single character codes:

d	an int
u	an unsigned int
o	an int in octal format
x	an int in hex format, using a-f
X	an int in hex format, using A-F
f	a float (to 6 dec places by default)
e	a float in exponential format (to 6 decimal places by default)
E	same as e except display E before exponent instead of e
g	a float in f or e format, whichever takes less space w/o losing precision
G	a float in f or E format, whichever takes less space
c	a char
s	a null-terminated char string (null not required if precision is given)
%	an actual percent sign

NOTES: characters in the format string not preceded by % are literally printed; floating pt formats display both floats and doubles; integer formats can display chars, short ints or ints (or long ints if type is preceded by l). EXAMPLE:

```
i1 = 10;  i2 = 20;
printf ("%d + %d is %#x\n",
        i1, i2, i1 + i2);
```

Produces: 10 + 20 is 0x1e

UNIX cc COMMAND

Format: cc [options] files

OPTION	DESCRIPTION
-c	Don't link the program; forces creation of a .o file
-D id=text	Define id with associated text (exactly as if #define id text appeared in prog); if just -D id is specified, id is defined as 1
-E	Run preprocessor only
-f	Compile for machine w/o floating point hardware
-g	Generate more info for sdb use
-I dir	Search dir for include files
-lx	Link prog with lib x; -lm for math
-o file	Write executable object into file; a.out is default
-O	Optimize the code
-p	Compile for analysis with prof cmd
-S	Save assembler output in .s file

NOTE: Some of the above are actually preprocessor (cpp) and linker (ld) options. The standard C library libc is automatically linked with a program.

EXAMPLES: cc test.c Compiles test.c and places executable object into a.out.
cc -o test main.c proc.c Compiles main.c and proc.c and places executable object into test.
cc -O stats.c -lm Compiles stats.c, optimizes it, and links it with the math library (-lm must be placed after stats.c).
cc -DDEBUG x1.c x2.c Compiles x1.c, with defined name DEBUG, and links it with x2.o

THE lint COMMAND

lint can help you find bugs in your program due to nonportable use of the language, inconsistent use of variables, uninitialized variables, passing wrong argument types to functions, and so on.
Format: lint [options] files

OPT	USE TO PREVENT FLAGGING OF
-a	long values assigned to not-long vars
-b	break statements that can't be reached
-h	suspected bugs, waste, or style
-u	functions and external vars used but not defined, or defined and not used
-v	unused function arguments
-x	vars declared extern and never used
------	Other options ------
-lx	check prog against lint library llib-lx.ln; (-lm uses lint math lib)
-n	don't use standard or portable lint lib
-p	check portability to other C dialects
-D	see cc command
-I	see cc command

scanf

scanf is used to read data from standard input. To read data from a particular file, use fscanf. To 'read' data from a character array, use sscanf. The general format of a scanf call is:

```
scanf (format, arg1, arg2, ...)
```

where format is a character string describing the data to be read and arg1, arg2, ... point to where the read-in data are to be stored. The format of an item in the format string is:

```
%[*][size][lh]type
```

- `*` specifies that the field is to be skipped and not assigned (i.e., no corresponding ptr is supplied in the arg list)
- `size` a number giving the max size of the field
- `lh` is 'l' if value read is to be stored in a long int or double, or 'h' to store in short int
- `type` indicates the type of value being read:

USE	TO READ A	CORRESPONDING ARG IS PTR TO
d	decimal integer	int
u	unsigned decimal integer	unsigned int
o	octal integer	int
x	hexadecimal integer	int
e,f,g	floating point number	float
s	string of chars terminated by a white-space character	array of char
c	single character	char
[...]	string of chars terminated by any char not enclosed between the [and]; if first char in brackets is ^, then following chars are string terminators instead	array of char
%	percent sign	not assigned

NOTES: Any chars in format string not preceded by % will literally match chars on input (e.g. scanf ("value=%d", &ival); will match chars "value=" on input, followed by an integer which will be read and stored in ival. A blank space in format string matches zero or more blank spaces on input.

EXAMPLE: scanf ("%s %f %ld", text, &fval, &lval); will read a string of chars, storing it into character array ptd to by text; a floating value, storing it into fval; and a long int, storing it into lval.

COMMONLY USED FUNCTIONS

FUNCTION	INCLUDE FILE	DESCRIPTION /ERROR RETURN/
int abs (n)		absolute value of n
double acos (d)	m	arccosine of d /0/
char *asctime (*tm)	t	convert tm struct to string and rtn ptr to it
double asin (d)	m	arcsine of d /0/
double atan (d)	m	arctangent of d
double atan2 (d1,d2)	m	arctangent of d1/d2
double atof (s)		ascii to float conv /HUGE,0/
int atoi (s)		ascii to int conversion
long atol (s)		ascii to long conversion
char *calloc (u1,u2)		allocate space for u1 elements each u2 bytes large, and set to 0 /NULL/
double ceil (d)	m	smallest integer not < d
void clearerr (f)	s	reset error (incl. EOF) on file
long clock ()		CPU time (microsec) since first call to clock
double cos (d)	m	cosine of d (d in radians)
char *ctime (*1)	t	convert time ptd to by 1 to string and rtn ptr to it
void exit (n)		terminate execution, returning exit status n
double exp (d)	m	e to the d-th power /HUGE/
double fabs (d)	m	absolute value of d
int fclose (f)	s	close file /EOF/
int feof (f)	s	TRUE if end-of-file on f
int ferror (f)	s	TRUE if I/O error on f
int fflush (f)	s	force data write to f /EOF/
int fgetc (f)	s	read next char from f /EOF/
int fgets (s,n,f)	s	read n-1 chars from f unless newline or end of file reached; newline is stored in s if read /NULL/
int fileno (f)	s	integer file descriptor for f
double floor (d)	m	largest integer not > d
double fmod (d1,d2)	m	d1 modulo d2
FILE *fopen (s1,s2)	s	open file named s1, mode s2; "w"=write, "r"=read, "a"=append, ("w+", "r+", "a+" are update modes) /NULL/
int fprintf (f,s,...)	s	write args to f according to format s /< 0/
int fputc (c,f)	s	write c to f /EOF/
int fputs (s,f)	s	write s to f /EOF/
int fread (s,n1,n2,f)	s	read n2 data items from f into s; n1 is number bytes of each item /0/
void free (s)	s	free block of space ptd to by s /NULL/
FILE *freopen (s1,s2,f)	s	close f and open s1 with mode s2 (see fopen) /NULL/
int fscanf (f,s,...)	s	read args from f using format s; return is as for scanf
int fseek (f,l,n)	s	position file ptr; if n=0, l is offset from beginning; n=1, from current pos; n=2, from end of file /non-zero/
long ftell (f)	s	current offset from start of file
int fwrite (s,n1,n2,f)	s	write n2 data items to f from s; n1 is no. bytes of each item /NULL/
int getc (f)	s	read next char from f /EOF/
int getchar ()	s	read next char from stdin /EOF/
char *getenv (s)		rtn ptr to value of environment name s /NULL/
int getopt (argc,argv,s)		return next option letter in argc that matches a letter in s; sets optarg (char *) pointing to it, and optind (int) to index in argv of next arg to be processed; returns EOF when all args processed
char *gets (s)	s	read chars into s from stdin until newline or eof reached; newline not stored /NULL/
int getw (f)	s	read next word from f; use feof & ferror to check for error
struct tm *gmtime (*1)	t	convert time ptd to by l to GMT
int isalpha (c)	c	TRUE if c is alphabetic
int isalnum (c)	c	TRUE if c is alphanumeric
int isascii (c)	c	TRUE if c is less than 0200
int iscntrl (c)	c	TRUE if c is 0177 or < 040
int isdigit (c)	c	TRUE if c is 0-9
int isgraph (c)	c	TRUE if c is 041-0176
int isprint (c)	c	TRUE if c is a printable char (040-0176)
int ispunct (c)	c	TRUE if c is neither a control nor alphanumeric char
int isspace (c)	c	TRUE if c is space, tab, carriage return, newline, vertical tab or form feed
struct tm *localtime (*1)	t	convert time ptd to by l to local time
double log (d)	m	natural log of d /0/
double log10 (d)	m	log base 10 of d /0/
void longjmp (env,n)	j	restore environment from jmp_buf env; causes setjmp to return n if supplied or 1 if n=0
char *malloc (u)		allocate u bytes of storage and return ptr to it /NULL/
char *memchr (s,c,n)	n	rtn ptr in s of 1st incident of c, looking at n chars at most, or NULL if not found
int memcmp (s1,s2,n)	n	rtn < 0, = 0, > 0 if s1 is lexicographically < s2, = s2 or > s2, comparing up to n chars
char *memccpy (s1,s2,c,n)	n	copy s2 to s1 until c is copied or n chars are copied
char *memcpy (s1,s2,n)	n	copy n chars from s2 to s1
char *memset (s,c,n)	n	set n chars ptd to by s to value c
int mknod (s,i1,i2)		create file s, mode i1; i2 needed only for certain values of i1 /-1/
char *mktemp (s)		create temp file; s contains six trailing X's that mktemp replaces with file name
int pclose (f)	s	close a stream opened by popen /-1/
void perror (s)		write s followed by description of last error to stdout
FILE *popen (s1,s2)	s	execute command in s1; s2 is "r" to read its output; "w" to write to its input; rtns ptr to stream /NULL/
double pow (d1,d2)	m	d1 to the d2-th power /0,HUGE/
int printf (s,...)	s	write args to stdout per format s (see descr.) /< 0/
int putc (c,f)	s	write c to f /EOF/
int putchar (c)	s	write c to stdout /EOF/
int puts (s)	s	write s to stdout /EOF/
int putw (n,f)	s	write word n to f /EOF/
int rand ()		random number (see srand)
char *realloc (s,u)		change the size of block s to u and rtn ptr to it /NULL/
void rewind (f)	s	rewind f
int scanf (s,...)	s	read args from stdin per format s (see descr.); rtns number of values read or EOF
int setjmp (env)	j	save stack environment in jump_buf env; rtns 0 (see longjmp)
double sin (d)	m	sine of d (d in radians)
unsigned sleep (u)		suspend execution for u seconds
int sprintf (s1,s2,...)	s	write args to buffer s1 per format s2 /< 0/
double sqrt (d)	m	square root of d /0/
void srand (u)		reset random number generator
int sscanf (s1,s2,...)	s	read args from string s1 per format s2; rtn is as in scanf
char *strcat (s1,s2)	r	concatenate s2 to end of s1; rtns s1
char *strchr (s,c)	r	rtn ptr to 1st occurrence of c in s or NULL if not found
int strcmp (s1,s2)	r	compare s1 and s2; rtns < 0, = 0, > 0 if s1 lexicographically < s2, = s2, or > s2
char *strcpy (s1,s2)	r	copy s2 to s1; rtns s1
int strlen (s)	r	length of s (not incl. null)
char *strncat (s1,s2,n)	r	concatenate at most n chars from s2 to end of s1; rtns s1
int strncmp (s1,s2,n)	r	compare at most n chars of s1 to s2; rtn is as in strcmp
char *strncpy (s1,s2,n)	r	copy at most n chars from s2 to s1; rtns s1
char *strrchr (s,c)	r	rtn ptr to last occurrence of c in s or NULL if not found
long strtol (s,*s,n)		ascii to long conversion, base n; on rtn, *s (if not NULL) pts to the char in s that terminated the scan /0/
int system (s)	s	execute s as if it were typed at terminal; rtns exit status /-1/
double tan (d)	m	tangent of d (radians) /HUGE/
char *tempnam (s1,s2)	s	create temporary file name in directory s1, with prefix chars s2 /NULL/
long time (*1)		returns time & date in seconds; if l is non-zero, time is stored in loc ptd to by l; convert time rtnd with ctime, localtime or gmtime
FILE *tmpfile ()	s	create temporary file, open for update, and rtn ptr to it; file is removed when prog finishes
char *tmpnam (s)	s	generate temporary file name; place result in s if s non-null, else rtn ptr to name
int toascii (c)	c	convert c to ascii
int tolower (c)	c	convert c to lowercase
int toupper (c)	c	convert c to uppercase
int ungetc (c,f)	s	insert c back into file f (as if c wasn't read) /EOF/
int unlink (s)		remove file s /-1/

NOTES:

Function argument types: c--char, n--int, u--unsigned int, l--long int, d--double, f--ptr to FILE, s--ptr to char

char and short int are converted to int when passed to functions; float is converted to double

Include files are abbreviated as follows: c--ctype.h, j--setjmp.h, m--math.h, n--memory.h, r--string.h, s--stdio.h, t--time.h

Value between slashes is returned if function detects an error; global int errno also gets set to specific error number.

Function descriptions based on UNIX System V

CMD LINE ARGS

Arguments typed in on the command line when a program is executed are passed to the program through argc and argv. argc is a count of the number of arguments, and is at least 1; argv is an array of character pointers that point to each argument. argv[0] points to the name of the program executed. Use sscanf to convert arguments stored in argv to other data types. For example:

```
check phone 35.79
```

starts execution (under UNIX) of a program called check; with

```
argc = 3
argv[0] = "check"
argv[1] = "phone"
argv[2] = "35.79"
```

To convert number in argv[2], use sscanf. EXAMPLE:

```
main (argc, argv)
   int argc;
   char *argv[];
   {
      float  amount;
      ...
      sscanf (argv[2],
          "%f", &amount);
      ...
   }
```

UNIX TOOLS

TOOL	DESCRIPTION
adb	debugger
ar	library archiver
cb	formats programs
cflow	ext references
ctrace	traces execution
cxref	X-ref listing
lint	checks progs for possible bugs and non-portable language usage
make	recreates program systems based on specified file dependencies
prof	displays performance statistics
SCCS	maintains large program systems
sdb	symbolic debugger

REMINDERS

1. Array indices start at 0 and go to number of elements minus 1.
2. Use "==" (not "=") for testing equality.
3. Use "->" for structure pointers and "." for structures.
4. Args to scanf must be ptrs (place "&" in front of non-ptrs).
5. 'x' is of type char; "x" is of type ptr to char.
6. If cp is ptr to char, and c is array of char, then cp="hello" is okay, but c="hello" isn't.
7. In x[i]=++i, it's not defined whether left or right side will be evaluated first.
8. In switch, omitting break causes fall-through to next case.
9. Return type for non-int fns must be declared unless fn previously defined.
10. Fn arg types must be consistent with type declared (e.g. sqrt (2) will produce the wrong result).
11. In ++p, value of expr is that of p after it's incremented; in p++, value is that of p before it's incremented.

ASCII

CHR	OC	HX
nul	0	0
soh	1	1
stx	2	2
etx	3	3
eot	4	4
enq	5	5
ack	6	6
bel	7	7
bs	10	8
ht	11	9
nl	12	A
vt	13	B
ff	14	C
cr	15	D
so	16	E
si	17	F
dle	20	10
dc1	21	11
dc2	22	12
dc3	23	13
dc4	24	14
nak	25	15
syn	26	16
etb	27	17
can	30	18
em	31	19
sub	32	1A
esc	33	1B
fs	34	1C
gs	35	1D
rs	36	1E
us	37	1F
sp	40	20
!	41	21
"	42	22
#	43	23
$	44	24
%	45	25
&	46	26
'	47	27
(	50	28
)	51	29
*	52	2A
+	53	2B
,	54	2C
-	55	2D
.	56	2E
/	57	2F
0	60	30
1	61	31
2	62	32
3	63	33
4	64	34
5	65	35
6	66	36
7	67	37
8	70	38
9	71	39
:	72	3A
;	73	3B
<	74	3C
=	75	3D
>	76	3E
?	77	3F
@	100	40
A	101	41
B	102	42
C	103	43
D	104	44
E	105	45
F	106	46
G	107	47
H	110	48
I	111	49
J	112	4A
K	113	4B
L	114	4C
M	115	4D
N	116	4E
O	117	4F
P	120	50
Q	121	51
R	122	52
S	123	53
T	124	54
U	125	55
V	126	56
W	127	57
X	130	58
Y	131	59
Z	132	5A
[	133	5B
\	134	5C
]	135	5D
^	136	5E
_	137	5F
`	140	60
a	141	61
b	142	62
c	143	63
d	144	64
e	145	65
f	146	66
g	147	67
h	150	68
i	151	69
j	152	6A
k	153	6B
l	154	6C
m	155	6D
n	156	6E
o	157	6F
p	160	70
q	161	71
r	162	72
s	163	73
t	164	74
u	165	75
v	166	76
w	167	77
x	170	78
y	171	79
z	172	7A
{	173	7B
¦	174	7C
}	175	7D
~	176	7E
del	177	7F

INTENTIONALLY BLANK

DO NOT PLACE ON HOT SURFACE

By: Stephen G. Kochan
Author of "Programming in C"
(Hayden Book Company)

111B