

EE160 C Programming Fundamentals

Philip Robbins - [REDACTED]



Contents

Philip Robbins
Spring 2002 - EE160 C Programming Fundamentals
First Draft 12.21.2001

Section 1 vi: standard commands

Section 2 EE160: Review of C Programming Fundamentals

Commands within the vi editor

<i>Esc</i>	- Command mode
<i>i</i>	- Input mode at cursor position
<i>a</i>	- Input mode after cursor position
<i>I</i>	- Insert at beginning of line
<i>A</i>	- Append at end of line
<i>o</i>	- Input mode in line below cursor
<i>O</i>	- Open line above
<i>h</i>	- Move cursor back one character
<i>j</i>	- Move cursor down one line
<i>k</i>	- Move cursor up one line
<i>l</i>	- Move cursor forward one character
<i>w</i>	- Forward a word
<i>b</i>	- Back a word
<i>YY</i>	- Cut line
<i>3YY</i>	- Cut 3 lines
<i>Yl</i>	- Cut letter
<i>yw</i>	- Cut word
<i>Y </i>	- Cut paragraph
<i>Y)</i>	- Cut sentence
<i>P</i>	- Paste before cursor
<i>P</i>	- Paste after cursor
<i>H</i>	- Go to top of screen
<i>L</i>	- Go to bottom of screen
<i>M</i>	- Go to middle of screen
<i>[</i>	- Beginning of paragraph
<i>]</i>	- End of paragraph
<i>(</i>	- Beginning of sentence
<i>)</i>	- End of sentence
<i>e</i>	- Move to end of word
<i>Ctrl Shift 6</i>	- Move to first non-white character of line
<i>J</i>	- Join lines
<i>dd</i>	- Deletes current line
<i>3dd</i>	- Delete 3 lines
<i>D</i>	- Delete rest of line
<i>D^</i>	- Delete from beginning of the line to current cursor position
<i>Dw</i>	- Delete word
<i>D\$</i>	- Delete from current cursor position to end of line
<i>x</i>	- Delete character after cursor (delete key)
<i>X</i>	- Delete the character before cursor (backspace key)
<i>0</i>	- Go to beginning of line
<i>\$</i>	- Go to end of line
<i>:\$</i>	- Go to end of file
<i>ctrl B</i>	- Page up
<i>ctrl F</i>	- Page down
<i>:#</i>	- Place cursor at line represented by #
<i>:-r filename</i>	- Imports file to vi
<i>u</i>	- Undo the previous command
<i>U</i>	- Undo all changes made to current line
<i>:w</i>	- Save file
<i>:q!</i>	- Force quit without save
<i>:q</i>	- Quits without changes made
<i>ZZ</i>	- Save and exit
<i>/text string</i>	- Search
<i>?text string</i>	- Search backwards
<i>;</i>	- Repeat last find
<i>,</i>	- Repeat the Inverse of the last find
<i>fx</i>	- Find the next x on the line
<i>Fx</i>	- Find the previous x on the line
<i>!e filename</i>	- Switch to new file without quitting.
<i>.</i>	- Last command executed
<i>5Y</i>	- Yank 5 lines (cuts to clipboard)

EE160 C Review of Programming Fundamentals

When starting to write a program the starting layout should look like:

```
main()
{
}
```

Note: C is case sensitive.

```
/* comment */           To insert a comment within the script.
;                         End statement, start new line
```

printf() -the output function.

`printf("argument\n");` will continuously print the given argument across the screen until it hits the newline.

```
printf("      \n");      To print an expression (start newline)
printf("      ");        To print an expression (without newline)
```

```
\n      newline - nonprinting character
\n\n\n multiple newlines \n 's can be placed within the printf() function to
skip lines.
```

printf() conversion characters

```
%c      as a character
%d      as a decimal integer
%e      as a floating point number in scientific notation
%f      as a floating point number
%g      in %e - format / %f - format (whichever is shorter)
%s      as a string
```

examples:

<u>code</u>	<u>output</u>
<code>printf("this is an example.\n");</code>	this is an example.
<code>printf("this\n");</code> <code>printf("is an\n");</code> <code>printf("example.\n");</code>	this is an example.
<code>printf("this is ");</code> <code>printf("an example.\n");</code>	this is an example.
<code>printf("this\nis an\nexample.\n");</code>	this is an example.

examples:

```
code
printf("Is this example: %s %d %f %c%c%c%c, "one", 2, 3.33, 'f', 'o', 'u', 'r');
```

```
output
Is this example: one 2 3.33 four
```

```
code
printf("%s%s\n",
"This example shows a format for printing",
"very long lines of text on the screen.");
```

```
output
This example shows a format for printing very long lines of text on the screen.
```

Field Width - the distance between the field of arguments can be adjusted by placing an integer *m* between the % and the conversion character.

Precision - for floating values, precision can be controlled by specifying *n*: %*m*.*n*f

```
printf("%c%3c%7c\n", 'A', "B", "C")      m represents spacing: A B C
printf("%.1f %.2f %.3f\n", 1.0, 2.0, 3.0) n represents precision: 1.0 2.00 3.000
```

Declarations - In C all variables must be declared before they are used in expressions and statements.

```
main()
{
    declarations
    statements
}
```

Data Types

float <i>expression</i>	Used to declare floating values
int <i>expression</i>	Used to declare integer values
char <i>expression</i>	Used to declare a character

Binary Arithmetic Operators

a + b	sum of a and b
a - b	difference of a and b
a * b	product of a and b
a / b	division of a and b
a % b	modulus operator: a mod b = remainder of a / b

scanf() -the input function.

scanf("%X", &expression); will prompt for X converted character and evaluate to the address of the expression. & is called the address operator.

scanf() conversion characters

%c to a character
%d to a decimal integer
%f to a floating point number
%lf to a floating point number (long)
%s to a string

example:

```
main()
{
float input1, input2;
char input3;

printf("\n\n%s", "input two numbers and a character: ");
scanf("%f%f%c", &input1, &input2, &input3);
printf("%s%f%f%c\n", "this is what you entered", input1, input2, input3);
}
```

Operators:

< less than
> greater than
== equal to
>= greater than or equal
<= less than or equal
!= not equal

Other condtions:

|| logical OR
&& logical AND

Increment and Decrement Operators:

++ and -- are unary operators applied to variables only.

example:

++i is the same as: i + 1
i++ is the same as: i + 1
--cnt is the same as: cnt - 1
cnt-- is the same as: cnt - 1

Nonprinting \ Hard-to-print characters:

\0 null
\b backspace
\t tab
\n newline
\f formfeed
\r carriage return
\" double quote
' single quote
\ backslash

while() -the repetitive action statement

The while command is a function that will execute as long as the condition is meet. If not the program will terminate.

1. Condition Initialization
2. Condition Test
3. Loop Body
4. Condition Update

A while loop has the general forms:

```
while(expression)
    statement

while(expression)
{compound statement;}
```

The statement can be single or a group (compound statement) enclosed in braces { }. In the

operation:

1. The expression is evaluated.
If the expression is true (nonzero) then the statement is executed.
Control is passed back to the beginning of the while loop.
 2. If the expression is false (zero) then the statement is not executed.
-

example: an infinite loop

```
main()
{
while(1){}
}
```

EOF -End of File

while(flag != EOF) will break loop if Ctrl D is pressed *(EOF - end of file)
for dos users Ctrl Z is pressed.

note: define and use "flag" expression and set it equal to **scanf()**

example:

```
flag = scanf("%d", &n);
while(flag != EOF)
    ...
```

?? Macro definition: `#define <symbol_name> <substitution_string>`

examples:

```
#define PI 3.14159
#define TRUE 1
#define FALSE 0
```

?? Include directives: `#include <filename>`
 `#include "filename"`

Note: All #include directives must be typed before the main () function.

examples: The C Standard Library

<code>#include <stdio.h></code>	Input and Output Routines
<code>#include <ctype.h></code>	Character class tests
<code>#include <string.h></code>	String Functions
<code>#include <stdlib.h></code>	Utility Functions
<code>#include <math.h></code>	Mathematical Functions ✓
<code>#include <limits.h></code>	Limits
<code>#include <float.h></code>	Limits
<code>#include <assert.h></code>	Diagnostics
<code>#include <errno.h></code>	Error Numbers
<code>#include <signal.h></code>	Signals
<code>#include <time.h></code>	Date and Time Functions
<code>#include <stdarg.h></code>	Variable Argument Lists
<code>#include <setjmp.h></code>	Non-local Jumps
<code>#include "custom.h"</code>	use " " to include a custom header file (custom.h)

When you use the math.h library in unix, you need to compile your program with the command : cc filename.ext -lm

if() & if()else() -The if and the if-else statements

Some general forms of the if statement and if-else statement:

```
if(expression) statement1;

if(expression) {compound statements1;}
```

```
if(expression) statement1;
else statement2;
```

```
if(expression)
{  compound statements1;
}
else
{  compound statements2;
}
```

operation: the if statement

1. The expression is evaluated.
If the expression is true (nonzero) then the statement is executed.
Control is passed to the next statement in the code (unlike the while() statement where control is passed back to while() statement.)
2. If the expression is false (zero) then the statement is not executed.
Control is passed to the next statement in the code.

operation: the if-else statement

1. The expression is evaluated.
If the expression is true (nonzero) then the statement(s)1 is executed.
Statement(s)2 is skipped.
2. If the expression is false (zero) then the statement(s)1 is not executed.
Statement(s)2 is executed.

The dangling else problem: an else attaches to the nearest if

Techniques:

```
if( )
if( ) else
if( ) if( ) if( )
if( ) else if( ) else if( ) else if( )
```

for() -The for statement

The for statement is closely related to the while statement.

General Construction:

```
for(expression1; expression2; expression3)
statement;
next statement;
```

Equivalent:

```
expression1;
while(expression2){
    statement
    expression3;
}
next statement;
```

operation: the for statement

1. Expression1 is evaluated. (Expression1 is typically an initialization of a variable).
Expression2 is evaluated.
If expression2 is true (nonzero) then the statement is executed.
Expression3 is evaluated.
Control is passed back to the start of the for loop.
Evaluation of expression1 is skipped.
2. If expression2 is false (zero) then the for loop is broken.
Control is passed to the next statement.

do() -the while variant

Unlike the while statement the do statement makes its test at the bottom of the loop. The do statement will continue or break depending on the while condition.

General Construction:

```
do{statement;
}
while(expression);
next statement;
```

operation: the do statement

1. The first statement/compound statements is executed.
The expression within the while statement is evaluated.
If the expression is true (nonzero) then control is passed back to the start of the do statement. (process repeats itself)
2. If the expression is false (zero) then the do statement is broken and control passes to the next statement.

Function Invocation

If you want to create a function you will need to declare it in the header above main() and define it after the closed } of main(). Note the placements of ;

i.e.

```
Samplefunction(float expression1, expression2);
main()
{ samplefunction (send 1, send 2);
}
Samplefunction(float expression1, expression2)
{
float value;           receive 1, receiver
return value;
}
```

Remember: That each function is restricted and unique to the variables and expressions defined for that function. (i.e. the variable "x" for a function A will not be the same for a function B)

return() -the return statement

The return statement is used to pass control back to the calling environment.

```
return (expression);
```

if a function does not return anything then void can be put in front of the function title. example: void main()

rand() & srand() -Random Number Generators

The function rand() from the standard library is used to generate a pseudo-random number.

```
int rand(void)
```

example:

```
r = rand();
printf("\n%d\n", r);
...
```

example: rand() % 2 (special case - even or odd number 0 or 1.)

```
r = rand() % 2
```

```
void srand(seed)
```

where the seed is equal to any integer.

The Data Type char

declaration: char c;

In the functions printf() and scanf() a %c is used to designate the character format.

examples: character processing

```
char c;
char c1 = 'A', c2 = 'B', c3 = 'C';

printf("%c%c%c", 'A', 'B', 'C');      /* ABC is printed */
printf("%d", 'a');                    /* 97 is printed */
printf("%c", 97);                    /* a is printed */
printf("%c", 'a');                    /* a is printed */
```

The American Standard Code for Information Interchange (ASCII) Character Set

000	CTRL - @	043	+	086	V
001	CTRL - A	044	,	087	W
002	CTRL - B	045	-	088	X
003	CTRL - C	046	.	089	Y
004	CTRL - D	047	/	090	Z
005	CTRL - E	048	0	091	[
006	CTRL - F	049	1	092	\
007	CTRL - G	050	2	093]
008	CTRL - H	051	3	094	^
009	CTRL - I	052	4	095	
010	CTRL - J	053	5	096	`
011	CTRL - K	054	6	097	a
012	CTRL - L	055	7	098	b
013	CTRL - M	056	8	099	c
014	CTRL - N	057	9	100	d
015	CTRL - O	058	:	101	e
016	CTRL - P	059	;	102	f
017	CTRL - Q	060	<	103	g
018	CTRL - R	061	=	104	h
019	CTRL - S	062	>	105	i
020	CTRL - T	063	?	106	j
021	CTRL - U	064	@	107	k
022	CTRL - V	065	A	108	l
023	CTRL - W	066	B	109	m
024	CTRL - X	067	C	110	n
025	CTRL - Y	068	D	111	o
026	CTRL - Z	069	E	112	p
027	CTRL - [070	F	113	q
028	CTRL - \	071	G	114	r
029	CTRL -]	072	H	115	s
030	CTRL - ^	073	I	116	t
031	CTRL - _	074	J	117	u
032	[space]	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	\$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	
039	'	082	R	125	}
040	(083	S	126	~
041)	084	T	127	[DEL]
042	*	085	U		

getchar() & putchar() -Character I/O

To read a character from the keyboard: `getchar()` is used.
To write a character to the screen: `putchar()` is used.

examples:

```
int c;
c = getchar();
...
char c;
c = getchar();
...
putchar('c');
putchar('a');
putchar('t');
putchar('\n');
```

```
#include <stdio.h>

main()
{
int c;
while((c = getchar()) != EOF){
putchar(c);
}
}
```

Macros in ctype.h

Character Test Macros - returns an int value if: true (nonzero) or false (zero).

Macro	True (Nonzero) value returned if:
<code>isalpha(c)</code>	c is a letter
<code>isupper(c)</code>	c is an uppercase letter
<code>islower(c)</code>	c is a lowercase letter
<code>isdigit(c)</code>	c is a digit
<code>isxdigit(c)</code>	c is a hexadecimal digit
<code>isspace(c)</code>	c is a white space character
<code>isalnum(c)</code>	c is a letter or digit
<code>ispunct(c)</code>	c is a punctuation character
<code>isprint(c)</code>	c is a printable character
<code>iscntrl(c)</code>	c is a control character
<code>isascii(c)</code>	c is an ASCII code

Conversion Macros

Macro	Effect:
<code>toupper(c)</code>	changes c from lowercase to uppercase
<code>tolower(c)</code>	changes c from uppercase to lowercase
<code>toascii(c)</code>	changes c to ASCII code

Fundamental Data Types:

char	short	int	long
unsigned char	unsigned short	unsigned	unsigned long
float	double		

typedef DATATYPE SUBSTITUTE;

sizeof() -Object size operator

`sizeof(object)`

The sizeof() operator returns an integer that represents the number of bytes needed to store the object in memory.

example:

```
int main()
{
printf("\nchar:%3d byte \n", sizeof(char));
printf("\nchar:%3d byte \n", sizeof(short));
printf("\nchar:%3d byte \n", sizeof(int));
printf("\nchar:%3d byte \n", sizeof(unsigned));
printf("\nchar:%3d byte \n", sizeof(long));
printf("\nchar:%3d byte \n", sizeof(float));
printf("\nchar:%3d byte \n", sizeof(double));
}
```

Mathematical Functions:

function	prototype	return
sqrt()	double sqrt(double x);	square root of x
pow()	double pow(double x, double y);	x^y
exp()	double exp(double x);	exponent of x = e^x
log()	double log(double x);	natural logarithm of x
sin()	double sin(double x);	sine of x
cos()	double cos(double x);	cosine of x
tan()	double tan(double x);	tangent of x

switch() -The switch statement

switch(expression) statement

typically the statement is a compound statement with case labels:

```
switch(expression0) {
case expression1: statement1
case expression2: statement2
case expression3: statement3
case expression4: statement4
...
default: statementX
```

In a switch statement control passes to the statement associated with the matching case label, and continues from there to all subsequent statements in the compound statement.

break & continue

The break statement causes an exit from the innermost enclosing loop or switch statement.

syntax: `break;`

The continue statement causes the current iteration of a loop to stop (but returns to the loop condition test to determine if the loop body is to be executed again otherwise) it will cause the next iteration of the loop to begin.

syntax: `continue;`

Nested Flow of Control:

example:

```
if(expression1) statement1;
else if(expression2) statement2;
else if(expression3) statement3;
else if(expression4) statement4;
else if(expression5) statement5;
...
else default statement;
next statement;
```

operation:

1. `expression1` is evaluated first.
If `expression1` is true (nonzero) then its corresponding `statement1` is executed. Skipping the other expressions & statements, control will pass to next `statement`.
2. If `expression1` is false (zero) then `statement1` is skipped and the next expression is evaluated until an expression is true.
3. If none of the expressions are true then the `default statement` is executed and control is passed to `next statement`. (note: `default statement` can be removed.)

goto

The `goto` statement is an unconditional branch to an arbitrary labeled statement in the function and is the most primitive method of interrupting ordinary control flow.

syntax: `label:`
 `goto label;`

`goto line expression:` in this case label is the line it will jump to.

```
example: infinite loop

main()
{
int i = 0;

loop: printf("%d\n", i++);
goto loop;
}
```

The Storage Classes

auto - The principal storage class. Variables declared inside of function bodies are automatic by default.

extern - All functions and variables declared outside of function bodies have external storage class. it is a method to transmit information across blocks and functions. The keyword **extern** is used to tell the system to look for a variable externally (i.e. in another file).

static - storing under the class **static** allows a local variable to retain its previous value upon reentry into a block. (in contrast to ordinary automatic variables which lose their value upon block exit.)

register - The use of storage class **register** is an attempt to improve execution speed. (note: the register declaration is taken as advice to a compiler.)

Pointers

Pointers are used in programs to access memory and manipulate addresses.

example declaration: **int *p;**

example assignments: **p = &i;**
 p = 0;

example: using pointers with **scanf()**:

int i, *p;

p = &i;
scanf("%d", p);

If **p** is a pointer, then *p is the value of the variable that **p** points to. The direct value of **p** is a memory location, whereas ***p** is the indirect value of **p**, the value at the memory location stored in **p**.

example: ***** is the inverse operator to **&**.

float x, y, *p;

p = &x;
y = *p;

equivalence: **y = *&x** OR **y = x**

The indirection operator * takes the value of p to be a memory location and returns the value stored in this location.

example: program

```
int main()
{
    int i = 9, *p;
    char x = 'A', *y;

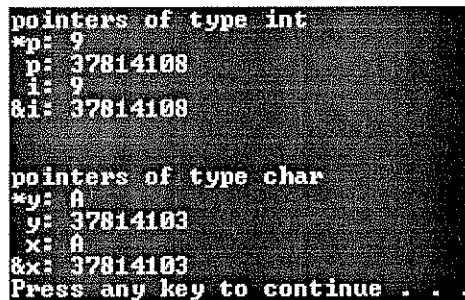
    y = &x;
    p = &i;

    printf("%s\n", "pointers of type int");
    printf("**p: %d\n", *p);
    printf(" p: %d\n", p);
    printf(" i: %d\n", i);
    printf("&i: %d\n", &i);

    printf("\n\n%s\n", "pointers of type char");
    printf("**y: %c\n", *y);
    printf(" y: %d\n", y);
    printf(" x: %c\n", x);
    printf("&x: %d\n", &x);

    system("PAUSE");
}
```

SCREEN OUTPUT:



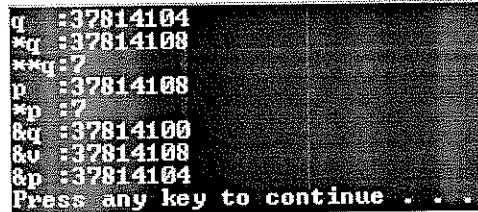
```
pointers of type int
*p: 9
p: 37814108
i: 9
&i: 37814108

pointers of type char
*y: A
y: 37814103
x: A
&x: 37814103
Press any key to continue . . .
```

example: program analysis

```
int main()
{
int v = 7, *p = &v, **q = &p;
printf("q :%d\n*q :%d\n**q:%d\np :%d\n*p :%d\n&q :%d\n&v :%d\n&p :%d\n",
q, *q, **q, p, *p, &q, &v, &p);
system("PAUSE");
}
```

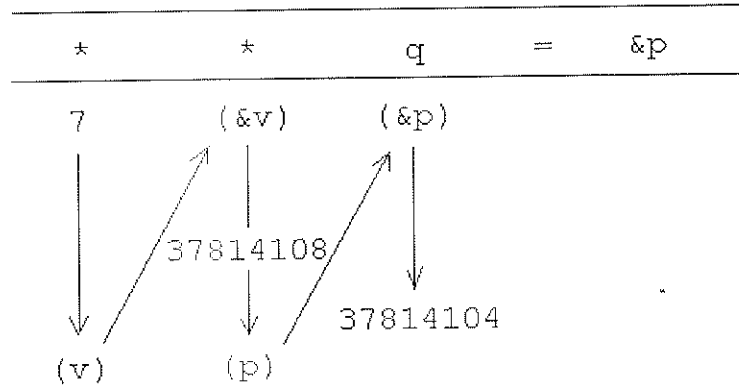
SCREEN OUTPUT:



REASONING:

assuming that $\&q = 37814100$, $\&v = 37814108$, $\&p = 37814104$.
we are aware that $v = 7$ and given that $\&v = 37814108$
given int: $*p = \&v = 37814108$ or $p = 37814108$ (remember: think about what is given
 v and $\&v$; what it is set equal to ; think about the definition for pointers).
By definition $*p$ will be the stored value at the memory location $p = 37814108$. This
is the same memory location being allocated for $v = 7$. Therefore $*p = 7$.
given int: $**q = \&p$

```
**q = &p
**q = *p = 7
```



Arrays - The Compound Data Type

Arrays allow a collection of data of the same type to be grouped into a single object.

Declaration: <type-specifier><identifier>[size]

Sample Initializations: static float x[] = {1, 2, 3, 4}
 int a[4] = {1, 2, 3, 4}
 int a[50] = {1, 2}

example:

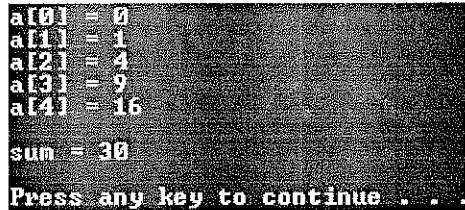
```
#define SIZE 5

int main()
{
int a[SIZE];                /* space for a[0], ..., a[4] allocated */
int i, sum = 0;

for(i = 0; i < SIZE; ++i)
a[i] = i * i;
for(i = 0; i < SIZE; ++i)
printf("a[%d] = %d\n", i, a[i]);
for(i = 0; i < SIZE; ++i)
sum += a[i];
printf("\nsum = %d\n\n", sum);

system("PAUSE");
}
```

SCREEN OUTPUT:



```
a[0] = 0
a[1] = 1
a[2] = 4
a[3] = 9
a[4] = 16
sum = 30
Press any key to continue . . .
```

multi-dimensional arrays

one-dimensional array: <type-specifier><identifier>[size]
two-dimensional array: <type-specifier><identifier>[size] [size]
three-dimensional array: <type-specifier><identifier>[size] [size] [size]

example: int b[3][5];

	col 1	col 2	col 3	col 4	col 5
row 1	b[0][0]	b[0][1]	b[0][2]	b[0][3]	b[0][4]
row 2	b[1][0]	b[1][1]	b[1][2]	b[1][3]	b[1][4]
row 3	b[2][0]	b[2][1]	b[2][2]	b[2][3]	b[2][4]

example:

```
#define M 3          /* row */
#define N 4          /* columns */

int main()
{
    int a[M][N], i, j, sum = 0;

    for(i = 0; i < M; ++i)
        for(j = 0; j < N; ++j)
            a[i][j] = i + j;

    for(i = 0; i < M; ++i){
        for(j = 0; j < N; ++j)
            printf("a[%d][%d] = %d   ", i, j, a[i][j]);
        printf("\n");
    }

    for(i = 0; i < M; ++i)
        for(j = 0; j < N; ++j)
            sum += a[i][j];

    printf("\nsum = %d\n\n", sum);

    system("PAUSE");
}
```

SCREEN OUTPUT:

```
a[0][0] = 0   a[0][1] = 1   a[0][2] = 2   a[0][3] = 3
a[1][0] = 1   a[1][1] = 2   a[1][2] = 3   a[1][3] = 4
a[2][0] = 2   a[2][1] = 3   a[2][2] = 4   a[2][3] = 5

sum = 30
Press any key to continue . . .
```

Strings

A string is a one-dimensional array of type char.
The null character \0 is used to delimit a string.
Header file used: <string.h>

Declaration: char <identifier>[size]

A function call such as scanf("%s", w) can be used to read a sequence of nonwhite space characters into the string w. scanf() automatically ends the string with the null character.

char *s[] = "test"; is equivalent to char *s[] = {'t', 'e', 's', 't', '\0'};

example:

```
#define max 100
int main()
{
    char w[max];

    w[0] = 'T';
    w[1] = 'E';
    w[2] = 'S';
    w[3] = 'T';
    w[4] = '\0';
    printf("%s\n\n", w);

    system("PAUSE");
}
```

SCREEN OUTPUT:

```
TEST
Press any key to continue . . .
```

example:

```
#include<stdlib.h>
#include<stdio.h>

int main()
{
    static char *s[]={"Put your message here."};
    printf("\n%s\n\n", *s);
}
```

SCREEN OUTPUT:

```
Put your message here.
Press any key to continue . . .
```

example:

```
int main()
{
    char *p = "This is a test.";
    printf("%s\n\n", p);

    system("PAUSE");
}
```

SCREEN OUTPUT:

```
This is a test.
Press any key to continue . . .
```

`strcpy(array, "Hello"); // need to include <string.h>`

example:

```
int main()
{
static char s[] = "Another Example...";
printf("%s\n\n", s);

system("PAUSE");
}
```

SCREEN OUTPUT:

```
Another Example...
Press any key to continue . . .
```

example:

```
int main()
{

static char s[] = {'c', 'h', 'a', 'r', '\0'};
printf("%s\n\n", s);

system("PAUSE");
}
```

SCREEN OUTPUT:

```
char
Press any key to continue . . .
```

example: reading a line of characters typed by a user into a string

```
#define max 100
int main()
{

char c, line[max];
int i;

for(i = 0; (c = getchar()) != '\n'; ++i)
line[i] = c;
line[i] = '\0';

printf("\n\noutput: %s\n\n", line);

system("PAUSE");
}
```

SCREEN OUTPUT:

```
This is a input test.
output: this is a input test.
Press any key to continue . . .
```


Three Standard Files in `stdio.h`

<code>stdin</code>	standard input file	connected to the keyboard
<code>stdout</code>	standard output file	connected to the screen
<code>stderr</code>	standard error file	connected to the screen

The function, `fscanf()` read formatted input from a file and `fprintf()` writes formatted output to a file.

`fprintf(stdout, . . .);` is equivalent to `printf(. . .);`

Input / Output and Files

`fopen(file_name, file_mode)`

The file open function associates a physical file with a file buffer or stream and returns a `FILE` pointer that is used to access the file.

prototype: `FILE * fopen(char * fname, char * mode);`

The pointer value `NULL` is returned if the `file_name` cannot be accessed. The file modes are "r" - read, "w" - write, and "a" - append.

example:

```
#include <stdio.h>
main()
{
  FILE *fin;

  fin = fopen("test.doc", "r");
  if(!fin){
    printf("Unable to open input file: test.doc\n");
    exit(0);
  }

  fclose(fin);
}
```

*ch = getc(inp)
putc(ch, stdout)*

`fclose(file_pointer)`

Empties buffers and breaks all connections to the file pointed to by `file_pointer`. EOF is returned if `file_pointer` is not associated with a file.

`getc(file_pointer)`

Retrieves the next character from the file pointed to by `file_pointer`. EOF returned if there is an error. `getchar()` is equivalent to `getc(stdin)`.

`fgetc(file_pointer)`

Acts similarly to `getc()`, but it is a function, not a macro.

`ungetc(c, file_pointer)`

Pushes the character value of `c` back onto the file pointed to by `file_pointer` and returns the int value of `c`. If the file is buffered and one or more characters have been read, then at least one character can be pushed back.

putc(c, file_pointer)

Places the character value of `c` in the output file pointed to by `file_pointer`. It returns the int value of the character written.

fputc(c, file_pointer)

Acts similarly to `putc(c, file_pointer)`, but it is a function, not a macro.

gets(s)

Reads a string into `s` from `stdin`. The argument `s` is a pointer to char (a string). Characters are read into `s` until a newline character is read, at which point the newline character is changed to a null character that is used to terminate `s`. The value of `s` (pointer to char) is returned.

fgets(s, n, file_pointer)

Reads a string into `s` from the file pointed to by `file_pointer`. Characters are read from the file and placed in `s` until `n - 1` characters have been read, or a newline character is read, whichever comes first. Unlike `gets()`, if a newline character is read, it is placed in `s`. In both cases `s` is terminated with a null character. The int value `n` is the maximum number of characters, including the null character, that can be read into `s`. The value of `s` (pointer to char) is returned.

system(command)

Provides a connection to the operating system. The string (pointer to char) `command` is passed to the operating system, executed as a command, and printed on the screen (`stdout`).

exit(status)

Terminates a program when it is called. All buffers are flushed and all files are closed. Calling process assumes that the program ran properly if `status` has value 0; a nonzero value indicates that it did not run properly.

Alphabetic Listing of System Routines

Common ANSI C system routines taken from EE160 Programming in C (C)1993 by B. Kinariwala & T. Dolbry

abort #include <stdlib.h>
 Prototype: void abort(void);
 Description: Causes the program to terminate abnormally.

abs #include <stdlib.h>
 Prototype: int abs(int n);
 Returns: absolute value of n

acos #include <math.h>
 Prototype: double acos(double x);
 Returns: arc cosine of x within the range -p/2 to p/2
 Description: x must be within -1 to 1. If not domain error: returns 0.

asin #include <math.h>
 Prototype: double asin(double x);
 Returns: arc sine of x within the range -p/2 to p/2
 Description: x must be within -1 to 1. If not domain error: returns 0.

assert #include <assert.h>
 Prototype: void assert(int exp);
 Returns: void
 Description: if exp is zero, prints message: "Assertion failed: exp, filename, line_number" It then aborts the program.

atan #include <math.h>
 Prototype: double atan(double x);
 Returns: arc tangent of x; 0 if error.
 Description: The returned value is within the range -p/2 to p/2.

atan2 #include <math.h>
 Prototype: double atan2(double y, double x);
 Returns: arc tangent of y/x; 0 if error.
 Description: The returned value is within the range -p to p.

atof #include <stdlib.h>
 Prototype: double atof(const char *str);
 Returns: floating point equivalent of string, str.
 Description: Function converts a string to a floating point value. The function converts the characters in the string, str, from left to right until it hits a character other than those allowed in a floating point number.

atoi #include <stdlib.h>
 Prototype: int atoi(const char *str);
 Returns: integer equivalent of string str.
 Description: Function converts a string into an integer value. The function converts each digit in str from left to right until it hits a non-digit character.

atol #include <stdlib.h>
 Prototype: long atol(const char *str);
 Returns: long integer value of string, str

bsearch #include <stdlib.h>
 Prototype: void *bsearch(const void *key, const void *base, size_t nelem,
 size_t width, int (*fcmp)(const void *, const void*));
 Returns: address of first match or 0.
 Description: Searches the table of nelem elements, starting at base and
 returns the address of the first entry, of size, width, that matches the
 key. If no match is found, it returns a 0. The comparison function is
 *fcmp given the key pointer as its first argument and the element pointer
 as its second. It should return 0 if the two entries are equal, a
 positive value if the first is greater, and a negative value if the first
 is less.

cabs #include <math.h>
 Prototype: double cabs(struct complex z);
 Returns: absolute value, i.e. magnitude of a complex number, z.
 Description: complex is a structure defined in <math.h> with a real part,
 x and an imaginary part, y: struct complex{double x, y;};

calloc #include <stdlib.h>
 Prototype: void *calloc(size_t number, size_t size);
 Returns: pointer to the allocated block of memory; NULL if unsuccessful.
 Description: The calloc() function returns a pointer to the allocated
 block of memory. The block size is the number of items times size bytes.
 The return value is NULL if there is insufficient memory available. All
 bytes of allocated memory are initialized to zero.

ceil #include <math.h>
 Prototype: double ceil(double x);
 Returns: smallest whole number not less than x.
 Description: The ceil() function returns a double value representing the
 smallest integer that is greater than or equal to x.

clearerr #include <stdio.h>
 Prototype: void clearerr(FILE *stream);
 Returns: void
 Description: The clearerr() function resets the stream's error and end of
 file indicators to 0.

cos #include <math.h>
 Prototype: double cos(double x);
 Returns: cosine of x
 Description: Ranges as for sin().

cosh #include <math.h>
 Prototype: double cosh(double x);
 Returns: hyperbolic cosine of x

div #include <stdlib.h>
 Prototype: `div_t div(int numerator, int denominator);`
 Returns: quotient and remainder of numerator divided by denominator
 Description: The type `div_t` is defined in <stdlib.h> as a structure of integers:
 typedef struct {
 int quot;
 int rem;
 } `div_t`;

ldiv #include <stdlib.h>
 Prototype: `ldiv_t ldiv(long numerator, long denominator);`
 Returns: quotient and remainder of numerator divided by denominator
 Description: The type `ldiv_t` is defined in <stdlib.h> as a structure of integers:
 typedef struct {
 long quot;
 long rem;
 } `ldiv_t`;

exit #include <stdlib.h>
 Prototype: `void exit(int status);`
 Returns: void
 Description: The function `exit()` terminates the calling process; buffers are written out and all files are closed. The value of `status` is returned as the exit status of the process; 0 usually implies no error and non-zero indicates some error.

atexit #include <stdlib.h>
 Prototype: `int atexit(void (*fcn)(void));`
 Returns: non-zero if the registration cannot be made.
 Description: The function `atexit()` registers the function, `fcn`, to be called when a program exits normally.

exp #include <math.h>
 Prototype: `double exp(double x);`
 Returns: exponential of `x`, i.e. e^x

fabs #include <math.h>
 Prototype: `double fabs(double x);`
 Returns: absolute value of a floating point `x`

fclose #include <stdio.h>
 Prototype: `int fclose(FILE *fp);`
 Returns: 0 if successful; EOF if unsuccessful. Closes stream `fp`.

feof #include <stdio.h>
 Prototype: `int feof(FILE *stream);`
 Returns: 0 if end of file is not reached for stream; non-zero if end of file indicator is set.

ferror `#include <stdio.h>`
 Prototype: `int ferror(FILE *stream);`
 Returns: 0 if no error is indicated for the stream; non-zero if an error indicator is set. Note: `clearerr()` or `rewind()` can reset the indicator.

fflush `#include <stdio.h>`
 Prototype: `int fflush(FILE *fp);`
 Returns: 0 if successful; EOF if not successful.
 Description: This function forces any data in the buffer associated with the output file, `fp`, to be written out.

fgetc `#include <stdio.h>`
 Prototype: `int fgetc(FILE *stream);`
 Returns: the next character read from stream converted to an integer if successful; EOF if unsuccessful.

fgetpos `#include <stdio.h>`
 Prototype: `int fgetpos(FILE *stream, fpos_t *pos);`
 Returns: Function stores the file pointer for stream into `*pos` and returns 0 if successful; otherwise it returns a nonzero value and sets `errno` to EBADF defined in `<stdio.h>`. The type, `fpos_t`, is declared in `<stdio.h>` to be long.

fgets `#include <stdio.h>`
 Prototype: `char *fgets(char *str, int n, FILE *fp);`
 Returns: `str` is returned if successful, NULL if unsuccessful.
 Description: This function reads either until a newline is read or until `n` characters are read from a file pointed to by `fp` into the string `str`. A NULL character is appended to mark the end of the string, but the newline is not stripped.

floor `#include <math.h>`
 Prototype: `double floor(double x);`
 Returns: largest whole number, of type double, not greater than `x`.

fmod `#include <math.h>`
 Prototype: `double fmod(double x, double y);`
 Returns: floating point remainder of `x/y` with the same sign as `x`.

fopen `#include <stdio.h>`
 Prototype: `FILE *fopen(char *filename, char *mode);`
 Returns: a file pointer if successful; NULL if unsuccessful.
 Description: This function is used for opening a file. The string, `filename`, specifies name of the file and mode specifies how the file is to be accessed as one of the following:
 "r" open text file for reading
 "w" create text file for writing
 "a" open or create text file for writing at end of file
 "r+" open text file for update, i.e. reading and writing
 "w+" create text file for update; previous contents discarded
 "a+" open or create text file for update, writing at end of file
 If a file is a binary file, append a `b` to the mode string; e.g. "rb" for opening a binary input file. In Unix systems, do not append a `b`; there is

no difference between text and binary streams. If a file is opened for update, fflush() must be used between reading and writing.

fprintf #include <stdio.h>
 Prototype: int fprintf(FILE *fp, char *format_string, arg1, ...);
 Returns: number of characters sent to the file; EOF if unsuccessful.
 Description: This function prints arg1 and those following to the file
 pointed to by fp. It behaves like printf() except that it writes to a
 file.

fputc #include <stdio.h>
 Prototype: int fputc(int ch, FILE *fp)
 Returns: ch if successful; EOF if unsuccessful.
 Description: Outputs a character, ch, to the file pointed to by fp.

fputs #include <stdio.h>
 Prototype: int fputs(char *str, FILE *fp);
 Returns: EOF if unsuccessful; non-negative if successful.
 Description: Function writes a string of characters, str, to a file
 stream, fp. Characters are written to the file until the NULL character
 is reached. The NULL character is not written to the file. If a newline
 character is in the string, it is not stripped.

free #include <stdio.h>
 Prototype: void free(void *ptr);
 Returns: void
 Description: This function releases the block of memory pointed to by
 ptr, previously allocated by calloc() or malloc(). Releasing an allocated
 block of memory places the block back into the free memory pool of the
 heap.

fread #include <stdio.h>
 Prototype: size_t fread(void *buffer, size_t size, size_t no_items, FILE
 *fp);
 Returns: number of items read; NULL if error or end of file.
 Description: This function reads no_items, each of size, size, bytes from
 stream, fp, into buffer.

freopen #include <stdio.h>
 Prototype: FILE *freopen(const char *filename, const char *mode, FILE
 *stream);
 Returns: stream if successful; NULL if unsuccessful.
 Description: The function opens filename with specified mode and
 associates stream to the file. This function is typically used to change
 the files associated with stdin, stdout, and stderr to specified files.

frexp #include <math.h>
 Prototype: double frexp(double x, int *exp);
 Description: Splits x into a mantissa, m, and an exponent, n, of base 2,
 such that x is $m * 2^n$. The mantissa, i.e. fractional part, is less than 1
 and not less than 0.5. It returns m and stores n into *exp.

fscanf #include <stdio.h>
 Prototype: int fscanf(FILE *stream, char *format_string, arg1, ...);
 Returns: number of items read if successful; EOF if end of file.
 Description: This function reads values from stream and stores them where the arguments point. The arguments arg1 and those following must be pointers. It behaves like scanf() except that it reads from a file.

fseek #include <stdio.h>
 Prototype: int fseek(FILE *stream, long offset, int whence);
 Returns: 0 if successful; nonzero on failure.
 Description: The function, fseek, sets the file pointer for stream to a new position that is offset bytes from the location specified by whence; whence must be 0 for the beginning, 1 for the current position, or 2 for the end of file.

fsetpos #include <stdio.h>
 Prototype: int fsetpos(FILE *stream, const fpos_t *pos);
 Returns: 0 on success; a nonzero on failure.
 Description: Sets the file pointer for stream to a new position whose value was obtained by fgetpos() for that stream. It also clears end of file indicator for the stream.

ftell #include <stdio.h>
 Prototype: long ftell(FILE *stream);
 Returns: the current file pointer in stream measured in bytes from the beginning.

fwrite #include <stdio.h>
 Prototype: int fwrite(const void *buffer, int size, int no_objs, FILE *stream);
 Returns: the number of objects written if successful; less than no_objs on error or end of file.
 Description: This function writes no_objs objects of size, size, from buffer to stream.

getc #include <stdio.h>
 Prototype: int getc(FILE *stream);
 Returns: character read from stream converted to integer type if successful; EOF if unsuccessful.

getchar #include <stdio.h>
 Prototype: int getchar();
 Returns: Character read from stdin converted to its integer value if successful; EOF if unsuccessful.

getenv #include <stdlib.h>
 Prototype: char *getenv(const char *name);
 Returns: a pointer to a string corresponding to the environment variable, name. If the named variable does not exist, an empty string is returned.

gets `#include <stdio.h>`
 Prototype: `char *gets(char *str);`
 Returns: `str` if successful, `EOF` if unsuccessful.
 Description: This function reads a line from the standard input. Characters are read into `str` until the newline character is encountered. A `NULL` character is appended to mark the end of the string and the newline character is discarded.

isalnum `#include <ctype.h>`
 Prototype: `int isalnum(int ch);`
 Returns: non-zero if `ch` is alphanumeric, zero if `ch` is not alphanumeric.
 Description: This function checks to see if the character, `ch`, is alphabetic (`'A' .. 'Z'` or `'a' .. 'z'`) or numeric (`'0' .. '9'`).

isalpha `#include <ctype.h>`
 Prototype: `int isalpha(int ch);`
 Returns: non-zero if `ch` is alphabetic, zero if `ch` is not alphabetic.
 Description: This function checks to see if the character, `ch`, is alphabetic (`'A' .. 'Z'` or `'a' .. 'z'`).

isascii `#include <ctype.h>`
 Prototype: `int isascii(int c);`
 Returns: non-zero if `c` is ASCII; zero if `c` is not ASCII.
 Description: This function checks to see if the low order byte of `c` is in the range 0-127.

iscntrl `#include <ctype.h>`
 Prototype: `int iscntrl(int c);`
 Returns: non-zero if `c` is a control character/delete character; 0 otherwise.
 Description: This function checks to see if `c` is a control character, 0-31, or a delete character 127.

isdigit `#include <ctype.h>`
 Prototype: `int isdigit(int ch);`
 Returns: non-zero if `ch` is a digit; zero if `ch` is not a digit.
 Description: This function checks if `ch` is a digit character (`'0' .. '9'`).

islower `#include <ctype.h>`
 Prototype: `int islower(int ch);`
 Returns: non-zero if `ch` is a lower case alphabetic character (`'a' .. 'z'`) or zero otherwise.

isprint `#include <ctype.h>`
 Prototype: `int isprint(int c);`
 Returns: non-zero if `c` is a printable character in the range 32-126; 0 otherwise.

ispunct `#include <ctype.h>`
 Prototype: `int ispunct(int ch);`
 Returns: non-zero if `ch` is printable but not a space, letter, or a digit; zero otherwise.

isspace #include <ctype.h>
 Prototype: int isspace(int ch);
 Returns: non-zero if ch is whitespace (newline, tab, space); zero otherwise.

isupper #include <ctype.h>
 Prototype: int isupper(int ch);
 Returns: non-zero if ch is an upper case letter ('A' .. 'Z'); zero otherwise.

isxdigit #include <ctype.h>
 Prototype: int isxdigit(int ch);
 Returns: non-zero if ch is a hexadecimal digit ('0' .. '9', 'A' .. 'F' or 'a' .. 'f'); zero otherwise.

labs #include <stdlib.h>
 Prototype: long labs(long n);
 Returns: absolute value of its argument.

ldexp #include <math.h>
 Prototype: double ldexp(double x, double exp);
 Returns: x times 2 raised to exp.

log #include <math.h>
 Prototype: double log(double x);
 Returns: log returns natural logarithm of x. Note: x > 0.

log10 #include <math.h>
 Prototype: double log10(double x);
 Returns: log10 returns base 10 logarithm of x. Note: x > 0.

modf #include <math.h>
 Prototype: double modf(double x, double *intptr);
 Returns: The signed fractional portion of x.
 Description: The function breaks x into fractional and integral parts each with the same sign as x. The integral part is stored in *intptr and the fractional part is returned.

malloc #include <stdlib.h>
 Prototype: void *malloc(unsigned size);
 Returns: pointer to a block of memory allocated if successful; NULL if memory not available.
 Description: This function allows a program to obtain a block of memory from the heap. The size of the block is size bytes. Use sizeof operator to determine the size of a type.

perror #include <stdio.h>
 Prototype: void perror(const char *s);
 Returns: void
 Description: Prints first s and then an error message to stderr stream corresponding to the latest error.

pow #include <math.h>
 Prototype: double pow(double x, double y);
 Returns: x^y
 note domain error if: $x == 0$ and $y <= 0$ or $x < 0$ and y is not an integer.

printf #include <stdio.h>
 Prototype: int printf(char *format_string, ...);
 Returns: Number of bytes output; EOF if unsuccessful.
 Description: Output values of a variable number of arguments that follow
 format_string which describes how the output will be formatted for each
 following argument. The format string consists of ordinary characters and
 conversion specifications. Ordinary characters are simply copied to
 stdout. Conversion specifications provide the format information for
 outputting values. A conversion specification starts with a % and ends
 with a conversion character. In between the two can be a flag, width,
 precision, and a size modifier. Default justification is right
 justification with blanks to pad on the left.
 Flags:
 - left justification
 + signed numeric output: begins with a sign + or -
 blank non-negative values begin with a blank; negative with a -.
 # octals and hexadecimals printed with leading 0 and 0X; for float,
 decimal point always included and trailing zeros included for g
 conversion.
 Width:
 n at least n characters are printed; more if needed.
 0n same as n except that padding on the left is with zeros.
 * width specified in the argument list preceding the argument to be
 printed.
 Precision:
 .0 default
 .n n characters or n decimal digits after decimal point; output may be
 truncated or rounded.
 * argument list supplies precision preceding the argument to be
 output.
 Size Modifier:
 h short integer
 l long integer
 Conversion Characters:
 d prints the arguments as a decimal integer
 i prints the arguments as a decimal integer
 u prints the arguments as an unsigned decimal integer
 x / X prints the arguments as an unsigned hexadecimal integer
 o prints the arguments as an unsigned octal integer
 f prints signed value of the form [-]dddd.dddd, d represents a
 decimal digit.
 e signed value of the form [-]d.dddesdd, e represents exponent, s
 represents a sign + or -.
 g signed value in either e or f form; e is used only if the exponent
 is greater than precision or less than -4.
 c prints the argument as a character.
 s prints a string pointed to by the argument.
 % character % is printed.
 note: These conventions apply to all formatted output functions
 fprintf(), and sprintf(). The function, sprintf(), generates it's
 output to a string passed as the first argument (a pointer to the
 string) and the output to a string should be one call to sprintf().

putc #include <stdio.h>
 Prototype: int putc(int ch, FILE *stream);
 Returns: ch if successful; EOF if unsuccessful.
 Description: This is a macro defined in <stdio.h> which outputs character, ch to stream.

putchar #include <stdio.h>
 Prototype: int putchar(int ch);
 Returns: ch if successful; EOF if unsuccessful.
 Description: It is a macro that writes ch to standard output.

puts #include <stdio.h>
 Prototype: int puts(char *str);
 Returns: non-negative value if successful; EOF if unsuccessful.
 Description: Writes a NULL terminated str to stdout. The NULL is not written, and a newline is appended.

qsort #include <stdlib.h>
 Prototype: void qsort(void *base, size_t nelem, size_t width, int (*fcmp)(const void *, const void *));
 Returns: void
 Description: Uses quicksort to sort an array. A pointer to the base, i.e. 0th element, is given by base. Argument, nelem, is the number of elements in the table. Argument, width, is the size of each element in bytes. Argument, *fcmp, is the comparison function that returns a positive, zero, or a negative value if the first dereferenced argument is greater than, equal to, or less than the second dereferenced argument.

rand #include <stdlib.h>
 Prototype: int rand(void);
 Returns: a pseudo-randomly generated integer.

realloc #include <stdlib.h>
 Prototype: char *realloc(char *oldptr, int size);
 Returns: pointer to new block of memory if successful; NULL if insufficient memory.
 Description: The realloc() function changes the size of a previously allocated memory block, *oldptr, to size. The returned pointer points to the beginning of the new block of memory. The contents of the block are unchanged up to the shorter of the new and old sizes.

remove #include <stdio.h>
 Prototype: int remove(char *filename);
 Returns: 0 if successful; non-zero if it fails.
 Description: Removes the file, filename.

rename #include <stdio.h>
 Prototype: int rename(char *old, char *new);
 Returns: non-zero if it fails, 0 if successful.
 Description: Changes the name of a file from old to new.

rewind `#include <stdio.h>`
 Prototype: `int rewind(FILE *stream);`
 Returns: 0 if successful; nonzero otherwise.
 Description: Positions the file pointer in stream to the beginning of the file and clears error and end of file indicators.

scanf `#include <stdio.h>`
 Prototype: `int scanf(char *format_string, ...);`
 Description: This function reads values from standard input and stores them where the arguments point. The argument, `format_string`, specifies how the input fields are to be interpreted. It consists of white space, non-white space, and conversion specifications. Except for character conversion, it skips over all leading white space characters until a non-white space is encountered. It matches and discards all non-white space characters. Conversion specifications are used to convert the values in the input and store them where the arguments point. Conversion specifications start with a `%` and end with a conversion character. They may optionally include:

assignment suppression character	<code>*</code>	read and discard the next field
width specifier	<code>n</code>	maximum chars to read
modifier	<code>h, l</code>	pointer to short or long int

Conversion characters are `d, u, o, x, f, lf` (for double), `c, s`.
 Note: The above discussion applies to `fscanf()` and `sscanf()`. Read from a string using a single call to `scanf()`.

setbuf `#include <stdio.h>`
 Prototype: `void setbuf(FILE *stream, char *buffer);`
 Returns: void
 Description: This function allows the user to control buffering for stream. The function must be called after the file is opened but before and I/O operation is performed. If the buffer argument is NULL, I/O to the file associated with stream is unbuffered. If not, buffer must point to a character array of length `BUFSIZ`, the buffer size as defined in `<stdio.h>`.

sin `#include <math.h>`
 Prototype: `double sin(double x);`
 Returns: sine of `x`

sqrt `#include <math.h>`
 Prototype: `double sqrt(double x);`
 Returns: square root of `x` note: `x >= 0`.

srand `#include <stdlib.h>`
 Prototype: `void srand(unsigned seed);`
 Returns: void
 Description: Seeds the pseudo-random generator.

strcat `#include <string.h>`
 Prototype: `char *strcat(char *dest, char *src);`
 Returns: pointer to the concatenated string, `dest`
 Description: This function concatenates two strings. The NULL character at the end of `dest` is removed and `src` is appended to `dest`; the result is stored in `dest`.

strchr #include <string.h>
 Prototype: char *strchr(const char *s, int c);
 Returns: A pointer to the first occurrence of c in s. If c is NULL, a pointer to the NULL is returned. If c is not in s, NULL is returned.

strrchr #include <string.h>
 Prototype: char *strrchr(const char *s, int c);
 Returns: A pointer to the last occurrence of c in s. If c is not in s, NULL is returned.

strcmp #include <string.h>
 Prototype: int strcmp(char *str1, char *str2);
 Returns: negative if str1 < str2; 0 if str1 ==str2; positive if str1 > str2.
 Description: The two strings, str1 and str2 are scanned and corresponding characters are compared. The scan stops when there is a mismatch or a NULL character is reached. When the scan stops, the difference between the corresponding characters of str1 and str2 is returned.

strcpy #include <string.h>
 Prototype: char *strcpy(char *dest, const char *src);
 Returns: dest
 Description: This function copies characters from src to dest until a NULL character is encountered in src. NULL is appended to dest.

strerror #include <stdio.h>
 Prototype: char *strerror(int errnum);
 Returns: A pointer to an error string corresponding to error number, errnum.

strlen #include <string.h>
 Prototype: int strlen(const char *str);
 Returns: number of characters in the string, str. The NULL character at the end of a string is not counted.

strncat #include <string.h>
 Prototype: char *strncat(char *dest, const char *src, int n);
 Returns: dest

strncmp #include <string.h>
 Prototype: int strncmp(const char *s1, const char *s2, int n);
 Returns: first n characters of s1 and s2 are compared, and either zero or the difference of the first unequal characters is returned. The value returned is:
 negative if s1 < s2
 0 if s1 == s2
 positive if s1 > s2

strncpy #include<string.h>
 Prototype: char *strncpy(char *dest, const char *src, int n);
 Returns: dest
 Description: The above three functions perform the same operations as strcat(), strcmp(), and strcpy(), except that n specifies the number of characters

that are concatenated, compared, or copied. When copying or concatenating, a NULL is not appended unless the length of src is less than n.

strpbrk #include <string.h>
 Prototype: char *strpbrk(const char *s1, const char *s2);
 Returns: A pointer to the first occurrence in s1 of any of the characters of s2. It returns NULL if s1 does not contain any of the characters in s2.

strspn #include <string.h>
 Prototype: size_t strspn(const char *s1, const char *s2);
 Returns: The length of the initial part of s1 that consists entirely of characters from s2.

strcspn #include <string.h>
 Prototype: size_t strcspn(const char *s1, const char *s2);
 Returns: The length of the initial part of s1 that consists entirely of characters not in s2.

strstr #include <string.h>
 Prototype: char *strstr(const char *s1, const char *s2);
 Returns: The first occurrence of s2 in s1, i.e. a pointer to a character in s1 where s2 occurs for the first time. If s2 is not in s1, it returns NULL.

strtod #include <stdlib.h>
 Prototype: double strtod(const char *s, char **endptr);
 Returns: value of s as a double.
 Description: Scans s and converts characters to double; s may be in decimal or exponential format. String scan stops when a character that does not belong to a double is encountered. If endptr is not NULL, it sets *endptr to point to the first character that stopped the scan.

strtok #include <string.h>
 Prototype: char *strtok(char *s, const char *delimit);
 Returns: a pointer to the token or NULL if there is no token.
 Description: Finds tokens in s delimited by one or more characters contained in delimit. After the first call, it puts a NULL character in s right after the first token and returns a pointer to the first character of the first token. Subsequent calls with NULL for the first argument will work through the string, s, in the above manner until no token remains at which point a NULL pointer is returned.

strtol #include <stdlib.h>
 Prototype: long strtol(const char *s, char **endp, int base);
 Returns: The long equivalent of the first part of s. note the first part is the leading part of a string that satisfies a property. The remaining part is the part after the first part.
 Description: Converts the first part of string, s to long and stores the remaining part of s into *endp. If base is zero, then a base of 8, 10, or 16 is assumed with a leading 0 in s specifying octal, a leading 0X specifying a hexadecimal. Otherwise, base can be between 2 and 36.

strtoul #include <stdlib.h>
 Prototype: unsigned long strtoul((const char *s, char **endp, int base);
 Returns: same as strtol() except that an unsigned long is returned.

tan #include <math.h>
 Prototype: double tan(double x);
 Returns: tangent of x.

tanh #include <math.h>
 Prototype: double tanh(double x);
 Returns: hyperbolic tangent of x.

tmpfile #include <stdio.h>
 Prototype: FILE *tmpfile(void);
 Returns: a file pointer if successful; returns NULL otherwise.
 Description: Creates a temporary file of mode wb+ which will be removed
 when closed or when the program terminates normally.

tolower #include <ctype.h>
 Prototype: int tolower(int ch);
 Returns: lower case equivalent of ch if ch is upper case; otherwise,
 returns ch unchanged. Characters are converted to integer type.

toupper #include <ctype.h>
 Prototype: int toupper(int ch);
 Returns: upper case equivalent of ch if ch is lower case; otherwise ch is
 returned unchanged. Characters are represented as integer types.

ungetc #include <stdio.h>
 Prototype: int ungetc(int c, FILE *stream);
 Returns: character c if successful; EOF if unsuccessful.
 Description: This function pushes character, c, back onto stream. The
 next character read from stream will be c. A maximum of one character per
 stream may be allowed to be pushed back.

