INPUT AND OUTPUT STATEMENTS

- Non-formatted input and output
- Formatted input and output.

Non - Formatted Input & Output Functions

- Unformatted I/O functions works only with character data type (char).
- The unformatted Input functions used in C are getch(), getche(), getchar(), gets().
- The unformatted output statements in C are putch(), putchar() and puts().
- Among these functions some are alpha numeric and some are only for characters.

getch () –

- **Syntax:** variable_name = getch();
- The getch() function reads the alphanumeric character input from the user. But, that the entered character will not be displayed.

getche() –

- **Syntax:** variable_name = getche();
- getche() function reads the alphanumeric character from the user input. Here, character you entered will be echoed to the user until he/she presses any key.
- Like getch(), getche() also accepts only single character, but unlike getch(), getche() displays the entered character on the screen.

getchar() –

- **Syntax:** character_variable_name = getchar();
- getchar() function reads the alpha character from the user input. Here, character you entered will be stored in the mentioned character variable.

gets() -

- **Syntax:** gets(variable_name);
- gets() accepts any line of string including spaces from the standard Input device (keyboard).
- gets() stops reading character from keyboard only when the enter key is pressed.

putch() –

- **Syntax:** putch(variable_name);
- putch displays any alphanumeric characters to the standard output device. It displays only one character at a time.

putchar() –

- **Syntax:** putchar(variable_name);
- putchar displays one character at a time to the Monitor.

puts() -

- **Syntax:** puts(variable_name);
- puts displays a single / paragraph of text to the standard output device.

Example 1:

```
// program demonstrating the use of gets, puts, getch

#include<stdio.h>
#include<conio.h>
void main()
{
char a[20];
gets(a);
puts(a);
getch();
}
```

Example 2:

```
// program demonstrating the use of getche, getchar, putch, putchar

#include<stdio.h>
#include<conio.h>
void main()
{
char a,b;
printf("Enter any alpha numeric character\n");
a=getche();
printf("Enter any character\n");
b=getchar();
printf("The Entered alpha numeric character is-\t");
putch(a);
printf("The Entered character is-\t");
putchar(b);
getch();
}
```

Displaying Output using printf( )

- printf( ) is an output function in C used to display the content on the screen.
- For precise output formatting, every printf( ) call contains a format control string that describes the output format.
- The format control string consists of:
  - Conversion specifiers.
  - Field widths.
  - Precisions.
  - Literal characters.
  - Flags.
- Together with percent sign (%), these, form conversion specifications.
- Function printf() can perform the following formatting capabilities:
  - ✓ **Rounding** floating-point values to an indicated number of decimal places.
  - ✓ **Aligning** a column of numbers with decimal points appearing one above the other.
  - ✓ Right-justification and left-justification of outputs.
  - ✓ **Inserting literal characters** at precise locations in a line of output.
  - ✓ Representing floating-point number in **exponential format**.
  - ✓ Representing unsigned integers in **octal** and **hexadecimal format**.
  - ✓ Displaying all types of data with fixed-size field widths and precisions
- There are various forms of printf statements.
- *General form* of printf( ) is: ***printf (format-control-string, other-arguments);***
- The format-control-string describes the *output format*, and other-arguments (optional) correspond to each conversion specification in the format-control-string.
- Each conversion specification begins with a percent sign (%) and ends with a conversion specifier.
- **Printing Integers** - Integer is a whole number, such as 880 or –456, that contains no decimal point.

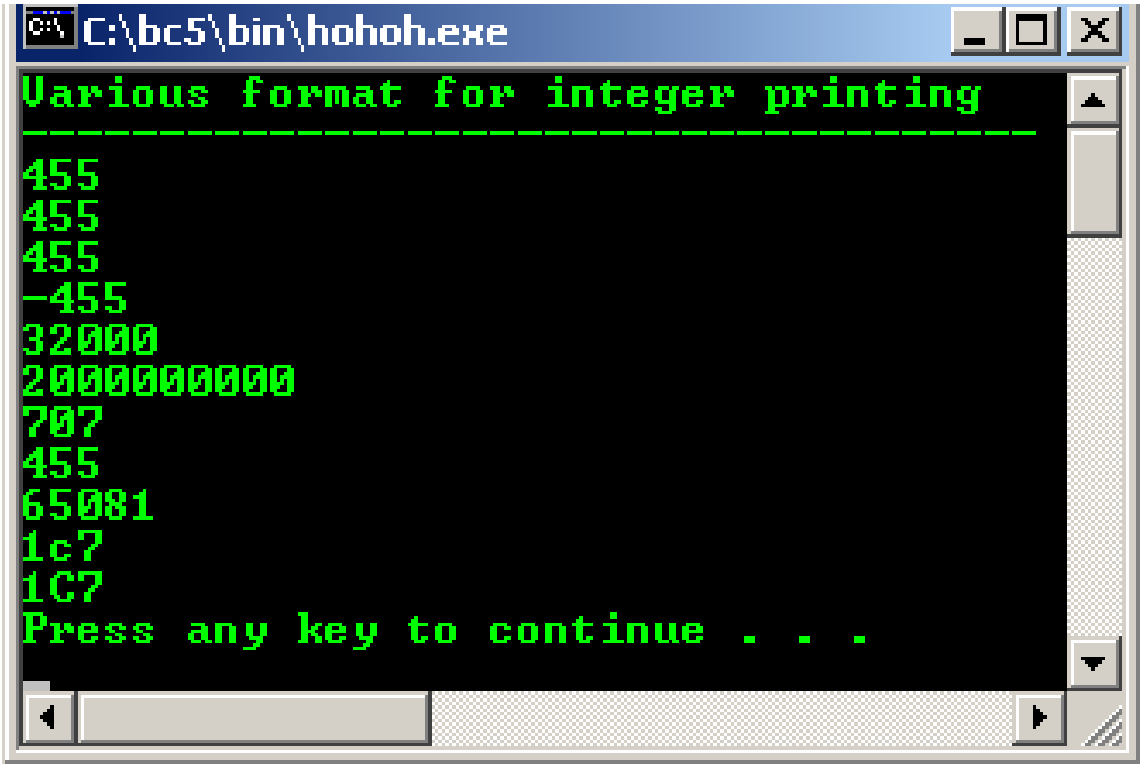
Conversion Specifier	Description
d	Display a signed decimal integer
i	Display a signed decimal integer (Note: Both are different in scanf())
o	Display a unsigned octal integer
u	Display a unsigned decimal integer
X or x	Display a unsigned hexa decimal integer
h or l (letter l)	Place before any integer conversion specifier to indicate that a short or long integer.

Example:

//Using the integer conversion specifiers

```
#include <stdio.h>
#include <stdlib.h>
        int main()
        {
                printf("Various format for integer printing\n");
                printf("-----------------------------------\n");
                printf("%d\n", 455);
                printf("%i\n", 455);                //i same as d in printf()
                printf("%d\n", +455);
                printf("%d\n", -455);
                printf("%hd\n", 32000);
                printf("%ld\n", 2000000000);
                printf("%o\n", 455);
                printf("%u\n", 455);
                printf("%u\n", -455);
                printf("%x\n", 455);
                printf("%X\n", 455);
        }
```

Output:



➲ **Printing Floating-point Numbers** - Contains the decimal point such as 35.5 or 7456.945.

Conversion specifier	Description
e or E	Display a floating-point value in exponential notation.
f	Display floating-point values.
g or G	Display a floating-point value in either the floating-point form f or the exponential form e (or E)
l	Place before any floating-point conversion specifier to indicate that a long double floating-point value is displayed.

➲ Exponential notation is the computer equivalent of scientific notation used in mathematics.

For example, 150.2352 is represented in scientific notation as:

1.502352×10^2

And is represented in exponential notation as

1.502352E+02 by computer

So, $150.2352 = 1.502352 \times 10^2 = 1.502352E+02$

➲ **E** stand for exponent.

➲ **e, E** and **f** will output with 6 digits of precision to the right of the decimal point by default.

Example:

```
//Printing floating-point numbers with
//floating-point conversion specifiers
        #include <stdio.h>
        #include <stdlib.h>
                void main()
                {
                        printf("Printing floating-point numbers with\n");
                        printf("floating-point conversion specifiers.\n");
                        printf("Compare the output with source code\n\n");
                        printf("1. %e\n", 1234567.89);
                        printf("2. %e\n", +1234567.89);
                        printf("3. %e\n", -1234567.89);
                        printf("4. %E\n", 1234567.89);
                        printf("5. %f\n", 1234567.89);
                        printf("6. %g\n", 1234567.89);
                        printf("7. %G\n", 1234567.89);
                }
```

Output:

```
Printing floating-point numbers with
floating-point conversion specifiers.
Compare the output with source code

1.  1.234568e+06
2.  1.234568e+06
3.  -1.234568e+06
4.  1.234568E+06
5.  1234567.890000
6.  1.23457e+06
7.  1.23457E+06
Press any key to continue . . .
```

- **Printing Strings & Characters** - c and s conversion specifiers are used to print individual characters and strings respectively.
- Conversion specifier c requires a char argument and s requires a pointer to char as an argument.
- s causes characters to be printed until a terminating NULL ('\0') character is encountered.

Example:

//Printing strings and characters

```
#include <stdio.h>
#include <stdlib.h>
        int main()
        {
                char character = 'A';
                char string[] = "This is a string";
                printf("---------------------------------\n");
                printf("---Character and String format---\n");
                printf("---------------------------------\n\n");
                printf("%c <--This one is character\n", character);
                printf("\nLateral string\n");
                printf("%s\n", "This is a string");
        }
```

Note: *% - Display the percentage character.*

```
#include <stdio.h>
#include <stdlib.h>
        int main()
        {
                printf("Printing a %% in a format control string\n");
        }
```

Output:

```
Printing a % in a format control string
```

- **Printing With Field Widths And Precisions** - A field width determines the exact size of a field in which data is printed.
- If the field width is larger than the data being printed, the data will normally be right-justified within that field.
- An integer representing the field width is inserted between the percent sign (%) and the conversion specifier in the conversion specification.
- Function printf( ) also provides the ability to specify the precision with which data is printed.
- Precision has different meanings for different data types.

Example 1 :

```
//Printing integers right-justified
#include <stdio.h>
#include <stdlib.h>
        int main()
        {
                printf(" Printing integers right-justified.\n");
                printf("Compare the output with the source code\n");
                printf("---------------------------------------\n\n");
                printf("%4d\n", 1);
                printf("%4d\n", 12);
                printf("%4d\n", 123);
                printf("%4d\n", 1234);
                printf("%4d\n\n", 12345);
                printf("%4d\n", -1);
                printf("%4d\n", -12);
                printf("%4d\n", -123);
                printf("%4d\n", -1234);
                printf("%4d\n", -12345);
        }
```

Example 2 :

```
//Using precision while printing integers,
//floating-point numbers and strings
#include <stdio.h>
#include <stdlib.h>
        int main()
        {
                int i = 873;
                float f = 123.94536;
                char s[] = "Happy Birthday";
                printf("Using precision while printing integers,\n");
                printf(" floating-point numbers, and strings.\n");
                printf("Compare the output with the source code\n");
                printf("---------------------------------------\n\n");
                printf("Using precision for integers\n");
                printf("\t%.4d\n\t%.9d\n\n", i, i);
                printf("Using precision for floating-point numbers\n");
                printf("\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f);
                printf("Using precision for strings\n");
                printf("\t%.11s\n", s);
        }
```

➲ **Flags in printf() Format Control String -** Flags used to supplement its output formatting capabilities

Flag	Description
- (minus sign)	Left-justify the output within the specified field
+ (plus sign)	Display a plus sign preceding positive values and a minus sign preceding negative values.
space	Print a space before a positive value not printed with the + flag.
#	Prefix 0 to the output value when used with the octal conversion specifier o. Prefix 0x or 0X to the output value when used with the hexadecimal conversion specifiers x or X. Force a decimal points for a floating-point number printed with e, E, f, g, or G that does not contain a fractional part. (Normally the decimal point is only printed if a digit follows it). For g and G specifiers, trailing zeros are not eliminated.
0 (zero)	Pad a field with leading zeros.

Example 1 -

```
//Right justifying and left justifying values
#include <stdio.h>
#include <stdlib.h>
int main()
{
printf("Right justifying and left justifying values.\n");
printf(" Compare the output with the source code.\n");
printf("----------------------------------------\n\n");
printf("%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23);
printf("%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23);
}
```

Example 2 -

```
//Printing numbers with and without the + flag
#include <stdio.h>
#include <stdlib.h>
int main()
{
printf("Printing numbers with and without the + flag.\n");
printf(" Compare the output with the source code\n");
printf("----------------------------------------\n\n");
printf("%d\n%d\n", 786, -786);
printf("%+d\n%+d\n", 786, -786);
}
```

Example 3 -

```
//Printing a space before signed values
//not preceded by + or -
#include <stdio.h>
#include <stdlib.h>
int main()
{
printf("Printing a space before signed values\n");
printf(" not preceded by + or -n\n");
printf("----------------------------------\n\n");
printf("% d\n% d\n", 877, -877);
}
```

Example 4 -

```
//o, x, X, and any floating-point specifier
#include <stdio.h >
#include <stdlib.h>
int main()
{
int c = 1427;
float p = 1427.0;
printf("o, x, X, and any floating-point specifiers\n");
printf("Compare the output with the source code\n");
printf("----------------------------------------\n\n");
printf("%#o\n", c);
printf("%#x\n", c);
printf("%#X\n", c);
printf("\n%#g\n", p);
printf("%#G\n", p);
}
```

Example 5 -

//Printing with the 0 (zero) flag fills in leading zeros

```
#include <stdio.h>
#include <stdlib.h>
        int main()
        {
                printf("Printing with the 0 (zero) flag fills in leading zeros\n");
                printf(" Compare the output with the source code\n");
                printf("-------------------------------------------------------\n\n");
                printf("%+09d\n", 762);
                printf("%09d", 762);
                printf("\n");
        }
```

Note:

1. String can also b printed directly -

printf(“ literal string”);

- ✓ Literal string may be any character or content. Whatever the content included within the double quotes will be displayed on the output screen
- ✓ ***Example:*** printf(“Welcome to India”);

  Output: Welcome to India

2. Guidelines for printf

- ✓ printf should contain control string or format string in quotes.
- ✓ Control string may or may not be followed by some variables or expressions.
- ✓ To print a value it needs convention specification or access specifier to be included.
- ✓ When printf statement is executed the convention specifier or access specifier will be replaced by its value.
- ✓ Additional characters included between the access specifiers will be maintained as it is on the output screen.

Inputting Values Using scanf

- Used for precise input formatting & to enter the input through the input devices like keyboard.
- Every scanf( ) function contains a format control string that describes the format of the data to be input.
- The format control string consists of conversion specifications and literal characters.
- Function scanf() has the following input formatting capabilities
  - Inputting all types of data.
  - Inputting specific characters from an input stream.
  - Skipping specific characters in the input stream.
- *General form of scanf( ):* **scanf (format-control-string, other-arguments);**

For example:

scanf("%e%f%g", &a, &b, &c);

- ✓ The format-control-string describes the formats of the input.
- ✓ The other-arguments are pointers to variables in which the input will be stored (i.e.., address of the variable where input values to be stored).

Example 1:

//Reading integers Constants

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int a, b, c, d, e, f, g;
printf("Reading integers from standard
input\n");
printf("----------------------------------
\n\n");
printf("Enter seven integers separated by
space: ");
scanf("%d%i%i%i%o%u%x", &a, &b,
&c, &d, &e, &f, &g);
printf("The input displayed as decimal
integers is: \n");
printf("%d %d %d %d %d %d %d\n", a, b,
c, d, e, f, g);
}
```

Example 2:

//Reading Real / folating numbers

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
float a, b, c;
printf(" Reading floating-point
numbers\n");
printf("Compare the output with the source
code.\n");
printf("----------------------------------------
\n\n");
printf("Enter three floating-point numbers,
separated by space: \n");
scanf("%e%f%g", &a, &b, &c);
printf("Here are the numbers entered in
plain\n");
printf("floating-point notation:\n");
printf("%f %f %f\n", a, b, c);
}
```

Example 3:

//Reading characters and strings

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
char x, y[20];
printf("Enter a string: ");
scanf("%c%s", &x, y);
printf("The input was: \n");
printf("the character \"%c\" ", x);
printf("and the string \"%s\"\n", y);
}
```

Example 4:

//input data with a field width

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int x, y;
printf("Enter a six digit integer: ");
scanf("%2d%d", &x, &y);
printf("The integers input were %d and
%d\n", x, y);
}
```

Note:

- **Guidelines for scanf**

1. No escape sequences or additional blank spaces should be specified in the format specifiers.

   Ex: scanf(“%d %f”,&a,&b); //invalid

   scanf(“%d\n%f”,&a,&b); //invalid

2. & symbol is must to read the values, if not the entered value will not be stored in the variable specified.

   Ex: scanf(“%d%f”,a,b); //invalid.

- For reading the string %s is the format specifier, where it will reads the string till it encounters with the blank space, if we want to read the complete sentence with the blank space as same as gets( ) we use the following statement:

  scanf(“%[^\n]s”,str);

1. scanf(“%[^\n]s”,str); - Same as gets( ) non formatted input function.
2. scanf(“%[A-Z]s”,str); - stores only capital letters to the character array str.
3. scanf(“%[^o]s”,str); - Reads all the characters but stops after first occurrence of ‘o’.