Lesson Number : 8

Input and Output in C :

There are numerous library functions available for I/O. These can be classified into two broad catagories.

1) Console I/O functions
2) File I/O functions.

1) Console I/O functions :

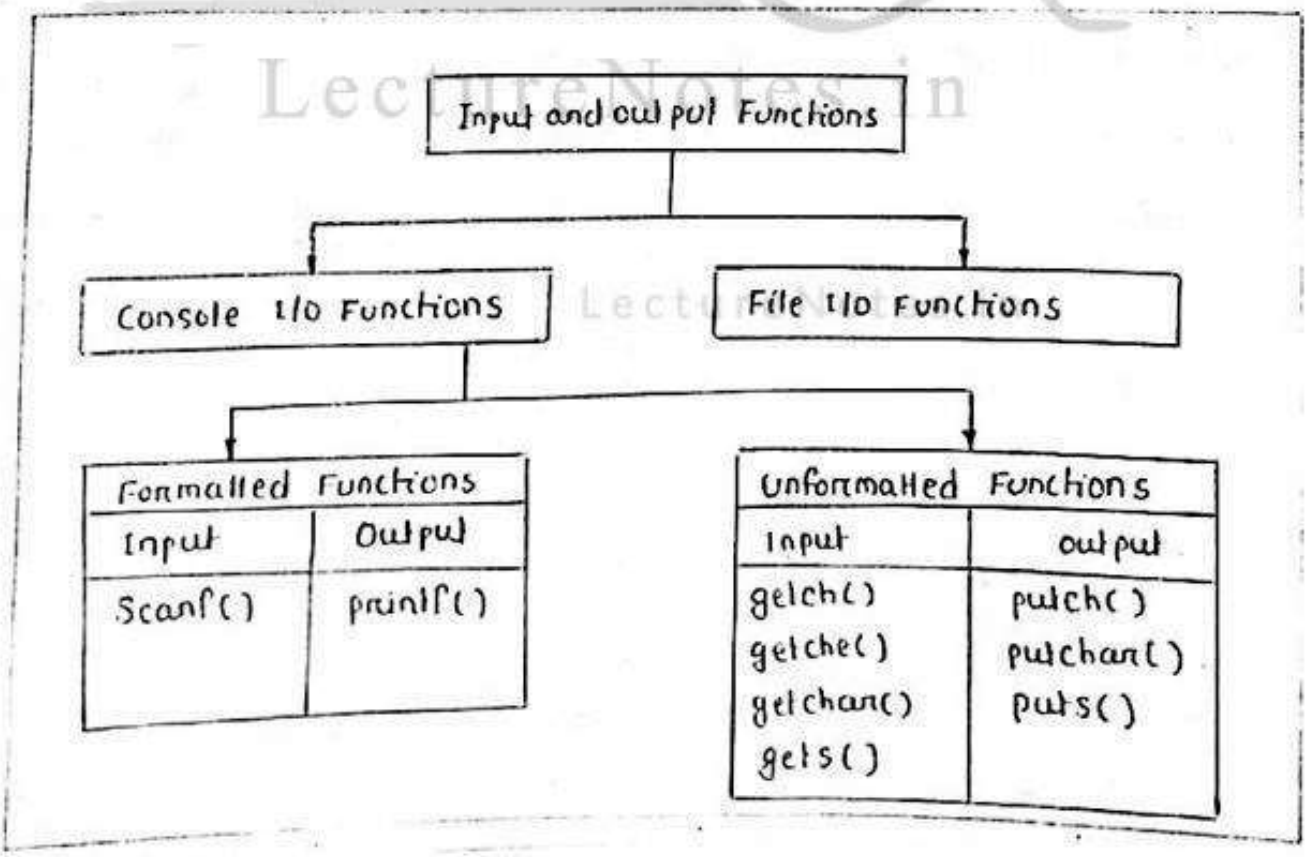
→ Console I/O functions are used to receive input from keyboard and write output to the monitor.

2) File I/O functions :

→ File I/O functions are used to perform I/O operations on a floppy disk or hard disk.

Console I/O Functions :

→ Console I/O functions can be further classified into two catagories
   i) Formatted console I/O functions
   ii) Unformatted console I/O functions.



i) Formatted Functions :

→ The formatted input/output functions read and write all types of data values.

→ They require conversion symbol to identify the data type. Hence they can be used for both reading and writing of all data values.

→ The formatted functions return the values after execution. The return value is equal to the number of variables successfully read or written. Using this value the user can findout the errors occuring during reading or writing the data.

Scanf() :

→ scanf() is the general purpose formatted console input function.

→ It can read all of the built-in data types and automatically convert numbers into the proper internal format.

→ The prototype is :

```
int scanf ( char * control-string , argument-list ) ;
```

scanf() returns the no. of data items successfully assigned a value. If an error occurs, EOF is returned.

→ The general form of scanf() is :

```
scanf ( "control strings" ,&argument1 ,&argument2 , .... ) ;
```

The control string specifies the field format in which the data is to be entered and the argument 1, argument 2 .... specifies the address of locations where the data is stored.

Control string and arguments are separated by Commas.

→ The control string consists of individual group of characters. with one character group for each data item. Each character group must begin with a percent sign (%). In its simplest form

a single character group will consists of the % sign, followed by a conversion character which indicates the type of the corresponding data item. within the control string multiple character groups can be separated by whitespace characters (i.e. blank spaces, tabs or newline characters).

→ The more frequently used conversion characters are ;

Specifier	Description
%c	Reads a single character.
%d	Reads a decimal integer.
%i	Reads a decimal integer.
%e	Reads a floating-point number.
%f	Reads a floating-point number.
%g	Reads a floating-point number.
%o	Reads an octal number.
%s	Reads a string.
%x	Read a hexadecimal number.
%p	Reads a pointer.
%u	Reads an unsigned integer.
%[]	Scan for a set of characters.

→ The field specification for reading an integer number is;

%wd

where Fieldwidth(w) : It is an integer no. that specifies the width of the no. to be read.

Datatype (d) : It indicates that the no. to be read is an integer mode.

Example:

```
scanf("%3d%4d", &a, &b);
```

If the data entered is 175 3245 then the value 175 will be assigned to a and 3245 to b.

Let the input data becomes 3245 175, then a will be assigned 324 (because of %3d) and b will be assigned 5 i.e. the unread part of 3245. The value 175 that is unread will be assigned to the next scanf call. This kind of error may be eliminated if we use the field specifications without the field width specifications.

For example: `scanf("%d%d", &a, &b);` will read the data 3245 and 175 correctly.

→ An input field may be skipped by specifying * in the place of field width.

Example: `scanf("%d%*d%d", &a, &b);`

If the data entered is 345 567 123, then it will assign 345 to a, 567 skipped (because of *) and 123 to b.

→ For inputting real numbers, we use the specification %f.

Example: `scanf("%f%f", &p, &q);`

If the input data is 2.312 457.10 then 2.312 will assing to p and 457.10 to q.

→ The field specification for reading a string containing sequence of character is;

%ws or %wc.

Example: `scanf("%s", str);`

→ All the variables used to receive values through scanf() must be passed by their addresses, which means that all arguments must be pointers to the variables used as arguments. This is ofcourse, C's way of creating a call-by-reference, which is a way of allowing a function to alter the contents of its calling arguments.

```
scanf ("%d", &count);

scanf ("%s", str);
```

The second line in the above code works because an array name without an index, is the address of the first element of the array. Hence the & operator is not required.

→ The specification for reading mixed data types can be written as ; `scanf ("%d %c %f %s", &i, &ch, &sum, name);`

If the data entered is 11 a 3.346 ram then it will assign the values to the variables in order in which they appear.

Printf():

→ printf() is the general purpose formatted console output function.

→ The prototype for printf() is;

```
int printf (char * control-string, argument-list);
```

→ The general form of printf() statement is;

```
printf ("control string", argument 1, argument 2 ....);
```

The control string indicates how many arguments follow and what their types are. The arguments are the variables whose values are formatted and printed according to the specifications of the control string.

→ The more frequently used formal specifiers in printf() are;

Specifier	Description
%c	character
%d	signed decimal integer.
%i	signed decimal integer.
%e	scientific notation (lowercase e)
%E	scientific notation (uppercase E)
%f	Decimal floating-point.
%s	string of characters.
%u	unsigned decimal integers.
%x	unsigned hexadecimal (lowercase letters)
%X	unsigned hexadecimal (uppercase letters)
%%	print a %

→ The Formal specification for printing an integer is

%wd

where Fieldwidth(w) specifies the min^m field wit width for the output and the datatype (d) specifies that the value to be printed is an integer.

Exp: printf("%d", number1);

If the number input is 4567 then the output will be 4567.

we can also write printf("%10d", num1);

Here if the no. input is 4567 then the output will be

4567 (i.e. after some space the no. is printed).

ii) Unformatted Functions:

→ The unformatted functions (input/output) only work with the character data type.

→ They donot require conversion symbol for identification of data types because they works only with character datatype. There is no need to convert the data.

→ The unformatted functions also return values but the return value of unformatted function is always the same.

getchar():

→ This function reads character type data from the standard input (keyboard).

→ It reads one character at a time till the user presses the enter key.

→ The getchar() takes the following form:

```
var-name = getchar();
```

where var-name is a valid char type variable.

→ Example:

```
char c;
c = getchar();
```

→ Use of getchar() function

```
#include <stdio.h>
void main()
{
  char ch;
  printf("\n Enter the character");
  ch = getchar();
  printf(" Entered character is: %c\n", ch);
}
```

output :

Enter the character a

Entered character is : a

getch() :

→ This function read any alphanumeric characters from the standard input device.

→ The character entered is not displayed by getch() function, but accepts the character.

getche() :

→ The getche() function is same as getch(), but it echoes the character on the screen.

→ The getche() accepts and display the character.

Example :

```
#include <stdio.h>
#include <conio.h>
Void main()
{ clrscr();
printf ("Enter two alphabetics");
getche();
getch();
}
```

output : Enter two alphabetics a

→ In the above program although two characters are entered, the user can see only one character on the screen. The second character is accepted but not displayed on the console. So the getche() accepts & displays the character whereas getch() accepts but does not display the character.

gets():

→ This function is used for accepting any string through keyboard untill enter key is pressed.

→ The header file stdio.h is needed for implementing this function.

Example:

```
#include <stdio.h>
void main()
{
  char name;
  printf("\n Enter the name");
  gets(name);
  printf("Entered name is : %s\n", name);
}
```

output : Enter the name xyz
Entered name is : xyz.

Putchar():

→ This function prints one character on the screen at a time which is read by the standard input.

→ The putchar() takes the following form;

putchar(var-name); where var-name is a valid char type variable.

Example :

```
#include <stdio.h>
void main()
{
  char ch;
  printf("\n Enter a character:");
  ch=getchar();
  printf("Entered character is:");
  putchar(ch);
}
```

output :

Enter a character : a

Entered character is : a .

putch() :

→ This function prints any alphanumeric characters taken by the standard input device.

→ Example :

```
# include <stdio.h>
void main()
{
  char ch ;
  printf ( " press any key to continue" ) ;
  ch = getch() ;
  printf ( "\n you pressed" ) ;
  putch (ch) ;
}
```

output : press any key to continue

You pressed 9

Here the function getch() reads a key stroke and assignes to the variable ch . The putch() displays the character pressed.

puts() :

→ puts() writes the string argument to the screen , followed by a new line .

→ gt cannot output numbers , but is faster than printf()

in outputting strings.

→ gt returns EOF , if an error occurs .

Example :

```
#include <stdio.h>
void main()
{
  char ch;
  puts("Enter a string");
  gets(ch);
  puts("Inputted string is :");
  puts(ch);
}
```

output :

```
Enter a string
Bhubaneswari
Inputted string is :
Bhubaneswari
```

Assignment : what do you mean by formatted I/O function and unformatted I/O functions ?

Formatted I/O Function :

→ The formatted input/output functions read and write all types of data values.

→ They require conversion symbol to identify the data type. Hence, they can be used for both reading and writing of all data values.

→ The formatted functions return the values after execution. The return value is equal to the number of variables successfully read or written.

→ The Formatted I/O Function include scanf() and printf() respectively.

Unformatted I/O Function :

→ The unformatted I/O functions only work with the character data type.

→ They don't require conversion symbol for identification of data types because they work only with character data type. There is no need to convert the data.

→ The unformatted functions return values but the return value of unformatted function is always same.

→ The unformatted I/O Function include getch(), getche(), getchar(), getc() and putch(), putchar(), puts() respectively.

Assignment : what are the function of console I/O function and file I/O function ?

→ Console I/O Functions are used to receive input from keyboard and write output to the monitor.

→ File I/O functions are used to perform I/O operations on a floppy disk or hard disk.

Assignment : write the general syntax of scanf() and printf().

→ The general syntax of scanf() is

```
scanf ("Control string" &argument1 &argument2 .....);
```

→ The general syntax of printf() is

```
printf ("Control string", argument1, argument2 ......);
```

Assignment : what is escape sequences ?

→ Escape sequences are control characters used to move the curson and print the characters such as ? , " , \ and so on.

→ Some of the escape sequences are ;

\a → Audible bell
\b → Backspace
\n → New line
\t → Horizontal tab
\v → vertical tab
\\ → Backslash
\0 → Null character.

Assignment : Find the o/p of the following program.

```
Void main()
  {
    char ch;
    printf("Enter a character\n");
    ch = getchar();
    printf("Entered character is :");
    putchar(ch);
  }
```

output : Enter a character a
Entered character is : a