UNIT – 3 Input and Output

There are three main function of any program

a. Input data
b. process data
c. output process data

The input operation involve movement of data moves from an input device (like keyboard) to computer memory, while in output operation the data moves from computer memory to the output device (monitor).
C language does not provide any facility for input/output operations. The input and output operation perform through a library function that are used by compiler. **“Library function is the predefine function, which is written by programmer, only we call that function and use in the program"**.
Before using library function in the program. we must include header file in the program, in which specific function is stored.
For example: for input and output operations we include **header file <stdio.h> "stdio" stands for standard input output operation and .h stands for header file**. All input output operation perform through this function. **Scanf();** and **printf()** is the member of stdio.h library function. How to take input and output data in C language, we must know what is Conversion Specification.

Conversion Specification

The function scanf( ) and printf( ) make use of conversion specification to specify the type and size of data.
Conversion specification is the way that specify and control input and output data. Data type size and type depend on this.
Each conversion specification must begin with a percent sign(%).

Some conversion specification give below-

%c	read/output a single character.
%d	read/output integer value.
%f	read/output floating point number.
%e	read/output floating point number
%lf	read/output long rang of float.
%h	for short integer.
%o	for octal integer.
%u	an unsigned integer.

For eg. to read data .

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

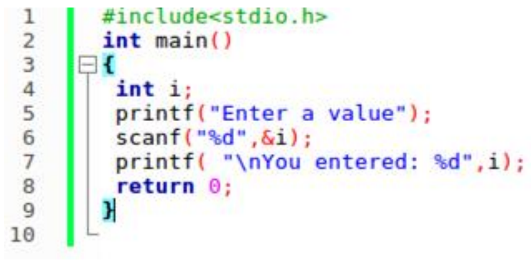
in above example %d specify that the input data is integer that is stored in variable 'a'. **&is used for address, with variable it create a memory location for variable, depend on data type.**
We declare float type variable then & create 4 byte on memory for variable. if we declare integer type variable then it create 2 byte memory.

Input/Output functions

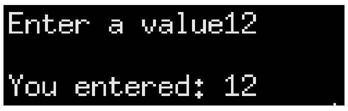
When we say Input, it means to feed some data into a program. An input can be given in the form of a file or from the command line. C programming provides a set of built-in functions to read the given input and feed it to the program as per requirement.
When we say Output, it means to display some data on screen, printer, or in any file. C programming provides a set of built-in functions to output the data on the computer screen as well as to save it in text or binary files.

printf() and scanf() functions.

printf() function returns the number of characters printed by it, and scanf() returns the number of characters read by it



```c
#include<stdio.h>
int main()
{
 int i;
 printf("Enter a value");
 scanf("%d",&i);
 printf( "\nYou entered: %d",i);
 return 0;
}
```



```
Enter a value12

You entered: 12
```

The function printf() is used for formatted output to standard output based on a format specification. The format specification string, along with the data to be output, are the parameters to the printf() function.

Syntax:

printf (format, data1, data2,……..);

In this syntax, format is the format specification string. This string contains, for each variable to be output, a specification beginning with the symbol % followed by a character called the **conversion character.**

Example:
printf (“%c”, data1);
The character specified after % is called a conversion character because it allows one data type to be converted to another type and printed.

The function scanf() is used for formatted input from standard input and provides many of the conversion facilities of the function printf().

Syntax:

scanf(format, num1, num2,……);

The function scanf() reads and converts characters from the standards input depending on the format specification string and stores the input in memory locations represented by the other arguments (num1, num2,….).

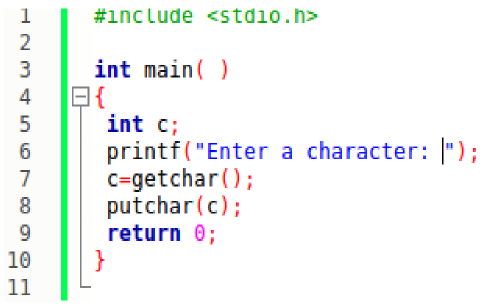
For Example:

scanf(" %c %d",&Name, &RollNo);

getchar() and putchar()

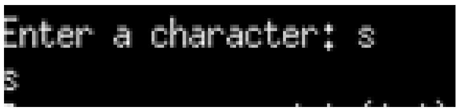
The getchar() function reads a character from the terminal and returns it as an integer. This function reads only single character at a time. We can use this method in the loop in case we want to read more than one characters.

The putchar() function prints the character passed to it on the screen and returns the same character. This function puts only single character at a time. In case we want to display more than one characters, use putchar() method in the loop.



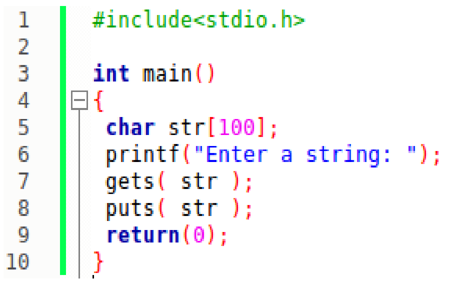
```c
#include <stdio.h>

int main( )
{
 int c;
 printf("Enter a character: ");
 c=getchar();
 putchar(c);
 return 0;
}
```



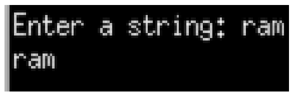
```
Enter a character: s
s
```

The gets() function reads a line from stdin into the buffer until either a terminating newline or EOF (end of file) occurs. The puts() function writes the string s and a trailing newline to stdout.



```c
#include<stdio.h>

int main()
{
 char str[100];
 printf("Enter a string: ");
 gets( str );
 puts( str );
 return(0);
}
```



```
Enter a string: ram
ram
```

The main difference between these two functions is that scanf() stops reading characters when it encounters a space, but gets() reads space as character too.

Printing Integers

- To output an integer, use %d in the format string, and an integer expression in the *list_of_expressions*.

  int numStudents = 35123;
  printf("FSU has %d students", numStudents);

  // Output:
  // FSU has 35123 students

- We can specify the field width (i.e. how many 'spaces' the item prints in). Defaults to right-justification. Place a number between the % and the d. In this example, field width is 10:

  ```
  printf("FSU has %10d students", numStudents);

  // Output:
  // FSU has      35123 students
  ```

- To left justify, use a negative number in the field width:

  ```
  printf("FSU has %-10d students", numStudents);

  // Output:
  // FSU has 35123      students
  ```

- Specifying the field width is most useful when printing multiple lines of output that are meant to line up in a table format

Printing Floating-point numbers

- Use the %f modifer to print floating point values in fixed notation:

  ```
  double cost = 123.45;
  printf("Your total is $%f today\n", cost);

  // Output:
  // Your total is $123.450000 today
  ```

- Use %e for exponential notation:

  ```
  printf("Your total is $%e today\n", cost);

  // Output:
  // Your total is $1.234500e+02 today
  ```

  Note that the e+02 means "times 10 to the 2nd power"

- We can also control the decimal precision, which is the number of places after the decimal. Output will round to the appropriate number of decimal places, if necessary:

  ```
  printf("Your total is $%.2f today\n", cost);

  // Output:
  // Your total is $123.45 today
  ```

- Field width can also be controlled, as with integers:

  ```
  printf("Your total is $%9.2f today\n", cost);

  // Output:
  // Your total is $   123.45 today
  ```

Printing characters and strings

- Use the formatting specifier %c for characters. Default field size is 1 character:

  ```
  char letter = 'Q';
  printf("%c%c%c\n", '*', letter, '*');

  // Output is:  *Q*
  ```

- Use %s for printing strings. Field widths work just like with integers:

```
printf("%s%10s%-10sEND\n", "Hello", "Alice", "Bob");
// Output:
 // Hello     AliceBob       END
```