C Input and Output

C programming language provides many built-in functions to read any given input and to display data on screen when there is a need to output the result.

`scanf()` and `printf()` functions

The standard input-output header file, named `stdio.h` contains the definition of the functions `printf()` and `scanf()`, which are used to display output on screen and to take input from user respectively.

```
#include<stdio.h>

void main()

{

    // defining a variable

    int i;

    /*

        displaying message on the screen

        asking the user to input a value

    */

    printf("Please enter a value...");

    /*

        reading the value entered by the user

    */

    scanf("%d", &i);

    /*

        displaying the number as output

    */

    printf( "\nYou entered: %d", i);

}
```

When you will compile the above code, it will ask you to enter a value. When you will enter the value, it will display the value you have entered on screen. `%d` inside the `scanf()` or `printf()` functions is known as **format string** and this informs the `scanf()` function, what type of input to expect and in `printf()` it is used to give a heads up to the compiler, what type of output to expect.

Format String	Meaning
`%d`	Scan or print an integer as signed decimal number
`%f`	Scan or print a floating point number
`%c`	To scan or print a character
`%s`	To scan or print a character string. The scanning ends at whitespace.

We can also **limit the number of digits or characters** that can be input or output, by adding a number with the format string specifier, like `"%1d"` or `"%3s"`, the first one means a single numeric digit and the second one means 3 characters, hence if you try to input `42`, while `scanf()` has `"%1d"`, it will take only `4` as input. Same is the case for output.

In C Language, computer monitor, printer etc output devices are treated as files and the same process is followed to write output to these devices as would have been followed to write the output to a file.

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

```
int i = printf("Computerscience");
```

In this program `printf("Computerscience");` will return `15` as result, which will be stored in the variable `i`, because `Computerscience` has 15 characters.

`getchar()` & `putchar()` functions

The `getchar()` function reads a character from the terminal and returns it as an integer. This function reads only single character at a time. You can use this method in a loop in case you want to read more than one character. The `putchar()` function displays the character passed to it on the screen and returns the same character. This function too displays only a single character at a time. In case you want to display more than one characters, use `putchar()` method in a loop.

```
#include <stdio.h>



void main( )
```

```
{
    int c;
    printf("Enter a character");
    /*
        Take a character as input and
        store it in variable c
    */
    c = getchar();
    /*
        display the character stored
        in variable c
    */
    putchar(c);
}
```

When you will compile the above code, it will ask you to enter a value. When you will enter the value, it will display the value you have entered.

`gets()` & `puts()` functions

The `gets()` function reads a line from **stdin**(standard input) into the buffer pointed to by `str` `pointer`, until either a terminating newline or EOF (end of file) occurs.
The `puts()` function writes the string `str` and a trailing newline to **stdout**.

`str` → This is the pointer to an array of chars where the C string is stored.

```
#include<stdio.h>
void main()
{
    /* character array of length 100 */
    char str[100];
    printf("Enter a string");
    gets( str );
    puts( str );
    getch();
}
```

When you will compile the above code, it will ask you to enter a string. When you will enter the string, it will display the value you have entered.

Difference between `scanf()` and `gets()`

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.

If you enter name as `Computer Science` using `scanf()` it will only read and store `Computer` and will leave the part after space. But `gets()` function will read it completely.