

Unit 3. Input and Output

3.1 Conversion specification,

3.2 Reading a character,

3.3 Writing a character,

3.4 I/O operations,

3.5 Formatted I/O

Displaying strings in screen

The two most frequently used ways to display text strings is to use C's library functions **printf()** and **puts()**.

- **The printf() Function**

The printf() function, part of the standard C library, is perhaps the most versatile way for a program to display data on-screen.

- Printing a text message on-screen is simple. Call the printf() function, passing the desired message enclosed in double quotation marks. For example,

```
printf("An error has occurred!");
```

Displays : An error has occurred!

- Similarly, puts() functions can also be used to display string in screen as:

```
puts("An error has occurred!");
```

- printf() function is used to display the values of variables, constants with formatting , but puts only prints text string in screen

Conversion specifications

- Displaying the value of program variables in screen is a little more complicated than displaying only a message.
- For example , if we have to display value of an integer variable x in screen, we can write as follows:

`printf("The value of x is %d", x);`

- Assume that value at variable x is 15 then the above statements display as:

The value of x is 15

- In above statement, %d is called the conversion specification used for the value of integer variable to be printed at that position.

Conversation specifications

The % Format Specifiers

<u>Format</u>	<u>Usual variable type</u>	<u>Display</u>
%c	char	single character
%d (%i)	int	signed integer
%e (%E)	float or double	exponential format
%f	float or double	single precision float
%g (%G)	float or double	use %f or %e as required
%o	int	unsigned octal value
%p	pointer	address stored in pointer
%s	array of char	sequence of characters(string)
%u	int	unsigned decimal
%x (%X)	int	unsigned hex value

A program to display value with format specifier

```
/*program */
#include<stdio.h>
int main()
{
    int x=10;
    float f=3.56;
    char ch='X';
    printf("Decimal NO x =%d",x);
    printf("\n Octal No x = %o",x);
    printf("\nHex NO x= %X",x);
    printf("\nFloat No f= %G",f);
    printf("\nFloat NO f =%f",f);
    printf("\nFloat No f =%E",f);
    printf("\nSingle Char ch = %c",ch);
    printf("\nString : %s","C-Program");
    return 0;
}
```

Output:

Decimal NO x = 10

Octal No x = 12

Hex NO x = A

Float No f = 3.56

Float NO f = 3.560000

Float No f = 3.560000E+000

Single Char ch = X

String : C-Program

Escape Sequence

- An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character.
- It is composed of two or more characters starting with backslash `\`. For example: `\n` represents new line.

Escape Sequence

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote

Escape Sequence

Character Input/Output

The `getchar()` and `putchar()` Functions

- The **int** `getchar(void)` function reads the next available character from the input buffer and returns it as an integer.

Example: `c = getchar();`

- This function reads only single character at a time. We can use this method in the loop in case of reading more than one character from the input buffer.
- The **int** `putchar(int c)` function puts the passed character on the screen and returns the same character.

Example `putchar(c);`

- This function puts only single character at a time. We can use this method in the loop in case to display more than one character on the screen.

Reading and writing a character

The scanf() and printf() Functions for read and write:

```
char c= '$';    /*declaration of char variable c */
```

```
printf("%c",c);    /*Prints character $ */
```

```
scanf("%c",&c);    for input a character in c
```

```
printf("%c",c);    for output a character in c
```

getc() and putc() character I/O Function

Function **getc()** : It reads a single character from the input and return an integer value. If it fails, it returns EOF. The syntax of getc() in C language is ,

```
int getc(FILE *stream);
```

Function **putc()**: It prints the passed character in screen

Example of getc() in C language

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

```
int main ()
```

```
{
```

```
    char ch;
```

```
    printf("Enter the character: ");
```

```
    ch = getc(stdin);
```

```
    printf("Character entered: ");
```

```
    putc(ch, stdout);
```

```
    return(0);
```

```
}
```

Output

```
Enter the character: a  
Character entered: a
```

The getch() Function

getch(): The function getch() is a non-standard function. It is declared in “conio.h” header file.

- Mostly it is used by Turbo C. It is not a part of C standard library.
- It immediately returns the entered character without even waiting for the enter key.
- Here is an example of getch() in C language:

```
#include <stdio.h>
#include<conio.h>
int main()
{
    char c;
    printf("Enter the character : ");
    c= getch();
    printf("Entered character : %c", c);
    return 0;
}
```

Output

```
Enter the character :
Entered character : #
```

The getch() Function

getche(): Like getch(), the getche() function is also a non-standard function and declared in "conio.h" header file.

- It reads a single character from the keyboard and returns it immediately without even waiting for enter key.
- Below is an example of getche() in C language.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    char val;
    printf("Enter the character : ");
    val = getche();
    printf(" Entered character : %c", val);
    return 0;
}
```

Output

- Enter the character : s Entered character : s

Input data with scanf()

- Most programs need to input data from the keyboard.
- The most flexible way our program can read numeric data from the keyboard is by using the scanf() library function.
- The scanf() function reads data from the keyboard according to a specified format and assigns the input data to one or more program variables.
- Like printf(), scanf() uses a same format string to describe the format of the input as
- For example, the statement **scanf("%d", &x);**
reads a decimal integer from the keyboard and assigns it to the integer variable x.
- Likewise, the following statement reads a floating-point value from the keyboard and assigns it to the variable rate:

scanf("%f", &rate);

scanf()

- The & symbol is C's address-of operator, which specifies the address of the variable stated
- The scanf() requires the & symbol before each numeric variable name in its argument list
- A single scanf() can input more than one value if we include multiple conversion specifiers in the format string and variable names (again, each preceded by & in the argument list).
- The following statement inputs an integer value and a floating-point value and assigns them to the variables **x** and **rate**, respectively:

scanf("%d %f", &x, &rate);

- When multiple variables are entered, scanf() uses white space to separate input into fields. White space can be spaces, tabs, or new lines.
- Each conversion specifier in the scanf() format string is matched with an input field and the end of each input field is identified by white space.
- This gives us considerable flexibility. In the preceding scanf(), we could enter
10 12.45
- Or or this:
10
12.45

An example of printf()/scanf()

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

```
Int main()
```

```
{
```

```
    float y;
```

```
    int x;
```

```
    puts( "Enter a float, then an int" );
```

```
    scanf( "%f %d", &y, &x);
```

```
    printf( "\nYou entered %f and %d ", y, x );
```

```
    return 0;
```

```
}
```

Output:

Enter a float, then an int: 4.56 45

You entered 4.560000 and 45

String Input with gets()

gets(): The gets() function reads a line from **stdin**(standard input) into the buffer pointed to by string pointer until either a terminating newline or EOF (end of file) occurs.

e.g.

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

```
int main()
```

```
{
```

```
    /* character array of length 100 */
```

```
    char str[100]; /* Will discussed later in chapter array in detail */
```

```
    printf("Enter a string : ");
```

```
    gets( str );
```

```
    printf("You Entered String: ");
```

```
    puts( str );
```

```
    return 0;
```

```
}
```

Output:

Enter a string: patan campus

You Entered String: patan campus

Input strings with scanf()

- We can use scanf() function to read text string from standard input. The syntax of scanf() for reading string literal is as below:

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

Example:

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

```
int main()
```

```
{
```

```
    /* character array of length 100 */
```

```
    char str[100]; /* Will discussed later in chapter array in  
detail */
```

```
    printf("Enter a string : ");
```

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

```
    printf("You Entered String: %s",str);
```

```
    return 0;
```

```
}
```

Output:

Enter a string: Patan

You Entered String: Patan

Note: Unlike gets(), scanf () reads string until any white space character is encountered.

Input strings with scanf()

- In previous example if we give the input with white space, it takes input only up to that white space character , rest is ignored.
- To read string with white space up to one line, scanf() function can be written as:

```
scanf("%[^\n]", string_variable) ;
```

Example:

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

```
int main()
```

```
{
```

```
    /* character array of length 100 */
```

```
    char str[100]; /* Will discussed later in chapter array in detail */
```

```
    printf("Enter a string : ");
```

```
    scanf("%[^\n]",str) ;
```

```
    printf("You Entered String: %s",str);
```

```
    return 0;
```

```
}
```

Output

Enter a string: Patan Campus

You Entered String: Patan Campus

Continued more Examples
In
Lab Sessions

Thank You