



# Input and Output in 'C'

---

## Session 4



# Objectives

---

- To understand formatted I/O functions -  
scanf() and printf()
- To use character I/O functions -  
getchar() and putchar()



# Standard Input/Output

---

- In C, the standard library provides routines for input and output
- The standard library has functions for I/O that handle input, output, and character and string manipulation
- Standard input is usually the keyboard
- Standard output is usually the monitor (also called the console)
- Input and Output can be rerouted from or to files instead of the standard devices



# The Header File `<stdio.h>`

---

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

This is a preprocessor command

- `stdio.h` is a file and is called the header file, contains the macros for many of the input/output functions used in 'C'
- `printf()`, `scanf()`, `putchar()`, `getchar()` functions are designed in such a way that, they require the macros in **`stdio.h`** for proper execution



# Formatted Input/Output

---

- **printf()** – for formatted output
- **scanf()** – for formatted input
- ***Format specifiers*** specify the format in which the values of the variables are to be input and printed



# printf ()-1

---

- printf() is used to display data on the standard output – console

**printf** ( "control string", argument list);

- The argument list consists of constants, variables, expressions or functions separated by commas
- There must be **one format command** in the control string for **each argument** in the list
- The format commands must match the argument list in number, type and order
- The control string must always be enclosed within double quotes, which are its delimiters



# printf ()-2

---

**The control string consists of one or more of three types of items:**

- 1. Text characters:**  
consists of printable characters
- 2. Format Commands:**  
begins with a % sign and is followed by a format code - appropriate for the data item
- 3. Nonprinting Characters:**  
Includes tabs, blanks and new lines



# Format codes-1

Format	printf()	scanf()
Single Character	%c	%c
String	%s	%s
Signed decimal integer	%d	%d
Floating point (decimal notation)	%f	%f or %e
Floating point (decimal notation)	%lf	%lf
Floating point (exponential notation)	%e	%f or %e
Floating point ( %f or %e , whichever is shorter)	%g	
Unsigned decimal integer	%u	%u
Unsigned hexadecimal integer (uses "ABCDEF")	%x	%x
Unsigned octal integer	%o	%o

In the above table c, d, f, lf, e, g, u, s, o and x are the *type specifiers*





# Format codes-2

---

Format Code	Printing Conventions
%d	The number of digits in the integer
%f	The integer part of the number will be printed as such. The decimal part will consist of 6 digits. If the decimal part of the number is smaller than 6, it will be padded with trailing zeroes at the right, else it will be rounded at the right.
%e	One digit to the left of the decimal point and 6 places to the right , as in %f above



# Control String Special Characters

---

<code>\\</code>	to print <code>\</code> character
<code>\ "</code>	to print <code>"</code> character
<code>%%</code>	to print <code>%</code> character



# control strings & format codes

No	Statements	Control String	What the control string contains	Argument List	Explanation of the argument list	Screen Display
1.	<code>printf("%d",300);</code>	<code>%d</code>	Consists of format command only	300	Constant	300
2.	<code>printf("%d",10+5);</code>	<code>%d</code>	Consists of format command only	10 + 5	Expression	15
3.	<code>printf("Good Morning Mr. Lee.");</code>	Good Morning Mr. Lee.	Consists of text characters only	Nil	Nil	Good Morning Mr. Lee.
4.	<code>int count = 100;</code> <code>printf("%d",count);</code>	<code>%d</code>	Consists of format command only	count	variable	100
5.	<code>printf("\nhello");</code>	<code>\nhello</code>	Consists of nonprinting character & text characters	Nil	Nil	hello on a new line
6.	<code>#define str "Good Apple "</code> ..... <code>printf("%s",str);</code>	<code>%s</code>	Consists of format command only	Str	Symbolic constant	Good Apple
7.	..... <code>int count,stud_num;</code> <code>count=0;</code> <code>stud_nim=100;</code> <code>printf("%d %d\n",count,stud_num);</code>	<code>%d %d</code>	Consists of format command and escape sequence	count, stud_num	two variables	0 , 100



# Example for printf()

---

Program to display integer, float , character and string

```
#include <stdio.h>
void main()
{
    int a = 10;
    float b = 24.67892345;
    char ch = 'A' ;

    printf("Integer data = %d", a);
    printf("Float Data = %f",b);
    printf("Character = %c",ch);
    printf("This prints the string");
    printf("%s","This also prints a string");
}
```



# Modifiers in printf()-1

---

## 1. '-' Modifier

The data item will be *left-justified* within its field, the item will be printed beginning from the leftmost position of its field .

## 2. Field Width Modifier

Can be used with type float, double or char array (string).

The field width modifier, which is an integer, defines , defines the *minimum field width* for the data item.



# Modifiers in printf()-2

---

## 3. Precision Modifier

This modifier can be used with type float, double or char array (string). If used with data type float or double, the digit string indicates the *maximum* number of digits to be printed to the right of the decimal.

## 4. '0' Modifier

The default padding in a field is done with spaces. If the user wishes to pad a field with zeroes this modifier must be used

## 5. 'l' Modifier

This modifier can be used to display integers as long int or a double precision argument. The corresponding format code is %ld



# Modifiers in printf()-3

---

## 6. 'h' Modifier

This modifier is used to display **short integers**.  
The corresponding format code is %hd

## 7. '\*' Modifier

If the user does not want to specify **the field width in advance**, but wants the program to specify it, this modifier is used



# Example for modifiers

---

```
/* This program demonstrate the use of Modifiers in printf() */
#include <stdio.h>
void main()
{
    printf("The number 555 in various forms:\n");
    printf("Without any modifier: \n");
    printf("[%d]\n", 555);
    printf("With - modifier : \n");
    printf("[%d]\n", 555);
    printf("With digit string 10 as modifier : \n");
    printf("[%10d]\n", 555);
    printf("With 0 as modifier : \n");
    printf("[%0d]\n", 555);
    printf("With 0 and digit string 10 as modifiers : \n");
    printf("[%010d]\n", 555);
    printf("With -, 0 and digit string 10 as modifiers: \n");
    printf("[%010d]\n", 555);
}
```





# scanf()

---

- Is used to accept data

The general format of scanf() function

```
scanf("control string", argument list);
```

- The format used in the printf() statement are used with the same syntax in the scanf() statements too.



# Differences in argument list of between `printf()` and `scanf()`

---

- **`printf()`** uses variable names, constants, symbolic constants and expressions.
- **`scanf()`** uses **pointers** to variables.

When using `scanf()` follow these rules for the argument list:

- If you wish to read in the value of a variable of basic data type, precede the variable name with a **&** symbol
- When reading in the value of a variable of derived data type, do not use a **&** before the variable name

# Differences in the format commands of the `printf()` and `scanf()`



---

- There is no `%g` option
- The `%f` and `%e` format codes are in effect the same



# Example for scanf()

---

```
#include <stdio.h>
void main()
{
    int a;
    float d;
    char ch, name[40];
    printf("Please enter the data\n");
    scanf("%d %f %c %s", &a, &d, &ch, name);
    printf("\n The values accepted are :
    %d, %f, %c, %s", a, d, ch, name);
}
```



# Buffered I/O

---

- used to read and write ASCII characters

A **buffer** is a temporary storage area, either in the memory, or on the controller card for the device

Buffered I/O can be further subdivided into:

- Console I/O
- Buffered File I/O



# Console I/O

---

- Console I/O functions direct their operations to the standard input and output of the system

In 'C' the simplest **console I/O functions** are:

- **getchar()** – reads one (and only one) character from the keyboard
- **putchar()** – outputs a single character on the screen



# getchar()

---

- Used to read input data, a character at a time from the keyboard
- Buffers characters until the user types a carriage return
- Has no argument, but the parentheses - must still be present



# Example for getchar()

---

```
/* Program to demonstrate the use of getchar() */
#include <stdio.h>
void main()
{
    char letter;
    printf("\nPlease enter any character : ");
    letter = getchar();
    printf("\nThe character entered by you is %c", letter);
}
```





# putchar()

---

- character output function in 'C'
- requires an argument

Argument of a **putchar()** function can be :

- A single character constant
- An escape sequence
- A character variable



# putchar() options & effects

---

Argument	Function	Effect
character variable	putchar(c)	Displays the contents of character variable c
character constant	putchar('A')	Displays the letter A
numeric constant	putchar('5')	Displays the digit 5
escape sequence	putchar('\t')	Inserts a tab space character at the cursor position
escape sequence	putchar('\n')	Inserts a carriage return at the cursor position



# putchar()

```
/* This program demonstrates the use of constants and escape sequences in putchar() */
```

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

```
void main()
```

```
{
```

```
    putchar('H'); putchar('\n');
```

```
    putchar('\t');
```

```
    putchar('E'); putchar('\n');
```

```
    putchar('\t'); putchar('\t');
```

```
    putchar('L'); putchar('\n');
```

```
    putchar('\t'); putchar('\t'); putchar('\t');
```

```
    putchar('L'); putchar('\n');
```

```
    putchar('\t'); putchar('\t'); putchar('\t');
```

```
    putchar('\t');
```

```
    putchar('O');
```

```
}
```

**Example**