Input output statements in 'C'

Streams ⇒ Streams are nothing but source data and the destination data. These streams are associated with Input and output devices.

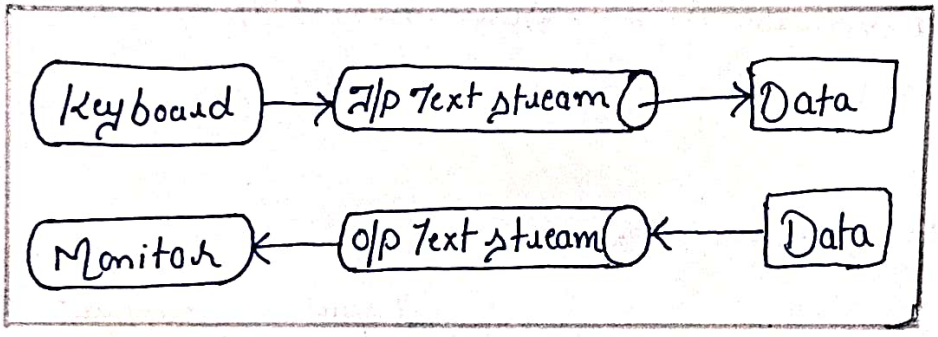
There are two types of streams.

→ Text stream.

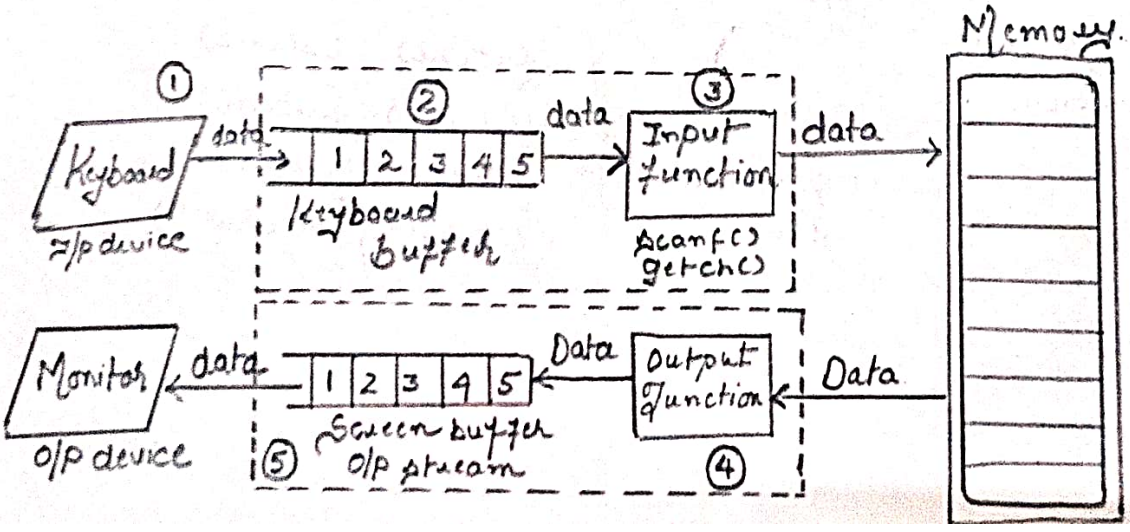
→ Binary stream.

Text stream, sequence of characters is divided into lines with each line being terminated with new line character. On the other hand binary stream contains data values using memory location.

Let us assume that source of data is keyboard & destination of data is monitor



Input output streams in 'C'.



① Data is Entered from Keyboard.

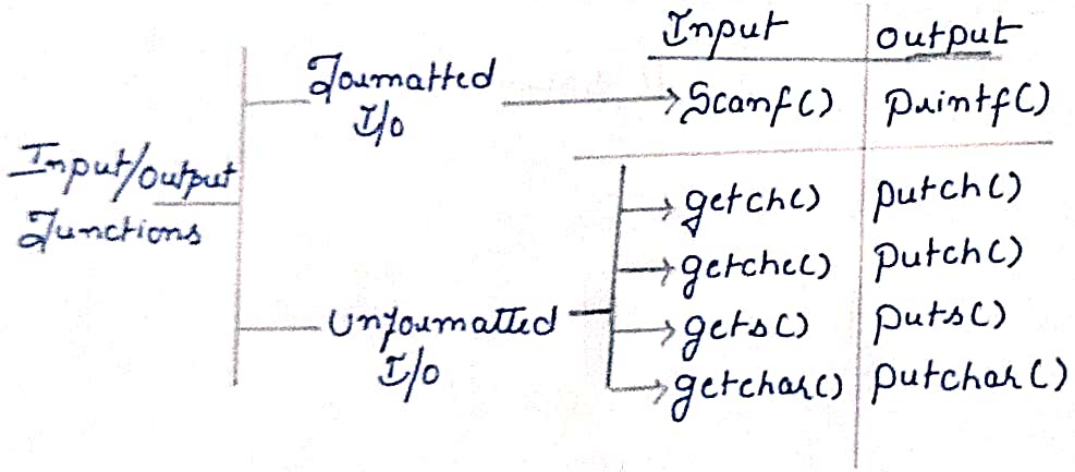
② Data Entered from the Keyboard will be stored in temporary storage area called keyboard buffer. This buffer holds characters untill we press ENTER. We can also edit characters entered by keyboard before pressing enter key.

③ using I/P functions (scanf(), getchar()) data stored in keyboard buffer is read & data is converted into appropriate data type. Converted data is then stored in memory location using variables.

④ Using o/p functions, data is read from the memory and stored in temporary storage area. This temporary storage is called Screen buffer.

⑤ Data available in the Screen buffer is visible on the Screen.

Types of I/O functions.



Rules for scanf()

Rule 1 ⇒ Scanf() works until
- a) Maximum number of characters has been processed.
- b) White space character is encountered.
- c) Error is detected.

Rule 2 ⇒ Every variable that has to be processed must have Conversion specification associated with it.

∴ Following scanf() will generate an error.

Ex: scanf("%d %d", &n1, &n2, &n3);

Error: n3 has no conversion spec" associated with it.

Rule 3 ⇒ There must be variable address for each Conversion specification.

∴ Following scanf() will generate an error.

Ex: scanf("%d %d %d", &n1, &n2);

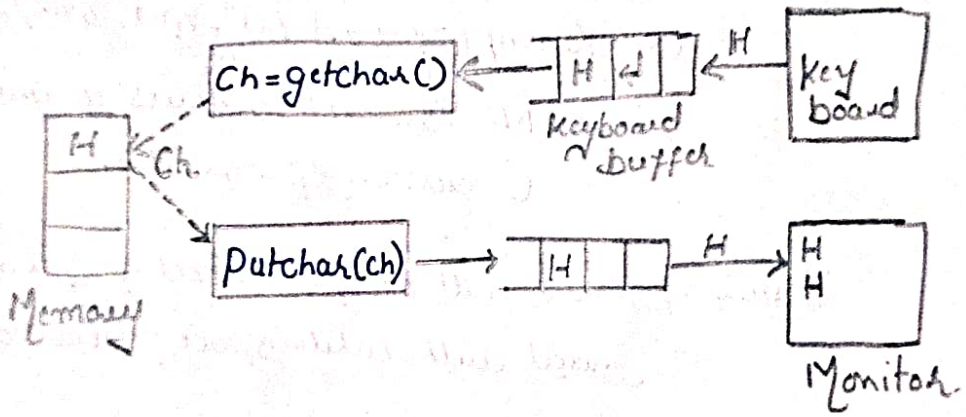
Error: No variable address is given for Conversion specification.

Rule 4 ⇒ Error will be generated if format string is Ended with white space character.

Rule 5 ⇒ Input data values must be seperated by the white spaces.

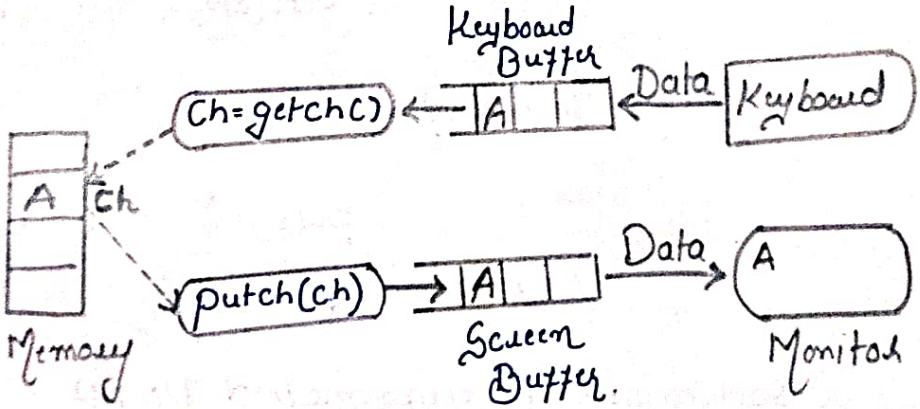
I/O functions	DESCRIPTION.
1) getchar()	Whenever we want to read a character from keyboard & store it into a memory location getchar() function is used. We have to press enter key after typing character.
2) Putchar()	Whenever we want to display the character stored in the memory location putchar() function is used.

Example Program for Putchar() & getchar()	`#Include<stdio.h>`
	`void main()`
	`{`
	`Char ch;`
	`ch=getchar(); //Input H↵`
	`putchar(ch); //output H`
	`}`

Pictorial representation	
3) getch()	When ever we want to read a character from the key board without echo (i.e., typed character will not be visible on the screen) we use getch(). The character thus entered will be stored in memory location.

I/O Function.	DESCRIPTION
getche()	Whenever we want to read a character from the keboard with echo (ie. typed character will be visible on the screen) then we use getche() function. The character thus entered will be stored in memory location.
Putch()	Whenever we want to display the character stored in memory location on the screen putch() can be used.

Example Prog for getch() & putch().	`#include<stdio.h>`
	`#include<conio.h>`
	`Void main()`
	`{`
	`Char ch;`
	`ch=getch();` // A (Character will not be visible)
	`putch(ch);` // A
	`}`

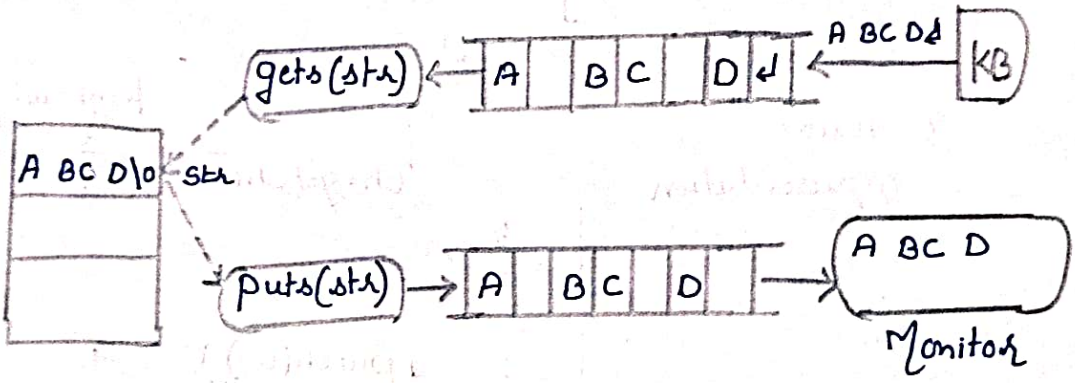
Pictorial Representation	

I/O function	DESCRIPTION
gets()	whenever we want to read sequence of characters from keyboard with spaces in between & store them in memory location then gets() is used.
puts()	whenever we want to display sequence of characters stored in the memory location then on the screen, then puts() function is used.

Example Prog

```
#Include <stdio.h>
#include <conio.h>
Void main()
{
  Char str[15];
  gets(str);    // A BC D ↵   (Input)
  puts(str);    // A BC D     (output)
}
```

Pictorial representation.

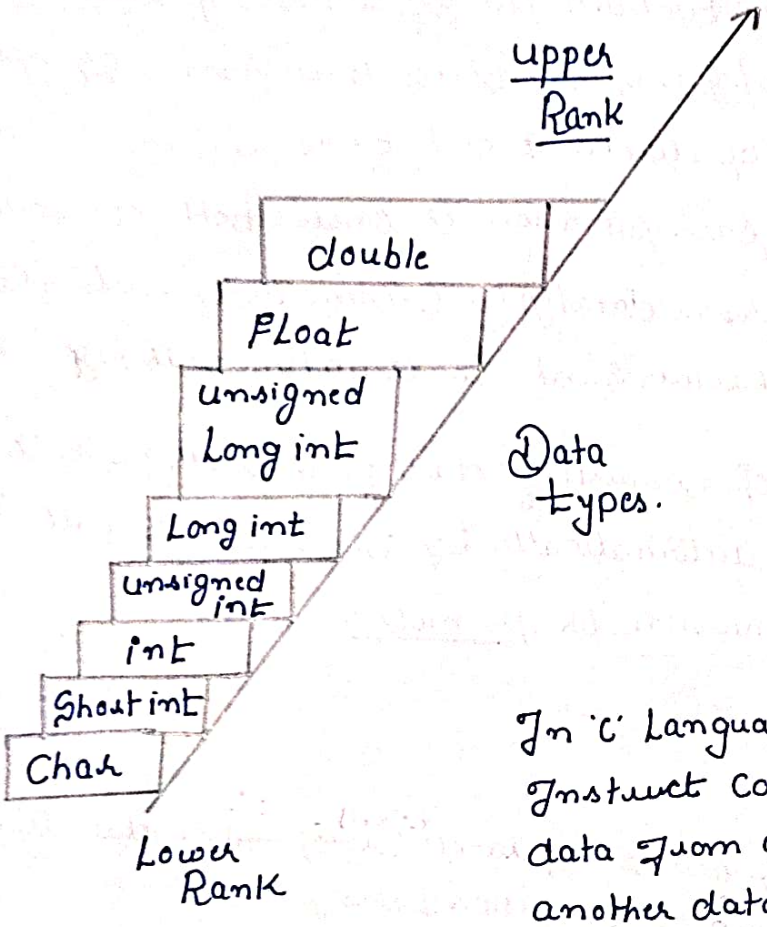


Disadvantages of unformatted I/O fn:

It is not possible to read/print any other data except characters. That is it is not possibe to read/print Integers, Floating Point numbers etc. "To overcome this disadvantage we use Formatted I/O functions".

Type Conversion and type Casting.

Type conversion or type casting of variables refers to changing the variable of one datatype to other. Type conversion is done Implicitly, where as typecasting has to be done Explicitly by the programmer.



In 'C' Language programmer can instruct compiler to convert data from one data type to the another datatype.

Some times compiler itself will convert the data from one data type to another datatype. This process of converting data from one datatype to another is called type conversion (conversion done Implicitly).

Type Conversion

Type	Example	after Conversion	Result
Implicit Conversion	1/2.0	1.0/2.0	0.500000
Explicit Conversion (Type Casting)	1/(float)2	1.0/2.0	0.500000.

Implicit Conversion ⇒ 'C' can Evaluate the Expressions if and only-iff data types of two operands are same. If operands are of different data types 'C' cannot evaluate. In such situation to ensure both operands are of same datatype, C-compiler converts datatype with lower rank to datatype with higher rank

This process of converting data from lower rank to the higher rank automatically by 'C' compiler is called Implicit Conversion OR Promotion.

For Ex:

int + int = int float + float = float	If both the operands are of same data type then no conversion takes place.
Char + int = int	If one operand is char & other operand is int, then operand with char is promoted to int
Int + float = float	the operand with int is promoted to float & result will be float. (This is because float is up in the ladder compared to int)

Numerical Examples :

1) int + int int

5 + 3 = 8

8 ------ ↑

No Conversion

2) int + float float

5 + 3.5 = 8.5

↓ ↑

5.0 + 3.5 ---- ↑

8.5 ------

(float)

int to float Conversion

3) int + double double.

5 + 2.1123 = 7.1123

↓ ↑

5.0 + 2.1123 ---- ↑

7.1123 ------

(double)

Note : When data is converted from Lower rank to higher rank, data is not Lost.

If data conversion is done from higher rank to Lower rank, we may lose the data. So compiler always converts data from Lower rank to higher rank so data is not lost.

Ex: Evaluate 4.0/3

4.0 ⇒ float

3 ⇒ Int.

float has higher rank than int. So Int is promoted to float

float/int = float

4.0/3 = 1.333333

↓ ↑

4.0/3.0 ------

1.333333

Explicit Type Conversion

When Implicit Conversion is done by the compiler automatically, the question is "what is the need for explicit type conversion"

Implicit type conversion is possible only when datatypes of both operands are same different. In a situation where datatype of two operands are same still Conversion is required, then we have to go for Explicit type Conversion.

Ex: Suppose we want to find ratio of two values. Then ratio is given by $n1/n2$.

Here Let us assume n1 & n2 are Integer values. But when divided to find ratio the result may end up with Float. In such a situation we are forced to to convert one of the operand to float. This is the place where we require Explicit type conversion.

Decesion Control & Looping Statements.

Decesion Control statements

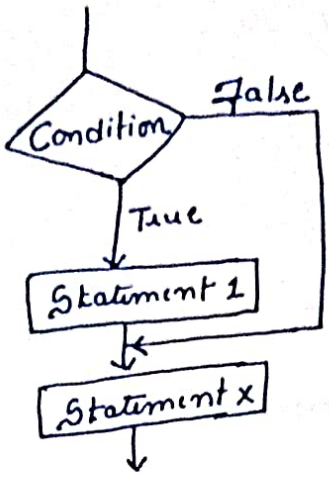
- → if statement
- → if-else statement } Conditional type
- → if-else-if statement
- → Switch statement } unconditional type.

if Statement

It is the simplest form of decision control statement that is frequently used in decision making.

Syntax

```
if (Condition)
{
  Statements 1;
  .....
  .....
  Statement n;
}
 Statement x;
```



Note: There is no Semicolon after Condition.

Ex: Write a prog to determine whether person is Eligible to Vote or not.

```
#include<stdio.h>
Void main()
{
   Int age;
   Printf("Enter age of a person : ");
   Scanf("%d", &age);
    if (age > 18)
       Printf("you are eligible to vote");
       Return(0);
   }
```

O/P

```
Enter age of a person : 20
You are eligible to vote.
```

if-else-if statement

This statement is used to check multiple cond^{n} in a sequence & execute different blocks based on which cond^{n} is true.

Syntax:

```
if (condition 1).
{
  // statements; ⇒ Executed if condition 1
                   is true.
}
Else if (condition 2)
{
  // statements; ⇒ Executed if condition 2
                   is true.
}
Else if (condition 3)
{
  // statements; ⇒ Executed if condition 3
                   is true.
}
Else
{
  // statements; ⇒ Non of the above cond$^{n}$
                   are true;
}
```

Ex:

```
#include<stdio.h>
void main()
{
int score=85;
if (score>=90)
{ pf ("Grade = A\n");
} else if (score>=80)
{ pf ("Grade = B\n");
} else if (score>=70)
{ pf ("Grade = C\n");
} else if (score>=60)
{ printf ("Grade = D\n");
} Else
{pf ("Grade = F"); } return(0); }
```

Iterative Statements

Iterative statements are used to repeat execution of list of statements, depending on the value of an Integer. 'C' Lang supports 3-types of Iterative statements.

1) While Loop
2) Do-while Loop
3) For Loop.

While Loop

Also known as "Pretest/top testing Loop"

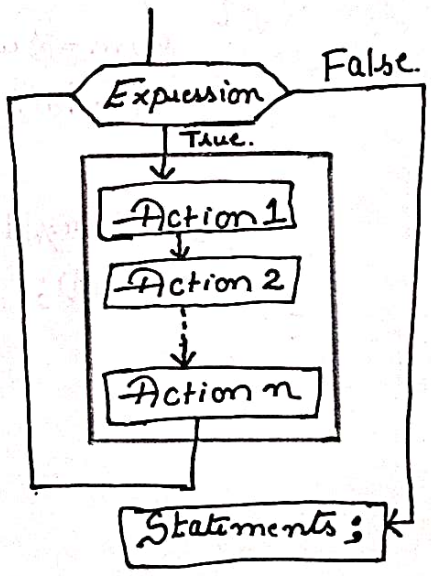
While Loop provides a mechanism to repeat one or more statements while particular condition is true. Once specified Condn is false comes out from the Loop.

Note: Expression(condition) is evaluated first & then body of the Loop is executed if result of Expn is true.

Syntax:

```
While (Expression)
{
  Action/task1;
  Action/task2;
    :
  Actionn/taskn;
}
```

Flowchart:



Example:

1) Write a program to print first 10-numbers;

```
#include<stdio.h>
Void main()
{
   Int i=1;
   while(i<=10) ——————— //Test the Condition.
   {
     Printf("%d", i);  } ——— //Execute body of the Loop
     i=i+1; ————————  } ——— //update Condition.
   }
}
```

2) Write a prog to calculate sum of first 10 numbers.

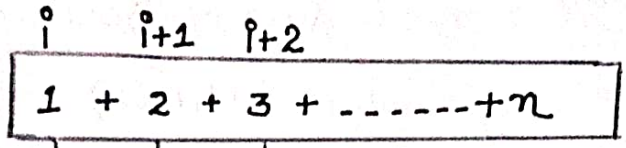
ie 1+2+3+4+.....+10;

```
#include<stdio.h>
Void main()
{
   Int i=1, sum=0;.
   while(i<=10)
   {
     Sum=Sum+i;
      i=i+1;
   }
   Printf("%d", Sum);
   return(0);
}
```

Tracing

i = 1;

Initially Sum = 0;



Step 1 ⇒ Sum = Sum + i

Sum = 0 + 1

Sum = 1.

i = i + 1;

Step 2 ⇒ Sum = 1 + 2

Sum = 3

i = 2 + 1 ⇒ 3

Step 3 ⇒ Sum = 3 + 3

Sum = 6

i = 3 + 1 ⇒ 4

In general.

Sum = Sum + n

Algorithm.

Step 1 : Start

Step 2 : [Initialization]
initialize Sum = 0.
i = 1.

Step 3 : [Find Sum of all terms]
while (i <= n)
Sum = Sum + i
i = i + 1
End while.

Step 4 : [Return the result]
Return result.

To compute sum of series using functions

```
#Include <stdio.h>
Int main().
{
   int n, sum;
   Printf("Enter n value:");
   Scanf("%d", &n);
   Sum = ComputeSum(n);
   Printf("Sum of Series = %d", Sum);
} Return(0);

int ComputeSum(int n)
{
   int Sum, i;
     Sum = 0;
        i = 1;

     While (i<=n)
     {
        Sum=Sum+i;
           i = i+1 OR i++;
     }
   Return(Sum);
}
```

Prog to Calculate Sum of numbers from m to n;

```
#include <stdio.h>
void main()
{
    int m, n, sum=0;
    Printf ("Enter values of m & n");
    Scanf ("%d %d", &m, &n);
    While (m<=n)
    {
        Sum = sum+m;
        m++;
    }
    Printf("Sum= %d", Sum);
}
```

7
8
9
10
11
45

Expected output:

Enter value of m & n: 7 11

Sum = 45

Prog to display Largest of 5 numbers using Ternary operator.

```
#Include<stdio.h>
void main()
{
    Int i=1, n, Large;        Initialize Large = -32768
    While (i<=5)
    {
        Printf ("Enter the number :");
        Scanf ("%d", &n);
        Large = n > Large ? n : Large;
    }
    Pf("Largest Number is =" %d", Large);
    return(0);
}
```

Prog to Compute Sum of Series

$1 + x + x^2 + x^3 + x^4 + \cdots x^n$

```
#include <stdio.h>
#include <math.h>
Void main() OR Int main()
{
    Int n, x, sum;
    Printf("Enter the value of x :");
    Scanf("%d", &x);
    Printf("Enter the value of n :");
    Scanf("%d", &n);
    Sum = ComputeSum(n, x);
    Printf("Sum of Series = %d ", Sum);
}

Int ComputeSum(int n, int x);
{
    Int Sum, i);
    Sum = 1, i = 1;
    while (i <= n)
    {
        Sum = Sum + pow(x, i);
        i++;
    }
    Return(sum);
}
```

Do-while Loop.

Do-while Loop is a type of Loop in 'C' Programming where you want to run the code atleast once regardless of the Condition.

Syntax:

```
do ------------------------ //Do first, Check Later
{
   Statements 1;
      :
   Statement n;      } ⇒ //Execute the Statements
}
while (condition); ------ //Check Condition after one run.
```

While Loop	do-while Loop.
1) Checks the Condition before running the Loop	1) Runs Loop once before checking the condition.
2) Does not Execute if Condition is false initially	2) Executes once, even if the Condition is false.

3) Syntax:
while (condition)
{
CODE;
}
	3) Syntax:
	do
	{
	//CODE;
	}
	while (condition);

4) No o/p if Condition is false	4) Executes & outputs the result at least once, even if condition is false.
5) No semicolon at End of while statement	5) There is semicolon at the End of do-while statement.

Example:

```
#include<stdio.h>
void main()
{
   int x=1;
   while(x>=10)
   {
     printf("%d", x);
     x=x+1;
   }
}
```

output : Doesn't displays anything.

```
#include<stdio.h>
void main()
{
   int x=1;
   do
   {
     printf("%d", x);
     x++;
   }
   while(x>=10);
}
```

o/p : value of x now is 1

Example 2 :
To reverse the given Number.

```
#Include<stdio.h>
void main()
{
   int num, rev, rem=0;
   Printf("Enter number");
   Scanf("%d", &num);
   int original num = num;
   while(num != 0)
   {
     rem = num %10;
     num = num/10;
     rev = rev *10 + rem;
   }
   Pf("reverse of %d is %d",
        original num, num);
}
```

Tracing.

Initially rev = 0;
num = 1234.
original num = 1234.

1st pass
rem = 1234%10
= 4
num = num/10
= 123
rev = 0*10 + 4.
= 0+4
= 4.
Now num = 123.
rem = 4
rev = 4.

IInd pass
rem = 123%10 ⇒ 3
num = 123/10 ⇒ 12
rev = 4*10 + 3
= 43

Pass III
rem = 12%10 = 2
num = 12/10 = 2
rev = 43*10 + 2.
= 432.

IV pass
num = 1
∴ rem = 1%10 ⇒ 1
num = 1/10 ⇒ 0.1
rev = 432*10 + 1
= 4321

orig ⇒ 1234.
Rev ⇒ 4321

Prog to find sum of digits of the number.

```
#include<stdio.h>
void main().
{
  Int num, rem, sum=0;
  Printf("Enter the value:");
  Scanf("%d", &num);
  while(num!=0)
  {
    rem=num%10;
    num=num/10;
    sum=sum+remainder;
    if(num!=0)
    {
      Printf("%d+");
    }
    else
    {
      printf("%d=");
    }
  }
  Printf("%d", sum);
  return(0);
  getch();
}
```

Tracing

I st pass

123

123 != 0 True.

rem = 123 % 10 ⇒ 3

num = 123/10 ⇒ 12.

sum = 0+3 ⇒ 3

II pass

12

rem = 12 % 10 ⇒ 2

num = 12/10 ⇒ 2

sum = 3+2 ⇒ 5

Now num = 1

III pass

rem = 1 % 10 ⇒ 0

num = 1/10 ⇒ 0

sum = 5+1 ⇒ 6

∴ Sum of 123 = 6

For Loop.

For Loop is provides mechanism to repeat set of actions a specific number of times.

Syntax:

```
for(initialization; condition; inc/dec/update)
{
    //Code to be Executed
      repeatidly untill
      condition is satisfied.
}
```

Example

Prog to find factorial of a number.

```
#include<stdio.h>
int main()
{
    int n, fact=1;
    Printf("Enter +ve number");
    Scanf("%.d", &n);
     if (n< 0)
      {
        Printf("Factorial not defined
                    for negative number");
      }
        Else.
        {
          for(i=1; i<=n; i++)
          {fact = fact * i;.
          }
      Printf("Factorial of a given number is %.d", fact);
    }
  return(0);
}
```