UNIT-II

Control Structures : Simple sequential programs Conditional Statements (if, if-else, switch), Loops (for, while, do-while) Break and Continue.

Control Structure:

- In C program, statements are normally executed sequentially in order in which they appear. But sometimes, we have to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are satisfied
- For this, C language provides control statements(or decision making) statements which alter the flow of execution and provide better control to the programmer on the flow of execution. They are two types
  - Selection/branching/decision making statement
  - Repetition/ looping/iterative

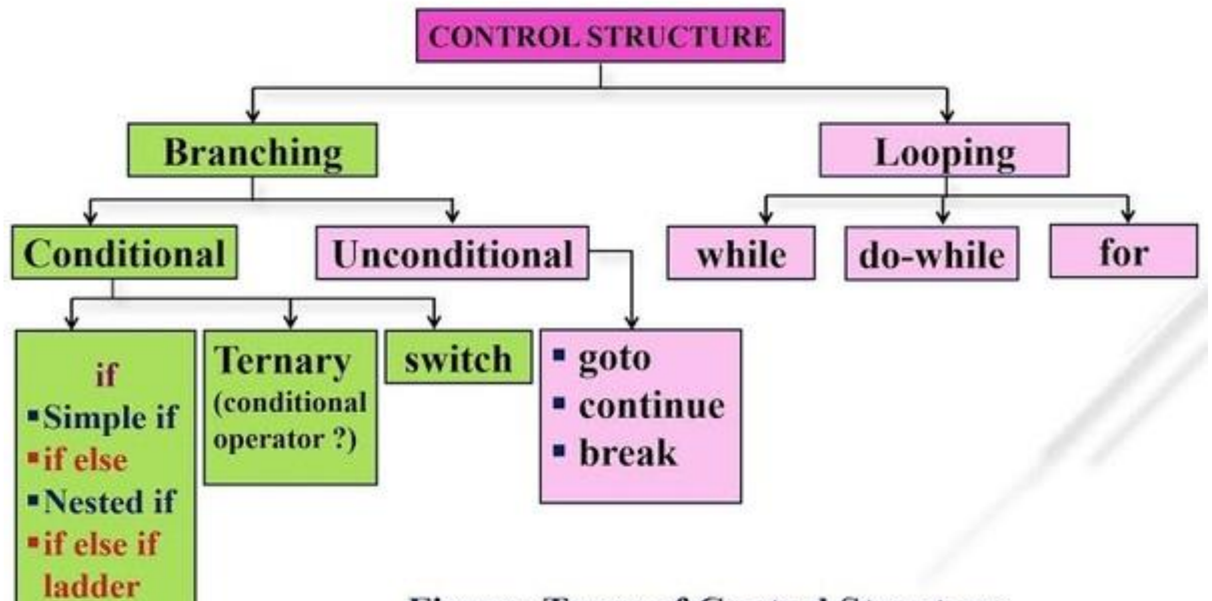


Figure: Types of Control Structure

<u>CONDITIONAL CONTROL STATEMENTS:</u>

Decision making (or) branching (or) selection statements:

Decision-making statements are the statements that are used to verify a given condition and decide whether a block of statements gets executed or not based on the condition result. The result is in the form of two expressions true or false.
There are two decision-making statements

1. **if statement**
2. **switch statement**

In C, if statement is used to make decisions based on a condition. The if statement verifies the given condition and decides whether a block of statements are executed or not based on the condition result.

if statement is classified into four types as follows...

1. **Simple if statement**
2. **if-else statement**
3. **Nested if statement**
4. **if-else-if statement (if-else ladder)**

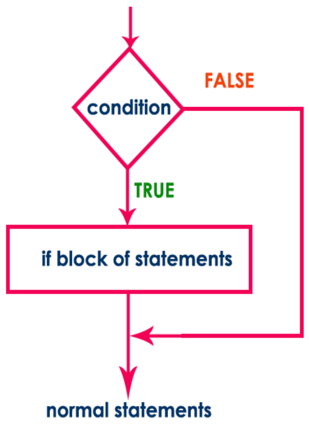
Simple if statement

- if is a one way selection .
- Simple if statement is used to verify the given condition and executes the block of statements based on the condition result.
- The simple if statement evaluates specified condition. If it is TRUE, it executes the block of statements. If the condition is FALSE, it skips (ignores)the block of statements.

Syntax

```
if ( condition )
{
    ....
    block of statements;
    ....
}
```

Execution flow diagram



Example program to test whether the given number is divisible by5:

```
#include<stdio.h>
void main()
{
  int n ;
  printf("Enter any integer number: ") ;
  scanf("%d", &n) ;
  if ( n%5 == 0 )
     printf("Given number is divisible by 5\n") ;
  printf("statement does not belong to if!!!") ;
}
```

Output1:

```
Enter any integer number: 50
Given number is divisible by 5
statement does not belong to if!!!
```

Output2:

```
Enter any integer number: 33
statement does not belong to if!!!
```

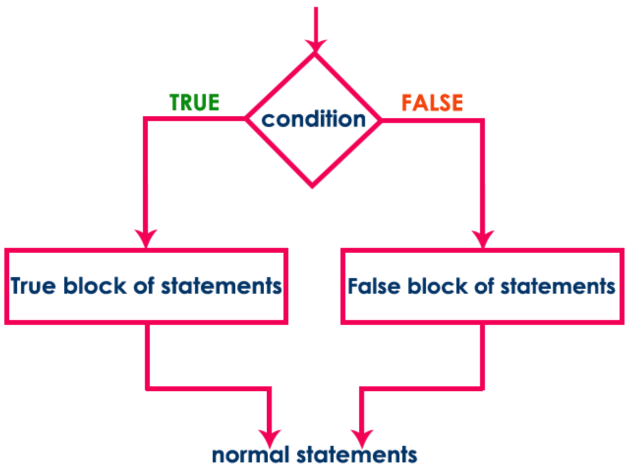
if-else statement

- if else is a two way selection.
- The if-else statement is used to verify the given condition and executes only one out of the two blocks of statements based on the condition result.
- The if-else statement evaluates the specified condition. If it is TRUE, it executes a block of statements (True block). If the condition is FALSE, it executes another block of statements (False block).

Syntax

```
if ( condition )
{
   ....
   True block of statements;
   ....
}
else
{
   ....
   False block of statements;
   ....
}
```

Execution flow diagram



Example program to test whether the given number is even or odd:

```
#include<stdio.h>
void main()
{
  int n ;
  printf("Enter any integer number: ") ;
  scanf("%d", &n) ;
  if ( n%2 == 0 )
    printf("Given number is EVEN\n") ;
  else
    printf("Given number is ODD\n") ;
}
```

Output1:

```
Enter any integer number: 20
Given number is EVEN
```

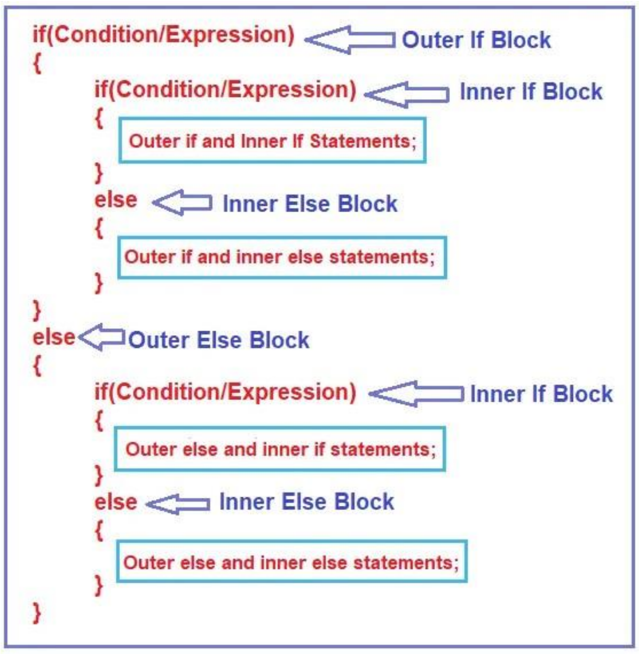
Output2:

```
Enter any integer number: 15
Given number is ODD
```

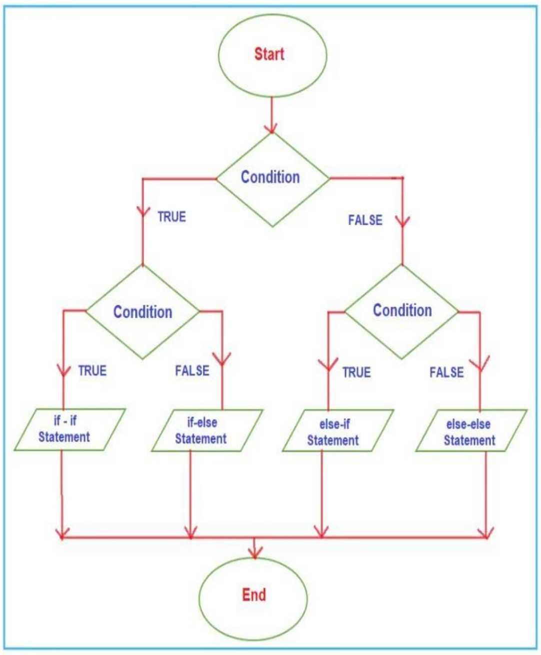
Nested if statement: Writing a if statement inside another if statement is called nested if statement.

Syntax

```
if ( condition1 )
{
    if ( condition2 )
    {
        ....
        True block of statements 1;
    }
    ....
}
else
{
     False block of condition1;
}
```



```
if(Condition/Expression)
{
        if(Condition/Expression)
        {
          Outer if and Inner If Statements;
        }
        else
        {
          Outer if and inner else statements;
        }
}
else
{
        if(Condition/Expression)
        {
          Outer else and inner if statements;
        }
        else
        {
          Outer else and inner else statements;
        }
}
```



1Example program to test whether the given number is even or odd if it is below 100.:

```
#include<stdio.h>
void main()
{
   int n ;
   printf("Enter any integer number: ") ;
   scanf("%d", &n) ;
   if ( n < 100 )
   {
      printf("Given number is below 100\n") ;
      if( n%2 == 0)
          printf("And it is EVEN") ;
      else
          printf("And it is ODD") ;
   }
   else
       printf("Given number is not below 100") ;
}
```

Output1:

```
Enter any integer number: 60
Given number is below 100
And it is EVEN
```

Output2:

```
Enter any integer number: 150
Given number is not below 100
```

2.**program to check the given numbers are geater,lessthan or equal using nested if?**

```
#include<stdio.h>
int main()
{
            int a,b;
    printf("Enter any two integers");
    scanf("%d%d",&a,&b);
    if(a<=b)
    {
                if (a<b)
                        printf(" %d is less than %d",a,b);
                else
                        printf("%d is equal to %d", a,b);
    }
    else
    {
        printf("%d is greater than %d", a,b );
    }
return 0;
}
```

Output1:

```
Enter any two integers 20 30
20 is less than 30
```

Output2:

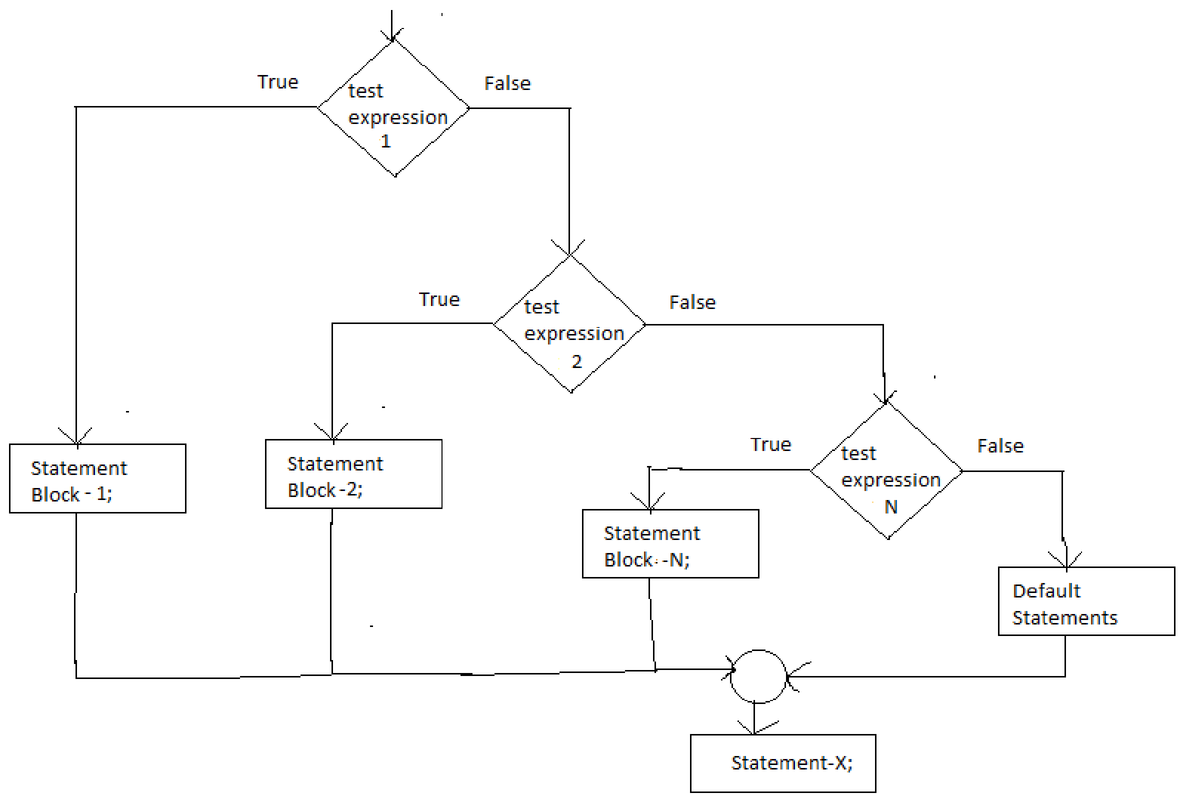
```
Enter any two integers 20 20
20 is equal to 20
```

if-else-if statement (or) else-if ladder (or) if-else chain:

- If we are having different-different test conditions with different-different statements thenfor these kind of programming we need else if ladder.
- else if ladder is not interdependent to any other statements or any other test conditions.

syntax :

```
if(condition 1)
 {
        Statement block 1;
 }
 else if(condition 2)
 {
        Statement block 2;
 }
 else if(condition 3)
 {
        Statement block 3;
 }
 ………..
else
 {
        Statement block x;
 }
 Statement y;
```



- This constructor is known as else-if ladder. This condition is evaluated from top to bottom.
- The condition is true the statement blocks is executed the control is transferred to the next statement when all the ‘n’ conditions become false then the final else containing the default statement(else block statements) will be executed.

1.Example program to find the largest of three numbers:

```
#include<stdio.h>
void main()
{
  int a, b, c ;
  printf("Enter any three integer numbers: ") ;
  scanf("%d%d%d", &a, &b, &c) ;
  if( a>=b && a>=c)
      printf("%d is the largest number", a) ;
   else if (b>=a && b>=c)
      printf("%d is the largest number", b) ;
   else
      printf("%d is the largest number", c) ;
}
```

Output:

```
Enter any three integer numbers:
50
10
90
90 is the largest number
```

2.Program to calculate the average of student marks and print the class using else if

```
#include<stdio.h>
int main( )
{
        int a,b,c,sum;
        float avg;
        printf("enter the marks");
        scanf("%d%d%d",&a,&b,&c);
        sum=a+b+c;
        avg=sum/3;
        printf("avg=%f\n",avg);
        if(avg>=70)
                printf("first class with distinction");
        else if(avg>=60&&avg<=69)
                printf("first class");
        else if(avg>=50&&avg<=59)
                printf("second lass");
        else if(avg>=35&&avg<=49)
                printf("pass");
        else
                printf("fail");
        return 0;
}
```

Output1:

```
enter the marks 90 60 70
avg=73.000000
first class with distinction
```

Output2:

```
enter the marks 40 30 40
40 30 40
avg=36.000000
pass
```

NOTE(Important points to be remembered):

When we use a conditional control statement like if statement, the condition might be an expression evaluated to a numerical value, a variable or a direct numerical value. **If the expression value or direct value is zero the condition becomes FALSE otherwise becomes TRUE**.

- **if(10)** - is TRUE
- **if(x)** - is FALSE if x value is zero otherwise TRUE
- **if(a+b)** - is FALSE if a+b value is zero otherwise TRUE
- **if(a = 99)** - is TRUE because a value is non-zero
- **if(10, 5, 0)** - is FALSE because it considers last value
- **if(0)** - is FALSE
- **if(a=10, b=15, c=0)** - is FALSE because last value is zero

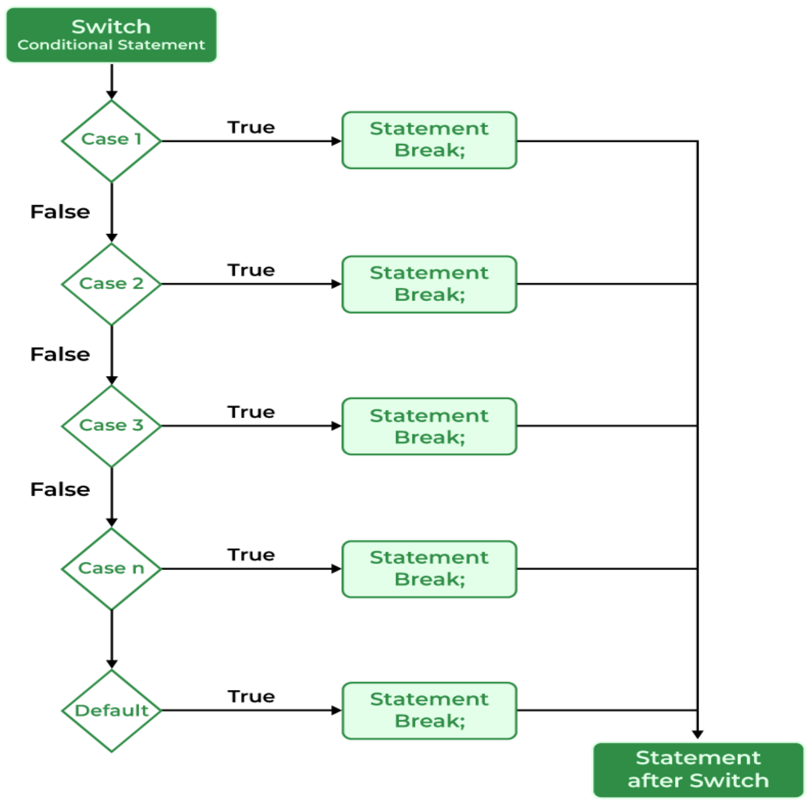
switch (multi way selection):

- Decision making are needed when the program encounters the situation to choose a particular statement among many statements, if a program has to choose one block of statement among many alternative, nested-if-else can be used but this makes programming logic complex.
- Consider a situation in which we have many options out of which we need to select only one option that is to be executed. Such kind of problems can be solved using **nested if** statement. But as the number of options increases, the complexity of the program also gets increased. This type of problem can be solved very easily using a **switch** statement.
- Instead of else-if ladder C has a built-in multi-way decision statement known as a **switch.**
- **In C, the switch case statement is used for executing one condition from multiple conditions. It is similar to an if-else-if ladder.**
- The switch statement consists of conditional-based cases and a default case.

Syntax:

```
switch(expression)
{
case value1: statement_1;
                    break;
case value2: statement_2;
                    break;
……..
case value_n: statement_n;
                    break;
default: default_statement;
                    break;
}
Statement x;
```

Flow chart



The switch statement contains one or more cases and each case has a value associated with it. At first switch statement compares the first case value with the switchValue, if it gets matched the execution starts from the first case. If it doesn't match the switch statement compares the second case value with the switchValue and if it is matched the execution starts from the second case. This process continues until it finds a match. If no case value matches with the switchValue specified in the switch statement, then a special case called **default** is executed.

When a case value matches with the switchValue, the execution starts from that particular case. This execution flow continues with the next case statements also. To avoid this, we use the "**break**" statement at the end of each case. That means the **break** statement is used to terminate the switch statement. However, it is optional.

When we use switch statement, we must follow the following rules:.

- Both **switch** and **case** are keywords so they must be used only in lower case letters.
- The data type of case value and the value specified in the switch statement must be the same.
- **switch and case values** must be either ***integer or character but not float or string.***
- A switch statement can contain any number of cases.
- The keyword **case** and its value must be separated with a white space.
- The case values need not be defined in sequence, they can be in any order.
- The **default** case is optional and it can be defined anywhere inside the switch statement.
- The switch value might be direct, a variable or an expression.

1.Simple program to print week days using switch:

```
#include <stdio.h>
int main()
{
        int day;
        printf("Enter weekday number(1:7)");
        scanf("%d",&day);
        switch(day)
        {
        case 1: printf("Sunday");
                        break;
        case 2: printf("Monday");
                        break;
        case 3: printf("Tuesday");
                        break;
        case 4: printf("Wednesday");
                        break;
        case 5: printf("Thursday");
                        break;
        case 6: printf("Friday");
                        break;
        case 7: printf("Saturday");
                        break;
        default : printf("Invalid number\n");
                        break;
        }
        return 0;
}
```

Output:

```
Enter weekday number(1:7)
2
Monday
```

2.Example Program | Display pressed digit in words :

```
#include<stdio.h>
int main()
{
  int n ;
  printf("Enter any digit: ") ;
  scanf("%d", &n) ;
  switch( n )
  {
        case 0: printf("ZERO") ;
                break ;
        case 1: printf("ONE") ;
                break ;
        case 2: printf("TWO") ;
                break ;
        case 3: printf("THREE") ;
                break ;

        case 4: printf("FOUR") ;
                break ;
        case 5: printf("FIVE") ;
                break ;
        case 6: printf("SIX") ;
                break ;
        case 7: printf("SEVEN") ;
                break ;
        case 8: printf("EIGHT") ;
                break ;
        case 9: printf("NINE") ;
                break ;
        default: printf("Not a Digit") ;
                break;
  }
  return 0;
}
```

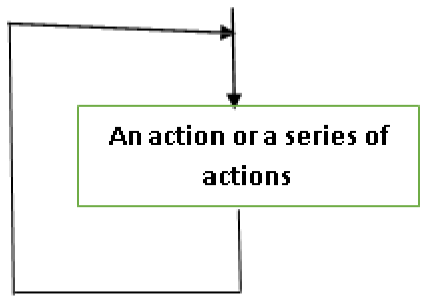
Output:

```
Enter any digit: 5
FIVE
```

<u>Looping statements(or)Repetition statements:</u>

Consider a situation in which we execute a single statement or block of statements repeatedly for the required number of times. Such kind of problems can be solved using **looping** statements in C. For example, assume a situation where we print a message 100 times. If we want to perform that task without using looping statements, we have to either write 100 printf statements or we have to write the same message 100 times in a single printf statement. Both are complex methods. The same task can be performed very easily using looping statements.

- Loop causes program to execute the certain block of code repeatedly until some conditions are satisfied.
- Each repetition is also referred as an iteration or pass through the loop.
- For example if we want to execute some code for 10 times. We can perform it by writingthat codes only one time and repeat the execution 10 times using loop.

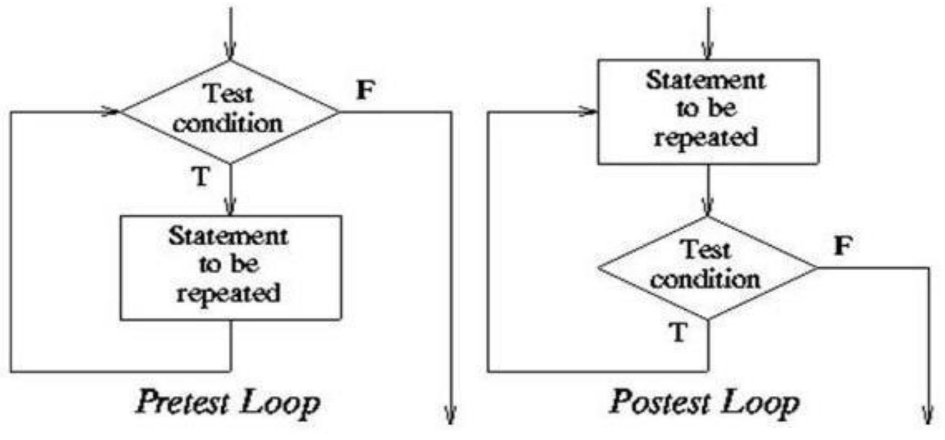


Loops

The looping statements are used to execute a single statement or block of statements repeatedly until the given condition is FALSE.

Pre-test and Post-test Loops

- To check the loop control expression either before or after each iteration of the loop.
- In a pre-test loop, the **condition is checked before the beginning** of each iteration.
- If the condition is true, the code is executed, else the loop is terminated.
- In a post-test loop the actions are executed. Then the **control expression is tested**(**Last**).
- If it is true, a new iteration is started; otherwise, the loop terminates. Here the actions areexecuted atleast once.



Loop comparisions:

Pre test loop	**post test loop**
Initialization -1	Initialization- 1
No.of tests- n+1	No.of tests- n
Action executed-n	Action executed-n
Updated executed-n	Updated executed-n
Minimum iterations-0	Minimum iterations-1

Initialization and Updating

- In loop control expression, two other process, initialization and updating are associatedwith almost all loops.

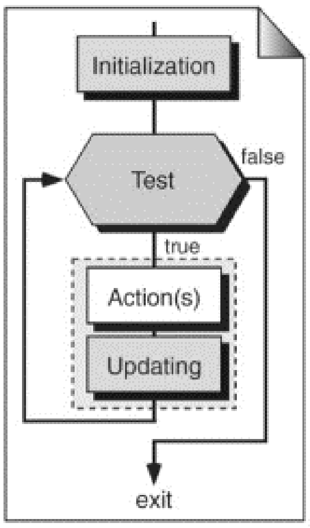
Loop initialization

- Before a loop start initialization must be done. It sets the stage for the loop action.
- Initialization may be implicit or explicit.

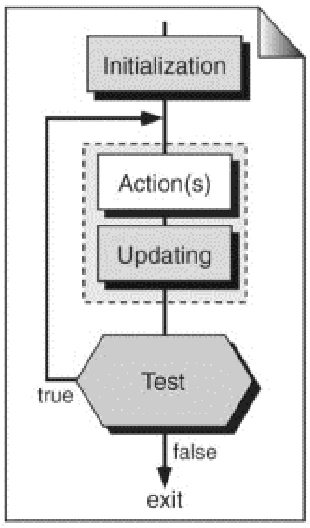
Loop update

- It is necessary to change the condition true to false, otherwise the loop will be executedindefinitely.

- The actions that cause these changes are known as loop update. Updating is done in eachiteration.
- If the body of the loop is repeated 'n' times, the updating is also done 'n' times.



(a) Pretest Loop

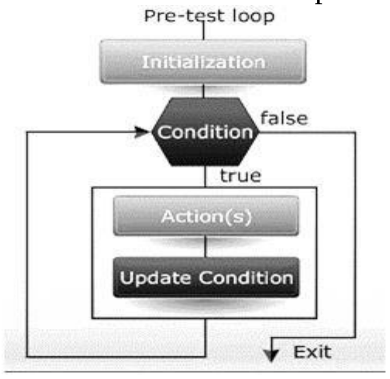


(b) Post-test Loop

Event and Counter Controlled Loops

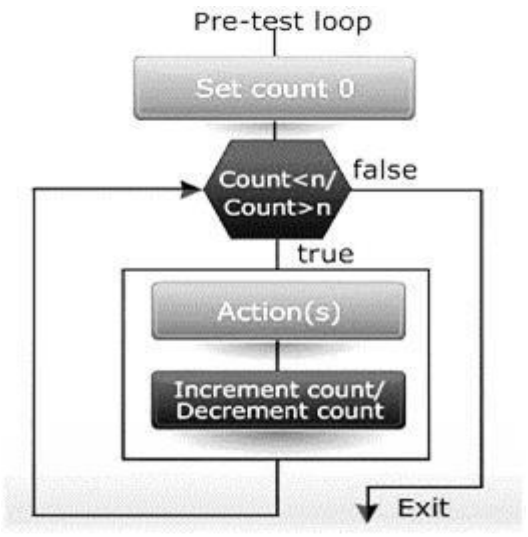
Event Controlled Loops:

- In an event-controlled an event that happens in the loops execution block that changes the loops control expression from true to false.
- The program can update the loop control expression explicitly or implicitly.
- In an event-controlled loop pre-test loop, the condition is tested first.If the condition is true, then the action and iterations takes place. If the condition is false, the loop terminates.



Counter-controlled Loops

- In a counter- controlled loop, the number of loop iterations can be controlled.
- In such a loop, we use a counter, which we must initialize, update and test.
- The number the loop assigned to the counter does not need to be a constant value, to update, we can increment or decrements the counter.



Three looping statements...

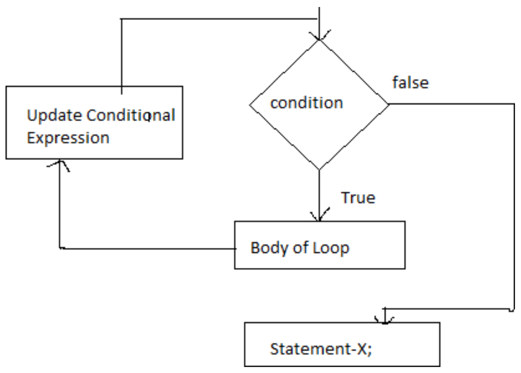
- **while /top testes/ entry loop/pretest**
- **do-while/bottom tested/post-tested loop/exit loop**
- **for loop/pretest**

while Statement

The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE. The while statement is also known as **Entry control looping statement**.

Syntax:

```
while(condition)
{
        Body of the while loop;
        Update expressions;
}
Statement x;
```



- It is an **entry-controlled loop/top checking loop**. In while loop, a condition is evaluated before processing a body of the loop.
- If a condition is true, then the body of a loop is executed. After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed <u>until the condition becomes false</u>.

- Once the condition becomes false, the control goes out of the loop. After exiting the loop,the control goes to the statements which are immediately after the loop.
- The body of a loop can contain more than one statement. If it contains only one statement, then the curly braces are not compulsory.

Ex1 Program to print series from 1 to 10:

```
#include<stdio.h>
int main()
{
        int i=1;        //initializing the variable
        while(i<=10)  //while loop with condition
        {
                printf("%d\t",i);
                  i++;            //incrementing operation
        }
        return 0;
}
```

Output:

```
1       2       3       4       5       6       7       8       9       10
```

Note:

- **while** is a keyword so it must be used only in lower case letters.
- If the condition contains a variable, it must be assigned a value before it is used.
- The value of the variable used in condition must be modified according to the requirement inside the while block.
- In a while statement, the condition may be a direct integer value, a variable or a condition.
- A while statement can be an empty statement.

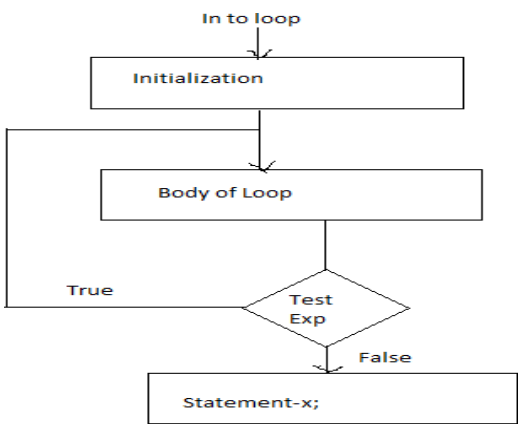
do-while loop:

- A do-while loop is **similar to the while loop** except that the condition is always executedafter the body of a loop.
- The do-while statement is used to execute a single statement or block of statements repeatedly as long as given the condition is TRUE. The do-while statement is also known as the **Exit control looping statement**.
- It is also called an **exit-controlled loop**. The basic format of do-while loop is as follows:

Syntax:

```
do
{
        Body of the loop;
        Update
        expressions;
}
while(condition);
Statement x;
```

- In the do-while loop, the body of a loop is always executed at least once.
- After the body executed, then it checks the condition. If the condition is true, then it willagain execute the body of a loop; otherwise control is transferred out of the loop.
- Once the control goes out of the loop the statements which are immediately after the loopis executed.



Disadvantage

- The using of do-while loop is that it always executes at least once, even if the user enterssome invalid data.

Program to print a table of number 5 using do while loop.

```
#include<stdio.h>
int main()
{
        int  i=1;         //initializing  the  variable

         do               //do-while loop
        {
                printf("%d\t",5*i);
                i++;      //incrementing operation
        }while(i<=10);
        return 0;
}
```

Output:

```
5         10       15       20       25       30       35       40       45       50
```

For loop:

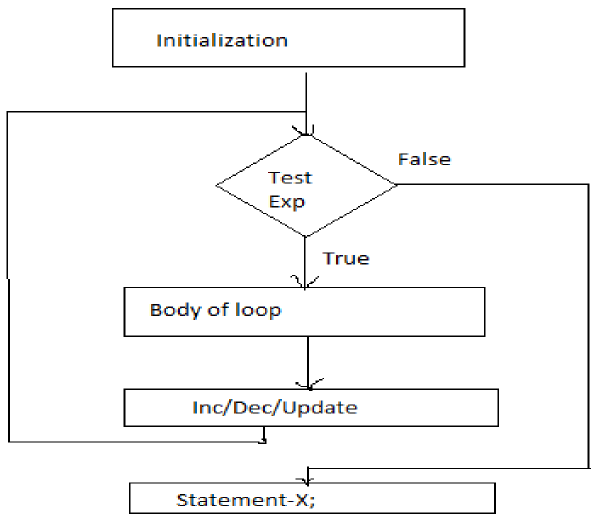
- A for loop is a more efficient loop structure in 'C' programming.
- At first, the for statement executes **initialization** followed by **condition** evaluation. If the condition is evaluated to TRUE, the single statement or block of statements of for statement are executed. Once the execution gets completed, the **modification** statement is executed and again the condition is evaluated. If it is TRUE, again the same statements are executed. The same process is repeated until the condition is evaluated to FALSE. Whenever the condition is evaluated to FALSE, the execution control moves out of the for block.

Syntax:

```
for(initialization; condition, increment/decrement)
{
        Code to be executed;
}
Statement x;
```

- The **initialization statement** of the **for loop** is performed only once, at the beginning.
- The **condition** is a Boolean expression that tests and compares the counter to a fixedvalue after each iteration, terminates the for loop when false is returned.
- The **incrementation/decrementation** increases (or decreases) the counter by a set value.

Flow chart:



Program to illustrate the use of a simple for loop:

```
#include<stdio.h>
int main()
{
        int i;
        for(i=10; i>=1;i--)     //for loop to print numbers in reverse order
        {
                printf("%d\t",i);        //to print the number
        }
        return 0;
}
```

Output:

```
10      9       8       7       6       5       4       3       2       10
```

2.Example Program | Program to display even numbers upto 10.

```
#include<stdio.h>
void main()
{
  int n = 1;
  printf("Even numbers upto 10\n");
  while( n <= 10 )
  {
    if( n%2 == 0)
      printf("%d\t", n) ;
    n++ ;
  }
}
```

Output:

```
Even numbers upto 10
2  4 6 8 10
```

Note:
When we use for statement, we must follow the following...

- **for** is a keyword so it must be used only in lower case letters.
- Every for statement must be provided with initialization, condition, and modification (They can be empty but must be separated with ";") Ex: **for (; ;)** or **for (; condition ; modification)** or **for (; condition ;)**
- In for statement, the condition may be a direct integer value, a variable or a condition.
- The for statement can be an empty statement.

Comma Expression

The for loop can have multiple expressions separated by commas in each part.

For example:

```
for (x = 0, y = num; x < y; i++, y--)
{
  statements;
}
```

Also, we can skip the initial value expression, condition and/or increment by adding a semicolon.

For example:

```
int i=0;

int max=10;
for ( ; i < max; i++)
{
        Printf("%d\n",i);
}
```

Nested loops:

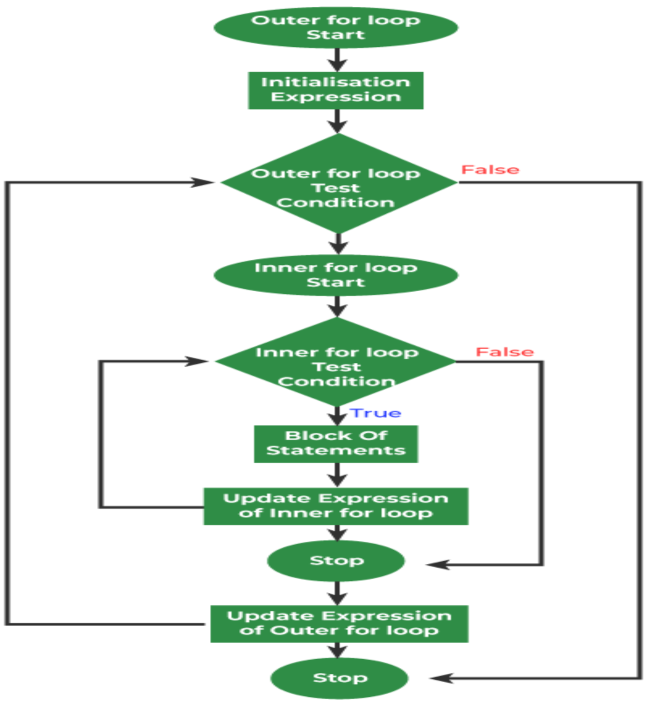
Nested for loop:

- Loops can also be nested where there is an outer loop and an inner loop.
- Nested for loop refers to any type of loop that is defined inside a ‘for’ loop.
- For each iteration of the outer loop, the inner loop repeats its entire cycle.

Syntax:

```
for ( initialization; condition; update )
{
     for ( initialization; condition; update )
     {

             // statement of inside loop
      }
     // statement of outer loop
}
```


Outer for loop Start
Initialisation Expression
Outer for loop Test Condition
False
Inner for loop Start
Inner for loop Test Condition
False
True
Block Of Statements
Update Expression of Inner for loop
Stop
Update Expression of Outer for loop
Stop

1.Program to illustrate the use of a simple for loop:

```
#include <stdio.h>
 int main()
{
   int i = 0;
   for (int i = 0; i < 5; i++)
  {
      for (int j = 0; j < 5; j++)
     {
         printf("* ");
      }
      printf("\n");
   }
   return 0;
}
```

Output:

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

2.Program to illustrate the use of a simple for loop:

```
#include<stdio.h>
int main()
{
int n,i,j;
printf("Enter the rows:");
scanf("%d",&n);
for (i=1;i<=n;i++)
{
    for(j=1;j<=i;j++)
    printf("*");
   printf("\n");
}
return 0;
}
```

Output:

```
 Enter the rows:7
*
**
***
****
*****
******
*******
```

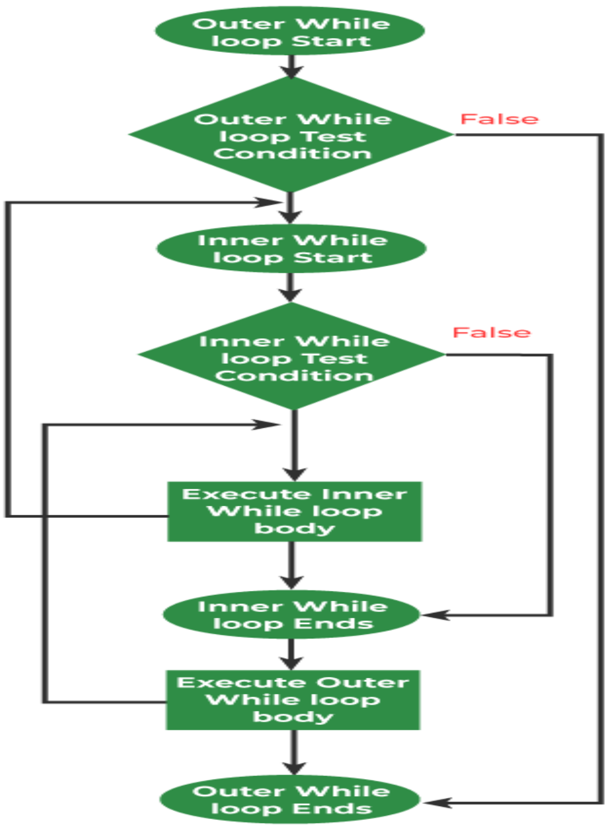
Nested while Loop

A nested while loop refers to any type of loop that is defined inside a 'while' loop. Below is the equivalent flow diagram for nested 'while' loops:

Syntax:

```
while(condition)
 {
      while(condition)
     {
           // statement of inside loop
     }

     // statement of outer loop
}
```


Outer While loop Start
Outer While loop Test Condition
False
Inner While loop Start
Inner While loop Test Condition
False
Execute Inner While loop body
Inner While loop Ends
Execute Outer While loop body
Outer While loop Ends

// C program to print pattern using nested while loops

```
#include <stdio.h>
int main()
{
   int n = 5;
   printf("Pattern Printing using Nested While loop");
   int i = 1;
   while (i <= n)
  {
      printf("\n");
      int j = 1;
      while (j <= i)
     {
         printf("%d ", j);
         j = j + 1;
      }
      i = i + 1;
   }
   return 0;
}
```

Output:

```
Pattern Printing using Nested While loop
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

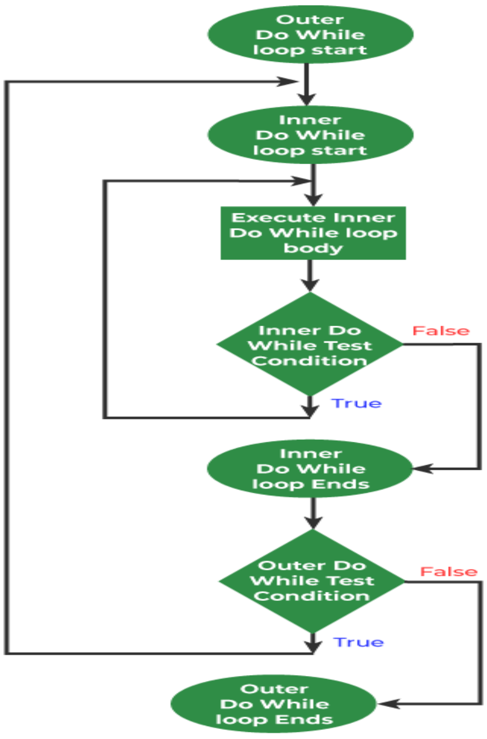
Nested do-while Loop

A nested do-while loop refers to any type of loop that is defined inside a do-while loop. Below is the equivalent flow diagram for nested 'do-while' loops:

Syntax:

```
do
{
    do
   {
          // statement of inside loop
    }while(condition);

  // statement of outer loop
}while(condition);
```


Outer Do While loop start
Inner Do While loop start
Execute Inner Do While loop body
Inner Do While Test Condition
False
True
Inner Do While loop Ends
Outer Do While Test Condition
False
True
Outer Do While loop Ends

Note: There is no rule that a loop must be nested inside its own type. In fact, there can be any type of loop nested inside any type and to any level.

Syntax:

```
do
{
  while(condition)
 {
      for ( initialization; condition; increment )
     {

       // statement of inside for loop
      }
     // statement of inside while loop
   }

  // statement of outer do-while loop
}while(condition);
```

UNCONDITIONAL CONTROL STATEMENTS

break, continue and goto in C

In c, there are control statements that do not need any condition to control the program execution flow. These control statements are called as **unconditional control statements**. C programming language provides the following unconditional control statements...

- **break**
- **continue**
- **goto**

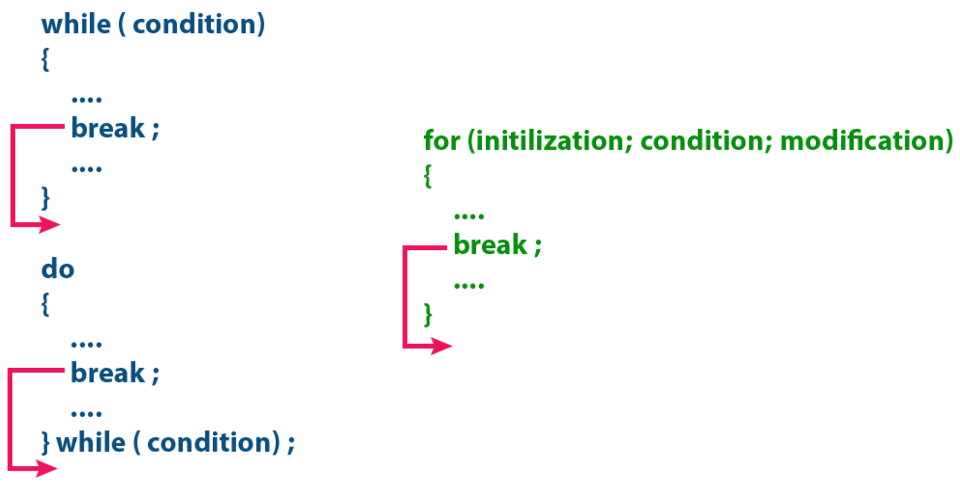
The above three statements do not need any condition to control the program execution flow.

break statement

In C, the **break statement** is used to perform the following two things...

1. **break statement is used to terminate the switch case statement**
2. **break statement is also used to terminate looping statements like while, do-while and for.**

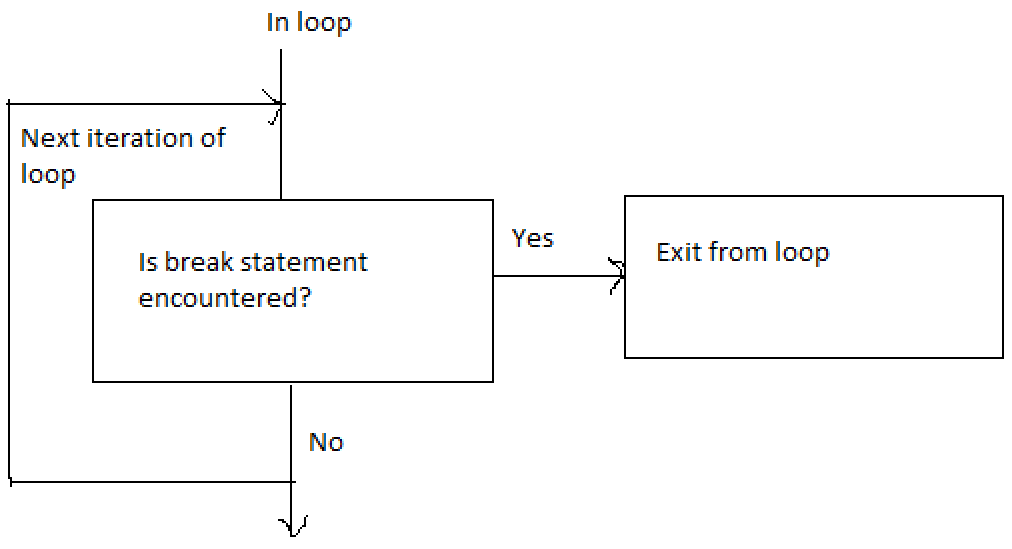
When a **break** statement is encountered inside the switch case statement, the execution control moves out of the switch statement directly. For example, consider the following program.



```
while ( condition)
{
    ....
    break ;
    ....
}

do
{
    ....
    break ;
    ....
} while ( condition) ;

for (initilization; condition; modification)
{
    ....
    break ;
    ....
}
```



Example program using break statement:

```
#include <stdio.h>
int main()
{
  int i;
  for (i = 0; i < 10; i++)
 {
    if (i == 4)
   {
      break;
    }
    printf("%d\t", i);
  }
     return 0;
}
```

Output:

```
0	1	2	3
```

Example2:

```
#include<stdio.h>
 void main()
{
   int n1, n2, result ;
   char operator;
    printf("Enter any two integer numbers: ") ;
   scanf("%d%d", &n1, &n2) ;
   printf("Please enter any arithmetic operator: ");
   operator = getchar();
   switch(operator)
   {
       case '+': result = n1 + n2 ;
                  printf("Addition = %d", result) ;
                  break;
       case '-': result = n1 - n2 ;
                  printf("Subtraction = %d", result) ;
                  break;
       case '*': result = n1 * n2 ;
                  printf("Multiplication = %d", result) ;
                  break;
       case '/': result = n1 / n2 ;
                  printf("Division = %d", result) ;
                  break;
       case '%': result = n1 % n2 ;
                  printf("Remainder = %d", result) ;
                  break;
       default: printf("\nWrong selection!!!") ;
   }
  }
```

Output:

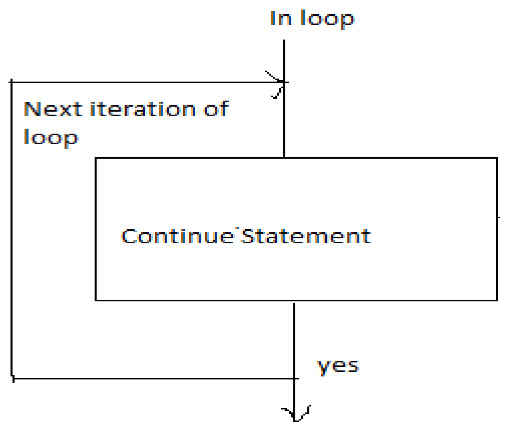
```
Enter any two integer numbers:  10 20
Please enter any arithmetic operator: * Multiplication =200
```

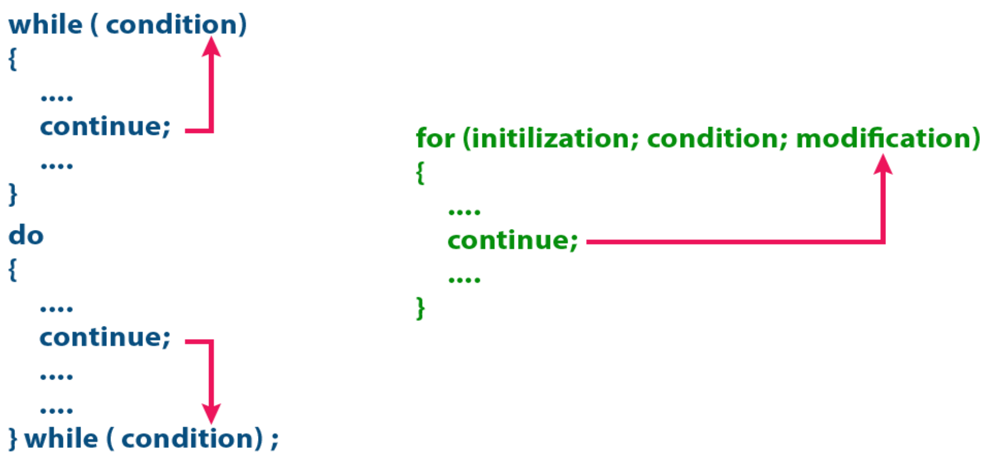
continue statement

The **continue** statement is used to move the program execution control to the beginning of the looping statement.

When the **continue** statement is encountered in a looping statement, the execution control skips the rest of the statements in the looping block and directly jumps to the beginning of the loop. The **continue** statement can be used with looping statements like while, do-while and for.

When we use **continue** statement with **while** and **do-while** statements the execution control directly jumps to the condition. When we use **continue** statement with **for** statement the execution control directly jumps to the modification portion (increment/decrement/any modification) of the for loop.





```
while ( condition)
{
    ....
    continue;
    ....
}
do
{
    ....
    continue;
    ....
    ....
} while ( condition) ;

for (initilization; condition; modification)
{
    ....
    continue;
    ....
}
```

Example program using continue statement:

```
#include <stdio.h>
int main()
{
  int i;
  for (i = 0; i < 10; i++)
  {
     if (i == 4)
     {
       continue;
     }
     printf("%d\t", i);
  }
  return 0;
}
```

Output: 0   1   2   3   5   6   7   8   9

goto Statement

- **goto** statement is used to jump from one line to another line in the program.
- Using **goto** statement we can jump from top to bottom or bottom to top.
- To jump from one line to another line, the goto statement requires a **label**. Label is a name given to the instruction or line in the program followed by colon.
- When we use a **goto** statement in the program, the execution control **directly jumps to** the line with the specified label.

Syntax: **goto label;**

```
Example:    goto label;
              ………….
              ………….
              label:
                      statements;
```

Program to illustrate the use of goto statement

```
#include<stdio.h>
void main()
{
    printf("We are at first printf statement!!!\n");
    goto last;
    printf("We are at second printf statement!!!\n");
    printf("We are at third printf statement!!!\n");
    last:
        printf("We are at last printf statement!!!\n");
}
```

Output:

```
We are at first printf statement!!!
We are at last printf statement!!!
```

NOTE:**Important points to remember:**

When we use break, continue and goto statements, we must follow the following...

- The **break** is a keyword so it must be used only in lower case letters.
- The **break** statement can not be used with **if** statement.
- The **break** statement can be used only in switch case and looping statements.
- The **break** statement can be used with **if** statement, only if that **if statement** is written inside the switch case or looping statements.
- The **continue** is a keyword so it must be used only in lower case letters.
- The **continue** statement is used only within **looping statements**.
- The **continue** statement can be used with **if** statement, only if that **if statement** is written inside the looping statements.
- The **goto** is a keyword so it must be used only in lower case letters.
- The **goto** statement must require a **label**.
- The **goto** statement can be used with any statement like if, switch, while, do-while, and for, etc.

More Standard Functions

- The standard functions are built-in functions. In C programming language, the standard functions are declared in header files and defined in .dll files.
- The standard functions can be defined as "the readymade functions defined by the systemto make coding more easy".
- The standard functions are also called as **library functions** or **pre-defined functions**.
- In C when we use standard functions, we must include the respective header file using **#include** statement.
- For example, the function **printf()** is defined in header file **stdio.h** (Standard Input Output header file). When we use **printf()** in our program, we must include **stdio.h** header file using **#include<stdio.h>** statement.
- C Programming Language provides the following header files with standard functions.

Header file	Purpose	Example function
stdio.h	Provides function to perform standard I/O operations	printf(),scanf()
conio.h	Provides function to perform consol I/O operations	clrscr(),getch()
math.h	Provides function to perform mathematical operations	sqrt(),pow()
string.h	Provides function to handle string data values	strlen(),strcpy()
stdlib.h	Provides function to perform general functions	calloc(), malloc()
time.h	Provides functions to perform operations on time and Date	time(), localtime()
ctype.h	Provides functions to perform - testing and mapping of character data values	isalpha(), islower()
setjmp.h	Provides functions that are used in function calls	setjump(), longjump()
signal.h	Provides functions to handle signals during program Execution	signal(), raise()
assert.h	Provides Macro that is used to verify assumptions made by the program	assert()
locale.h	Defines the location specific settings such as date formats and currency symbols	setlocale()
stdarg.h	Used to get the arguments in a function if the arguments are not specified by the function	va_start(), va_end(), va_arg()
errno.h	Provides macros to handle the system calls	Error, errno
graphics.h	Provides functions to draw graphics.	circle(), rectangle()
float.h	Provides constants related to floating point data values	
stddef.h	Defines various variable types	
limits.h	Defines the maximum and minimum values of various variable types like char, int and long	

1.Write a C Program to print the Factorial of a given number

```
#include<stdio.h>
main( )
{
int n,i;
int fact=1;
printf("Enter a number");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
   fact=fact*i;
 }
printf("Factorial of %d is %d\n",n,fact);
}
```

Output:

```
Enter a number5

Factorial of 5 is 120
```

2.Write a C program to Check whether the given number is Armstrong or not.

A positive integer is called an Armstrong number (of order n) if
abcd... = an + bn + cn + dn +
In the case of an Armstrong number of 3 digits, the sum of cubes of each digit is equal to the number itself. For example, 153 is an Armstrong number because

153 = 1*1*1 + 5*5*5 + 3*3*3

Program:

```
#include<stdio.h>
main( )
{
int n,rem,sum=0,temp;
printf("Enter a Number:");
scanf("%d",&n);
temp=n;
while(n!=0)
{
   rem=n%10;
   sum=sum+(rem*rem*rem);
   n=n/10;
}
if(sum==temp)
printf("%d is Armstrong\n",temp);
}
```

Output:

```
Enter a Number:153
153 is Armstrong
```

3.Write a C program to Check whether the given number is palindrome or not.

An integer is a palindrome if the reverse of that number is equal to the original number.

Program:

```
#include <stdio.h>
int main()
{
   int n, rev= 0, rem, temp;
   printf("Enter an integer: ");
   scanf("%d", &n);
   temp= n;
   while (n != 0)
   {
      rem = n % 10;
      rev = rev * 10 + rem;
      n =n/10;
   }
   if (temp== rev)
      printf("%d is a palindrome.", temp);
   else
      printf("%d is not a palindrome.", temp);
   return 0;
}
```

Outpu1t:

```
Enter an integer: 123
123 is not a palindrome.
```

Output2:

```
Enter an integer: 151
151 is a palindrome.
```

4.Write a C program to display fibonacii sequence

The Fibonacci sequence is a sequence where the next term is the sum of the previous two terms. The first two terms of the Fibonacci sequence are 0 followed by 1.
The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21

Program:

```
#include <stdio.h>
int main()
{
  int i, n;
  int t1 = 0, t2 = 1;
  int t3 = t1 + t2;
  printf("Enter the number of terms: ");
  scanf("%d", &n);
   printf("Fibonacci Series: %d, %d, ", t1, t2);
  for (i = 3; i <= n; ++i)
  {
    printf("%d, ", t3);
    t1 = t2;
    t2 = t3;
    t3 = t1 + t2;
  }
  return 0;
}
```

Output1:

```
Enter the number of terms: 5
Fibonacci Series: 0, 1, 1, 2, 3,
```

Output2:

```
Enter the number of terms: 8
Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13,
```