PROGRAMMING FUNDAMENTALS

Let's explore technology
together to live in the future

Checkout more on
https://github.com/Sy-hash-collab

Sy-hash-collab

Looping Structures

Loops:

"A type of control structure that repeates a statement or set of statement or set of statements is looping structure."

=> It is also known as iterative or repititive.

- Sequential Structure → all statments executed once
- Conditional Structure → may execute or skip statme on condition.

USAGE

To execute a statement or no. of statements for specified no. of times.
e.g To display name on screen 10 times.

To use a sequence of values.
e.g To display set of numbers from 1 to 10.

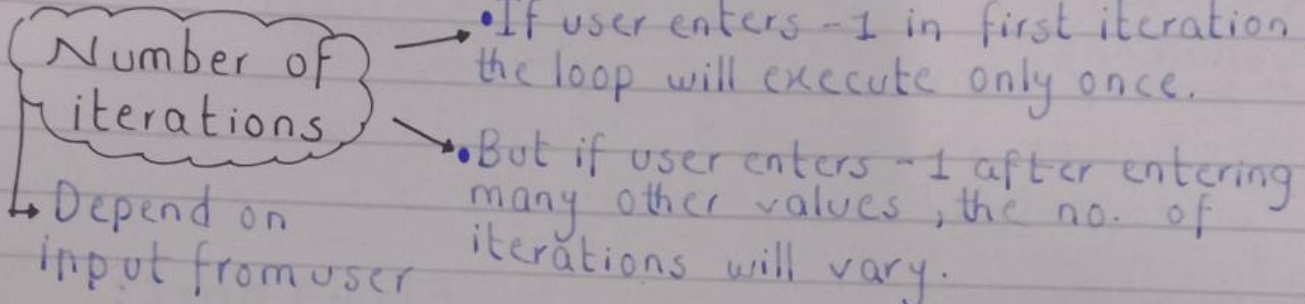
Counter-Controlled Loops:

- This type of loop depends on value of variable.
- The value of counter variable is incremented or decremented each time the body of loop is executed.
- This loop terminates when value of counter variable reaches a particular value.
- The iterations of this loop depend on the following:

=> Initial value of counter variable
=> Terminating Condition.
=> Increment / Decrement Value.

Sentinal Controlled Loop:

=> This type of loop depends on sentinal value. 'Sentinal' is a value in a list of values that indicates end of the list.
=> The loop terminate when sentinal value encountered These types of loops are known conditional loops
e.g A loop may execute while the value of variable is not -1. Here -1 is sentinal value that is used to terminate the loop.



Example:

```
int main()
{
  int n;
  cout << "Enter a number (0 to exist): ";
  cin>>n;
  while (n != 0)
{
   cout << "Square of " << n << "= " << n * n <<endl;
   cout << "Enter a number (0 to exit): ";
   cin >> n;
}
   cout << "Program ends";
}
```

'while' loop:

'Simplest' loop of C++ language.

=> It executes one or more statements while the given condition remains true.

=> It is useful when no. of iterations is not known in advance.

(* An iteration is an execution of the loop body.)

Syntax:

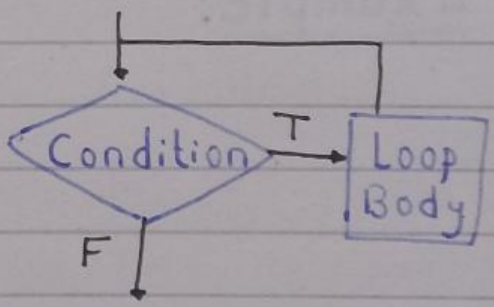
```
while (condition)
    statement;
```

Condition → It is given as relational exp. Statement is executed if condition is true and not when it is false.

Statement → Instruction when the condition is true.

```
while (condition)
{
    statement 1;
    statement 2;
    ⋮
    statement N;
}
```

Flow Chart:



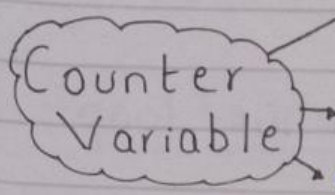
Example:

Display first five num. and their sum.

```
int main()
{ int c=1;
  int sum = 0;
  while (c<=5)
  { cout<<c<<endl;
    sum = sum+c;
    c = c+1; }
  cout<<"Sum = "<<sum;
```

Output:

1
2
3
4
5
Sum = 15



- A variable that is incremented / decrement each time a loop iterates.
- It can be use to control execution of loop
- Must be initialized.

infinite loop:

=> The loop must contain code to allow the condition to eventually become fasle so the loop can be exited.
=> Otherwise, you have infinite loop.

Example:

```
int main()
{
  int n = 1;
  while (n<=10)
  { cout << "Pakistan";
  }
}
```

(do-while) loop:

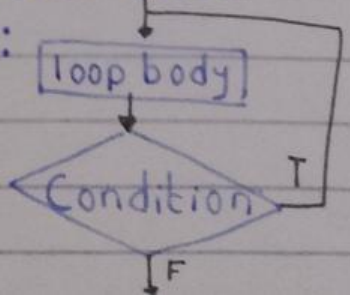
- Iterative control in C++.

=> This loop executes one or more statements while the given condition is.
=> In this loop, the condition comes after body of loop
=> It is important loop in situation when the loop body must be executed at least once.

Syntax:

```
do
{ statement
}
while(condition)
```

Flowchart:



Difference

'while' loop	'do-while' loop
• In while loop, condition comes before body of loop.	• In do-while loop, condition comes after body of loop.
• If condition is false in beginning, while loop is never executed	• 'do-while' loop is executed at least once even if condition is false in the beginning.

Example:

Program that gets starting and ending point from user and displays all odd numbers in given range, using do-while loop.

```
int main()
{
  int c, s, e;
  cout << "Enter starting number: ";
  cin >> s;
  count << " Enter ending number: ";
  cin >> e;
  c = s;
  do
  {
    if (c % 2 != 0)
      cout << c << endl;
    c = c + 1;
  }
  while (c <= e)
}
```

Output:

```
Enter starting number: 5
Enter ending number: 15
5
7
9
11
13
15
```

'for' loop:

Also called counter-controlled loop. 'for' loop executes one or more statements for a specified number of items.

=> Most flexible loop, frequently used loop by programmers.

Syntax:

```
for (initialization; condition; increment/
                                  decrement)
{
   statement 1;
   statement 2;
   :
   statement N;
}
```

Initialization: It specifies starting value of counter variable. One or many variables can be initialized in this part. To initialize many variables, each variable is separated by comma.

Condition: The condition is given as relational expression. The statement is executed only if the given condition is true. If false, never executed

Increment / Decrement: This part of loop specifies the change in counter variable after each execution of loop. To change many var. each variable must be separated by comma.

Statement: Statement is instruction that is executed when condition is true. If two or more statements are used, given in braces.

```
for (x=1, y=1 ; x<=10 ; x++ , y++)
```

Initialization — Condition — Increment / decrement.

Step 1: Perform initialization

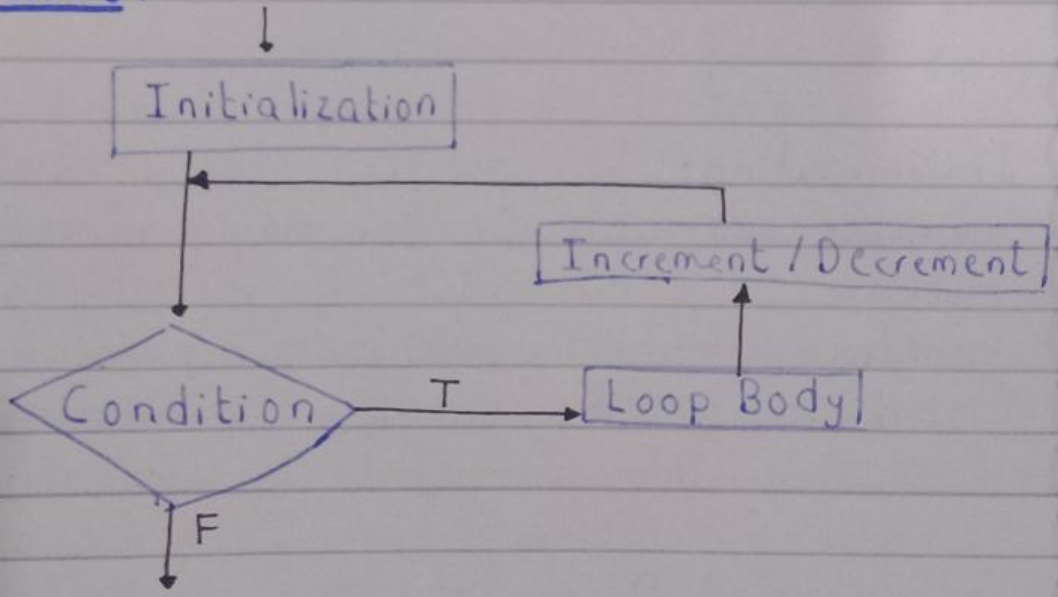
2) Evaluate exp. if true go to step 3, if false terminates the loop.

```
for (count=1; count<=5; count++)
{ cout << "Hello" << endl;
}
```

3) Execute the body of loop.

4) Perform the update expression. Then go back to step 2.

Flow Chart:



Displays counting from 1 to 5.

```
int main()
{
  int n;
  for (n=1; n<=5, n++)
    cout << n << endl;
}
```

Output:

```
1
2
3
4
5
```

For loop modifications:

=> You can define variables in initialization code
```
for (int n=1; n<=5; n++)
```
=> Can omit initialization if already done.
```
int n=1;
for (; n<=5; n++)
```
=> Can omit update if done in loop body.
```
for (n=1; n<=5; )
    n++
```
=> The condition in for loop is mandatory, can't be ommitted.
```
int n=1;
for (; n<=5; )
```

Example:

Program that inputs table number and length of table and then displays the table.

```
int main ()
{
  int table, length, c;
  cout << "Enter number for table: ";
  cin >> table;
  cout << "Enter length of table: ";
  cin >> length;
  for (c=1; c<= length ; c++)
  cout << tab << "*" << c << "=" << tab * c << endl;
}
```

Output:

```
Enter number for table: 2
Enter length of table: 5
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
```

Nested Loops:

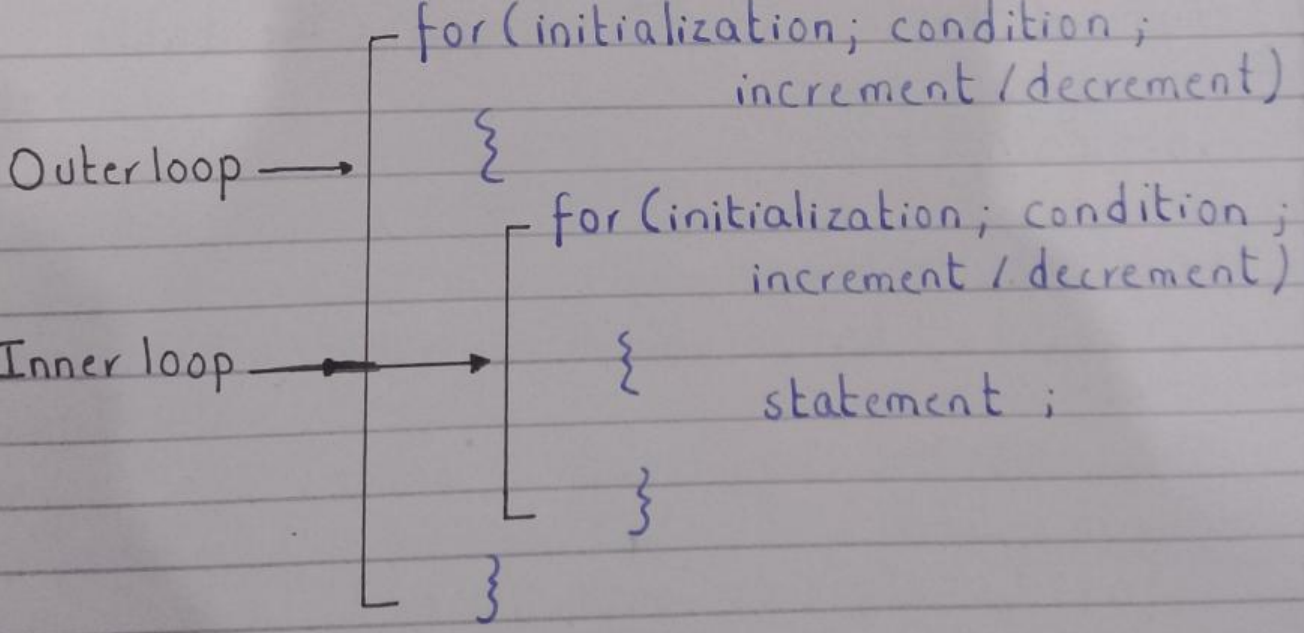
A loop within a loop is called nested loop. Nested loops consist of an outer loop with one or more inner loops.

Any loop can be used as inner of another loop. e.g 'while' loop can be used as outerloop. and 'for' loop can be used as inner in nested loop.

In nested loops, the inner loop is executed completely with each change in the value of counter variable of outer loop.

The nesting can be done up to any level. The increase in level of nesting increases the complexity of nested loop.

Syntax / General Form:



The inner loop goes through all its iterations for each iteration of outer loop.

Total number of iterations for inner loop is product of number of iterations of two loops.

Example:

```
int i, j;
{for(i=1 ; i<=2; i++)
    {
        for (j=1 ; j<=3 ; j++)
          cout << "Outer:" << i << "Inner:" << j ;
    }
}
```

Explanation: The above example uses nested for loop. The outer loop executes two times and the inner loop executes three times with each iteration of outer loop. It means that inner loop executes six times in total.

=> When the value of i is 1 in first iteration of outer loop, the value of j changes from 1 to 3 in three iterations of inner loop.

Similarly, when the value of i is 2 in second iteration of outer loop, the value of j again changes from 1 to 3 in three iterations of inner loop.

Output:

```
Outer: 1  Inner: 1
Outer: 1  Inner: 2
Outer: 1  Inner: 3
Outer: 2  Inner: 1
Outer: 2  Inner: 2
Outer: 2  Inner: 3
```

Write a program that displays the following block using nested for loop.

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

```
{
    int m, n ;

    for (m = 1 ; m <= 5 ; m++)
      {
        for (n = 1 ; n <= 5 ; n++)
          {cout << " * " ;
          }
        cout << endl;
      }
}
```

=> Write a program that displays all prime numbers between 10 and 100.
Prime numbers are divided by own or 1.

```
{
   int s;
    for ( s = 10 ; s <= 100 ; s++)
      {
        if (s % 2 == 0 || s % 3 == 0 || s % 5 == 0 || s % 7 == 0)
        {
          continue ;
        }
      cout << endl;
    }
```

Output:

```
11
13
17
19
23
29
:
:
97
```

'continue' Statement :

• The continue statement is used in the body of loop.

=> It is used to move the control to start of the loop body.

=> When this statement is executed in the loop body, the remaining statements of current iteration are not executed.

=> The control directly moves to the next iteration.

Example:

```
int x;
for (x=1; x<=5; x++)
{
  cout << "Hello World!\n";
  continue;
  cout << "Knowledge is power";
}
```

* The above example has two cout stat. One statement is before continue and one is after continue statement.
The second statement is never executed.

* This is because each time continue stat. is executed, the control moves back to the start of the body. Son "Knowledge is power" is never displayed.

'break' statement:

The break statement is used in the body of loop to exit from the loop.

=> When this statement is executed in the loop body, the remaining iterations of the loop are skipped.

=> The control directly moves outside the body and the statement that comes after the body is executed.

=> It can be used with while, do-while, for or switch structure.

=> When used in inner loop, break terminates that loop only and returns to outer loop.

Example:

```
for (int x=1 ; x<=5 ; x++)
 {
  cout << "Questioning \n";
  break;
  cout << "Gateway to knowledge";
 }
 cout << "Bye";
```

The counter variable x indicates that loop should execute for five times.

But it is executed only

In first iteration the cout statement in loop displays "Questioning" and control moves to break statment. This statement moves control out of the loop. Message apears once.