A C++ program is given below to display numbers 1, 2, 3, 4 and 5 on the screen.

```
#include<iostream.h>
void main()
{
   cout<<1<<endl;
   cout<<2<<endl;
   cout<<3<<endl;
   cout<<4<<endl;
   cout<<5<<endl;
}
```

A different program is given below which displays numbers 1, 2, 3, 4 and 5 on the screen.

```
#include<iostream.h>
void main()
{
   int k=1;
   cout<<k<<endl;
   k++;
   cout<<k<<endl;
   k++;
   cout<<k<<endl;
   k++;
   cout<<k<<endl;
   k++;
   cout<<k<<endl;
}
```

But if I want to display numbers 1, 2, 3, …, 99, 100 on the screen then the above programs will be neither suitable nor practical to produce desired output. So there has to be a better way or an elegant way of producing same output on the screen and this is possible by using loop in C++. In the second program, pair of statements `cout<<k<<endl;` and `k++;` are getting repeated 4 times. Using a loop in C++ it is possible to repeat a statement or a block. C++ supports three types of loops: **`while`** loop, **`for`** loop and **`do-while`** loop. Generally any loop in C++ has three components:

a) **Control Variable and its Initialization**: Generally a loop in C++ has a control variable. Control variable is generally an integer type or character type or floating point type. But we can have loop in C++ without a control variable. We will discuss such loops later. Usually a control variable of a loop is initialized.

b) **Terminating Condition**: Generally a loop should terminate after certain number of iterations. Terminating condition of the loop in C++ is a condition (logical expression) involving the control variable. Condition or logical expression plays a very important in the working of a loop in C++ since number of times loop is being repeated depends on the condition.

c) **Updating Control Variable**: Every C++ loop repeats statement or block. Inside the block or statement of a loop, control variable is updated (value of the control variable is either incremented or decremented), so that the loop terminates after certain number of iterations.

`while` **loop**

```
Rule:  while (Condition)
          Block / Statement;
```

The `Condition` is a logical expression involving control variable. First the `Condition` is evaluated; if the `Condition` is **TRUE** (nonzero), then the `Block` / `Statement` is executed. Inside the `Block` / `Statement`, control variable is updated and `Condition` is tested once again. These two steps are repeated till `Condition` is **FALSE**. When the condition is evaluated to false, the loop terminates.

Usage of **`while`** loop:

```
#include<iostream.h>
void main()
{
   int k=1;
   while (k<=5)
   {
      cout<<k<<endl;
      k++;
   }
}
```

Instead of `k++` one can either write `++k` or `k+=1` or `k=k+1`. Running of the program produces following output:

```
1
2
3
4
5
```

Explanation of output and working of the program (Initial value of `k` is `1`):

k<=5	cout<<k<<endl;	k++
TRUE	1	2
TRUE	2	3
TRUE	3	4
TRUE	4	5
TRUE	5	6
FALSE	6	

The **`while`** loop is an example of entry controlled loop, since first the `Condition` is tested and if the `Condition` is true then the `Block` or the `Statement` is executed. Let us take few more examples of **`while`** loop. Write a C++ program to generate and display following series on screen:

a) 1, 3, 5, 7, …, 19
b) 2, 4, 6, 8, …, 20
c) 30, 27, 24, …, 3

a)
```
#include<iostream.h>
void main()
{
   int k=1;                 //1 is initial value of the series
   while (k<=19)            //19 is final value of the series
   {
      cout<<k<<endl;
      k+=2;                 //common difference is +2
   }
}
```

b)
```
#include<iostream.h>
void main()
{
   int k=2;                //2 is initial value of the series
   while (k<=20)           //20 is final value of the series
   {
      cout<<k<<endl;
      k+=2;                //common difference is +2
   }
}
```

c)
```
#include<iostream.h>
void main()
{
   int k=30;               //30 is initial value of the series
   while (k>=3)            //3 is final value of the series
   {
      cout<<k<<endl;
      k-=3;                //common difference is -3
   }
}
```

When a loop used to generate and display a series, following points are to be noted:

- Initial value of the `Control` variable is the starting value of the series
- Terminating condition is `Control` variable lesser than equal to final value of the series if the series is ascending order. Terminating condition is `Control` variable greater than equal to final value of the series if the series is descending order.
- Updation of `Control` variable is += common difference or -= common difference depending on whether the series in ascending order or in descending order

Instead of using a fixed value as a final value, we can input a value like n (an integer value) to decide the final value of the series. For example write a C++ program to input n, then generate and display following series using **`while`** loop:

a) 1, 2, 3, 4, …, n
b) 2, 4, 6, 8, …, 2n
c) 2n-1, 2n-3, 2n-5, …, 5, 3, 1

a)
```
#include<iostream.h>
void main()
{
   int n;
   cout<<"Input n? ";
   cin>>n;
   int k=1;
   while (k<=n)
   {
      cout<<k<<endl;
      k++;
   }
}
```

Running of the program:
```
Input n? 5
1
2
3
4
5
```

Explanation of output:
Terminating condition is `k<=5`. Value of `k` less than 6 will satisfy the terminating condition. Hence the output `1` to `5`.
```
Input n? -4
```
No output since terminating condition `k<=n` is false when `n` is `-4`.

b)
```
#include<iostream.h>
void main()
{
   int n;
   cout<<"Input n? ";
   cin>>n;
   int k=2;
   while (k<=2*n)
   {
      cout<<k<<endl;
      k+=2;
   }
}
```

Running of the program:
```
Input n? 5
2
4
6
8
10
```

Explanation of output:
Terminating condition is `k<=10`. Value of `k` less than `12` will satisfy the terminating condition. Hence the output all even numbers between `2` and `10`.

b)
```
#include<iostream.h>
void main()
{
   int n;
   cout<<"Input n? "; cin>>n;
   int k=1;
   while (k<=n)
   {
      int t=2*k;
      cout<<t<<endl;
      k++;
   }
}
```

Running of the program:
```
Input n? 5
2
4
6
8
10
```

Explanation of output:
Terminating condition is `k<=5`. Value of `k` less than `6` will satisfy the terminating condition. Hence the output all even numbers between `2` and `10`.

c)
```
#include<iostream.h>
void main()
{
   int n;
   cout<<"Input n? ";
   cin>>n;
   int k=2*n-1;
   while (k>=1)
   {
      cout<<k<<endl;
      k-=2;
   }
}
```

Running of the program:
```
Input n? 5
9
7
5
3
1
```

Explanation of output:
Terminating condition is `k>=1`. Value of `k` greater than `0` will satisfy the terminating condition. Hence the output all odd numbers between `9` and `1`.

c)
```
#include<iostream.h>
void main()
{
   int n;
   cout<<"Input n? "; cin>>n;
   int k=n;
   while (k>=1)
   {
      int t=2*k-1;
      cout<<t<<endl;
      k--;
   }
}
```

Running of the program:
```
Input n? 5
9
7
5
3
1
```

Explanation of output:
Terminating condition is `k>=1`. Value of `k` greater than `0` will satisfy the terminating condition. Hence the output all odd numbers between `9` and `1`.

for loop

As mentioned earlier that there are three types of loop. Now we will learn about **for** loop. **for** loop is similar to **while** loop but it is more compact than **while** loop. As discussed earlier that any loop has three important components: an initialization of control variable, a terminating condition and updating of control variable. In **for** loop all three components are inside a parenthesis immediately after the keyword **for**. A **for** loop is also an example of entry controlled loop.

```
Rule: for (Initialization; Condition; Update)
          Block / Statement;
```

When the loop starts for the first time, Control variable is initialized. Next the Condition is tested. If the Condition is **TRUE** (nonzero), then the Block / Statement is executed. Next Control variable is updated and Condition is tested once again. When the condition is evaluated to false, the loop terminates. Although Initialization, Condition and Updation are together but their order of execution is different. Secondly Initialization of Control variable is executed only once when the loop starts for first time.

Usage of **while** loop:

```
#include<iostream.h>
void main()
{
   for (int k=1; k<=7; k++)
      cout<<k<<endl;
}
```

Instead of k++ one can either write ++k or k+=1 or k=k+1. Running of the program produces following output:

```
1
2
3
4
5
6
7
```

Explanation of output and working of the program (Initial value of k is 1):

k<=5	cout<<k<<endl;	k++
TRUE	1	2
TRUE	2	3
TRUE	3	4
TRUE	4	5
TRUE	5	6
TRUE	6	7
TRUE	7	8
FALSE	8	

Let us take few more examples of **for** loop. Write a C++ program to generate and display following series on screen:

a) 2, 4, 6, 8, …, 20
b) 5, 10, 15, 20, …, 100
c) 40, 36, 32, …, 12, 8, 4

a)
```
#include<iostream.h>
void main()
{
   for (int k=2; k<=20; k+=2)
      cout<<k<<endl;
}
```

b)
```
#include<iostream.h>
void main()
{
   for (int k=5; k<=100; k+=5)
      cout<<k<<endl;
}
```

c)
```
#include<iostream.h>
void main()
{
   for (int k=40; k>=4; k-=4)
      cout<<k<<endl;
}
```

Instead of using a fixed value as a final value, we can input a value like n (an integer value) to decide the final value of the series. For example write a C++ program to input n, then generate and display following series using **`for`** loop:

a) 1, 2, 3, 4, …, n
b) 1, 3, 5, 7, …, 2n-1
c) 2n, 2n-2, 2n-4, …, 6, 4, 2

a)
```
#include<iostream.h>
void main()
{
   int n;
   cout<<"Input n? ";
   cin>>n;
   for (int k=1; k<=n; k++)
      cout<<k<<endl;
}
```
	Running of the program:
	```
	Input n? 5
	1
	2
	3
	4
	5
	```
	Explanation of output:
	Terminating condition is `k<=5`. Value of `k` less than 6 will satisfy the terminating condition. Hence the output `1` to `5`.

b)
```
#include<iostream.h>
void main()
{
   int n;
   cout<<"Input n? ";
   cin>>n;
   for (int k=1; k<=2*n-1;k+=2)
      cout<<k<<endl;
}
```
	b)
	```
	#include<iostream.h>
	void main()
	{
	   int n;
	   cout<<"Input n? "; cin>>n;
	   for (int k=1; k<=n; k++)
	   {
	      int t=2*k-1;
	      cout<<t<<endl;
	   }
	}
	```

Running of the program:
`Input n? 5`
`1`
`3`
`5`
`7`
`9`
 	
Explanation of output:
Terminating condition is `k<=9`. Value of `k` less than `11` will satisfy the terminating condition. Hence the output all odd numbers between `1` and `9`.
	Running of the program:
	`Input n? 5`
	`1`
	`3`
	`5`
	`7`
	`9`
	
	Explanation of output:
	Terminating condition is `k<=5`. Value of `k` less than `6` will satisfy the terminating condition. Hence the output all odd numbers between `1` and `9`.

c) `#include<iostream.h>`
`void main()`
`{`
`   int n;`
`   cout<<"Input n? ";`
`   cin>>n;`
`   for (int k=2*n; k>=2; k-=2)`
`      cout<<k<<endl;`
`}`
	c) `#include<iostream.h>`
	`void main()`
	`{`
	`   int n;`
	`   cout<<"Input n? "; cin>>n;`
	`   for (int k=n; k>=1; k--)`
	`   {`
	`      int t=2*k;`
	`      cout<<k<<endl;`
	`   }`
	`}`

Running of the program:
`Input n? 5`
`10`
`8`
`6`
`4`
`2`
 	
Explanation of output:
Terminating condition is `k>=2`. Value of `k` greater than `0` will satisfy the terminating condition. Hence the output all odd numbers between `10` and `2`.
	Running of the program:
	`Input n? 5`
	`10`
	`8`
	`6`
	`4`
	`2`
	
	Explanation of output:
	Terminating condition is `k>=1`. Value of `k` greater than `0` will satisfy the terminating condition. Hence the output all odd numbers between `10` and `2`.

`do-while` loop

The **`do-while`** loop is almost similar to **`while`** loop but only difference is in **`while`** loop terminating condition is at the top where as in **`do-while`** loop terminating condition is at the bottom. That loop is executed first and then the condition is tested. The **`do-while`** loop is called exit control loop. Exit control loop is executed at least once.

Rule:
```
do
{
   //C++ Statements
}
while (Condition);
```

The `Condition` is a logical expression involving control variable. First the `Condition` is evaluated; if the `Condition` is **TRUE** (nonzero), then the `Block` / `Statement` is executed. Inside the `Block` /

`Statement`, control variable is updated and `Condition` is tested once again. These two steps are repeated till `Condition` is **FALSE**. When the condition is evaluated to false, the loop terminates.

Usage of **`while`** loop:

```
#include<iostream.h>
void main()
{
   int k=1;
   while (k<=5)
   {
      cout<<k<<endl;
      k++;
   }
}
```

Instead of `k++` one can either write `++k` or `k+=1` or `k=k+1`. Running of the program produces following output:

```
1
2
3
4
5
```

Explanation of output and working of the program (Initial value of `k` is `1`):

k<=5	cout<<k<<endl;	k++
TRUE	1	2
TRUE	2	3
TRUE	3	4
TRUE	4	5
TRUE	5	6
FALSE	6	

Usage of **`do-while`**

```
#include<iostream.h>
void main()
{
   int n, k=1, fact=1;
   cout<<"Input positive integer? "; cin>>n;
   do
   {
      fact*=k;
      k++;
   }
   while (k<=n);
   cout<<"Factorial="<<fact<<endl;
}
```

Running of the program

```
Input positive integer? 10
Factorial=3628800
```

Explanation of output and working of the output

Inputted value in `n` is `10` and initial value of `k=1`

k	fact*=k	k++	k<=10
1	1	2	**TRUE**
2	2	3	**TRUE**
3	6	4	**TRUE**
4	24	5	**TRUE**
5	120	6	**TRUE**
6	720	7	**TRUE**
7	5040	8	**TRUE**
8	40320	9	**TRUE**
9	362880	10	**TRUE**
10	3628800	11	**FALSE**

```
#include<iostream.h>
void main()
{
   int n, sum=0;
   cout<<"Input positive integer? "; cin>>n;
   for (int k=1; k<=n; k++)
      sum+=k*k;
   cout<<"Sum="<<sum<<endl;
}
```

Running of the program

```
Input positive integer? 7
Sum=55
```

Explanation of output and working of the output

Inputted value in `n` is `5` and initial value of `k=1`

k<=10	k	k*k	sum+=k*k	k++
TRUE	1	1	1	2
TRUE	2	4	5	3
TRUE	3	9	14	4
TRUE	4	16	30	5
TRUE	5	25	55	6
FALSE				

```
//Program to find HCF and LCM of two integers
#include<iostream.h>
void main()
{
   int a, b;
   cout<<"Input two integers? "; cin>>a>>b;
   int prod=a*b, r;
   do
   {
      r=a%b;
      a=b;
      b=r;
   }
   while(r>0);
   cout<<"HCF="<<a<<endl;
   cout<<"LCM="<<(prod/a)<<endl;
}
```

```
//Count digits, sum of digits, product of digits
#include<iostream.h>
void main()
{
   int n, digit=0, sum=0, prod=1;
   cout<<"Input an integer? ";cin>>n;
   while (n!=0)
   {
      int r=n%10;
      digit++;
      sum+=r;
      prod*=r;
      n/=10;
   }
   cout<<"Number of digits="<<digit<<endl;
   cout<<"Sum of digits="<<sum<<endl;
   cout<<"Product of digits="<<prod<<endl;
}
```

```
//Program to reverse an inputted integer
#include<iostream.h>
void main()
{
   int n, m=0;
   cout<<"Input an integer? "; cin>>n;
   while (n!=0)
   {
      int d=n%10;
      m=10*m+d;
      n/=10;
   }
   cout<<"Reversed integer="<<m<<endl;
}
```

```
//Check for Prime Number
#include<iostream.h>
void main()
{
   int n;
   cout<<"Input an integer? ";cin>>n;
   int k=2, prime=1;
   while (k<n && prime==1)
   {
      if (n%k==0)
         prime=0;
      k++;
   }
   if (prime==1)
      cout<<n<<" Prime Number"<<endl;
   else
      cout<<n<<" Composite Number"<<endl;
}
```

```
//Check for Armstrong Number
#include<iostream.h>
void main()
{
   int n;
   cout<<"Input an integer? ";cin>>n;
   int t=n, s=0;
   while (n!=0)
   {
      int d=n%10;
      s+=d*d*d;
      n/=10;
   }
   if (s==t)
      cout<<t<<" Armstrong Number"<<endl;
   else
      cout<<t<<" Not Armstrong Number"<<endl;
}

//Check for Palindromic Integer
#include<iostream.h>
void main()
{
   int n;
   cout<<"Input an integer? ";cin>>n;
   int t=n, m=0;
   while (n!=0)
   {
      m=10*m+n%10;
      n/=10;
   }
   if (t==m)
      cout<<t<<" Palindromic integer"<<endl;
   else
      cout<<t<<" Not Palindromic integer"<<endl;
}
```

- **Nested loop**
  A loop may contain another loop in its block or statement. This form of a loop is called nested loop. But in a nested loop, the inner loop must terminate before the outer loop. An example is given below:

```
#include<iostream.h>
void main()
{
   for (int k=1; k<=4; k++)          //Outer loop
   {
      for (int j=1; j<=k; j++)       //Inner loop
         cout<<'*';
      cout<<endl;
   }
}
```

Running of the program

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

Explanation of the output

The block of outer loop (**for** k-loop) contains inner loop (**for** j-loop). Outer loop is executed 4 times. Therefore the block with outer loop is also executed 4 times. The block of outer loop contains inner loop. The table given below explains execution of inner loop.

k	Iteration of Inner loop	Output
1	1	*
2	2	**
3	3	***
4	4	****

```
#include<iostream.h>    //Same program using while loop
void main()
{
   int k=1;
   while (k<=4)         //Start of Outer loop
   {
      int j=1;
      while (j<=k)      //Start of Inner loop
      {
         cout<<'*';
         j++;
      }                 //End of Inner loop
      cout<<endl;
      k++;
   }                    //End of Outer loop
}
```

```
#include<iostream.h>    //Same program using do-while loop
void main()
{
   int k=1;
   do                             //Start of Outer loop
   {
      int j=1;
      do                //Start of Inner loop
      {
         cout<<'*';
         j++;
      }
      while (j<=k);     //End of Inner loop
      cout<<endl;
      k++;
   }
   while (k<=4);                  //End of Outer loop
}
```

- **Infinite loop**

  A loop that never terminates is called an infinite loop. Kind of program we will do using loops, will involve only finite loop (loop that terminates) but there are no harm in learning about infinite loop. Infinite loop can be implemented with **`while`** or **`for`** or **`do-while`** loop.

  An important point to remember is that in C++ any non-zero value is considered to be **TRUE** and zero (0) value is **FALSE**.

  Rule:
  ```
  while (NonZeroValue)         //Infinite while loop
     Block / Statement

  for (;;)
     Block / Statement         //Infinite for loop

  do
  {                            //Infinite do-while loop
     //C++ Statements
  }
  while (NonZeroValue)
  ```

  Examples of **<u>infinite</u>** loops are given below:

  a) Using **`while`** loop:
  ```
  #include<iostream.h>
  void main()
  {
     while (1)
        cout<<"*";
  }
  ```
  The program displays stars (*) on the screen infinitely. To terminate the program and the loop click the Close Icon of the DOS Window.

  b) Using **`do-while`** loop:
  ```
  #include<iostream.h>
  void main()
  {
     do
     {
        cout<<"*";
     }
     while (1);
  }
  ```

  c) Using **`for`** loop:
  ```
  #include<iostream.h>
  void main()
  {
     for ( ; ; )
        cout<<"*";
  }
  ```