Object Oriented Programming

By:- Diksha Ruhela

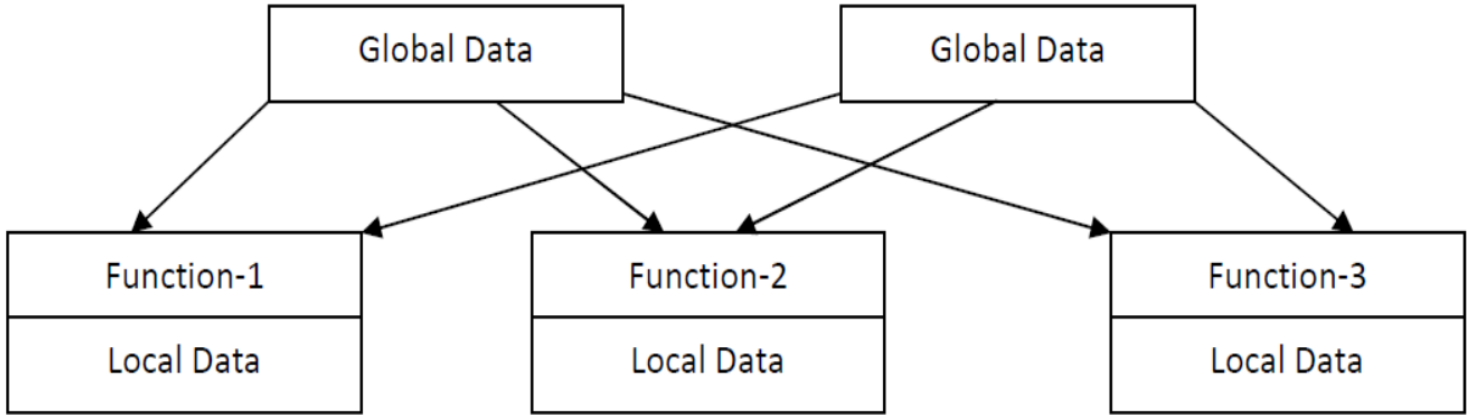
1. INTRODUCTION

- **Procedure/ structure oriented Programming**

1.Conventional programming, using high level languages such as COBOL, FORTRAN and C, is commonly known as procedure-oriented programming (POP).

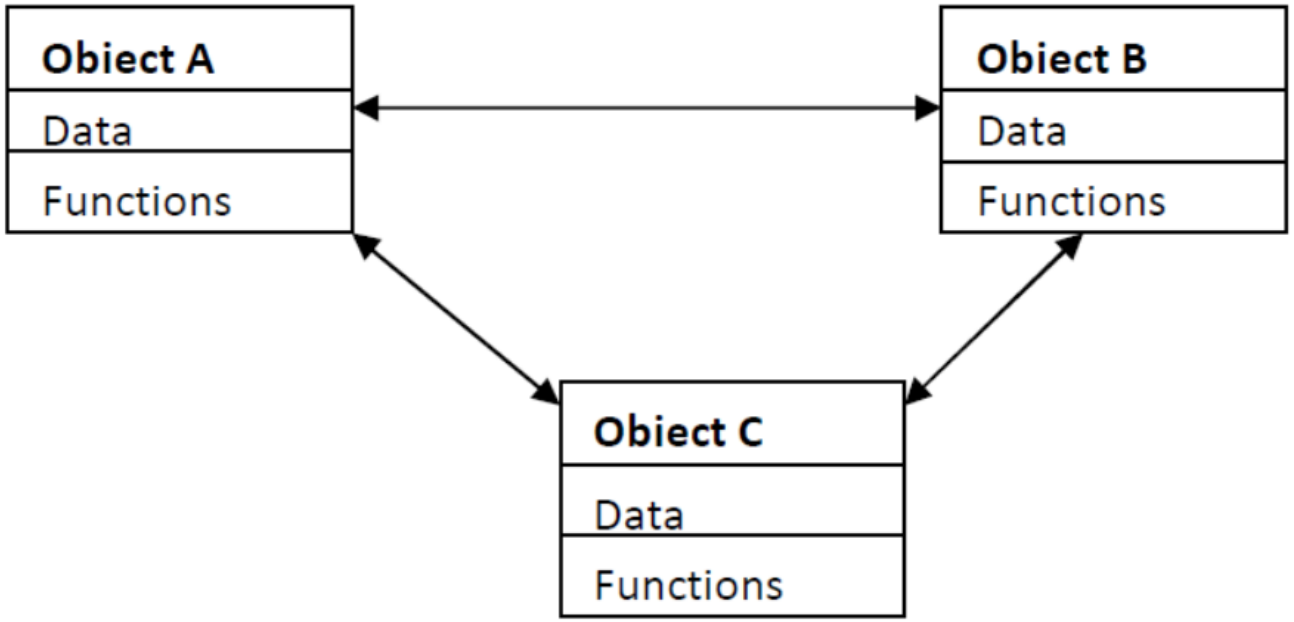
2.In the procedure-oriented approach, the problem is viewed as a sequence of things to be done such as reading, calculating and printing. A number of functions are written to accomplish these tasks.

3. The primary focus is on functions



- **Object Oriented Programming**

1. Emphasis is on data rather than procedure.
2. Programs are divided into what are known as objects.
3. Data is hidden and cannot be accessed by external functions.
4. Objects may communicate with each other through functions.
5. New data and functions can be easily added whenever necessary.



Procedural Oriented Programming	Object-Oriented Programming
In procedural programming, the program is divided into small parts called ***functions***.	In object-oriented programming, the program is divided into small parts called ***objects***.
Procedural programming follows a ***top-down approach***.	Object-oriented programming follows a ***bottom-up approach***.
There is no access specifier in procedural programming.	Object-oriented programming has access specifiers like private, public, protected, etc.
Adding new data and functions is not easy.	Adding new data and function is easy.
Procedural programming does not have any proper way of hiding data so it is ***less secure***.	Object-oriented programming provides data hiding so it is ***more secure***.
In procedural programming, overloading is not possible.	Overloading is possible in object-oriented programming.
In procedural programming, there is no concept of data hiding and inheritance.	In object-oriented programming, the concept of data hiding and inheritance is used.
In procedural programming, the function is more important than the data.	In object-oriented programming, data is more important than function.
Procedural programming is based on the ***unreal world***.	Object-oriented programming is based on the ***real world***.
Procedural programming is used for designing medium-sized programs.	Object-oriented programming is used for designing large and complex programs.
Procedural programming uses the concept of procedure abstraction.	Object-oriented programming uses the concept of data abstraction.
Code reusability absent in procedural programming,	Code reusability present in object-oriented programming.
Examples: C, FORTRAN, Pascal, Basic, etc.	**Examples:** C++, Java, Python, C#, etc.

Basic Concepts of Object-Oriented Programming

- **Objects**

Objects are the basic runtime entities in an object oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.

- **Class**

Object contains data, and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the

help of a class.

```
//Class name is Car
class Car
{
    //Data members
    char name[20];
    int speed;
    int weight;

public:
    //Functions
    void brake(){
    }
    void slowDown(){
    }
};

int main()
{
   //ford is an object
   Car ford;
}
```

- **Data Encapsulation**

  The wrapping up of data and functions into a single unit is known as encapsulation.

  The data is not accessible to the outside world, only those function which are wrapped in the can access it.

  These functions provide the interface between the object's data and the program.

  This insulation of the data from direct access by the program is called **data hiding** or **information hiding**.

- **Data Abstraction**

  Abstraction refers to the act of representing essential features without including the background details or explanations.

  Since classes use the concept of data abstraction, they are known as **Abstract Data Types (ADT)**.

- **Inheritance**

  Inheritance is the process by which objects of one class acquire the properties of objects of another class.

  In OOP, the concept of inheritance provides the idea of reusability. This means we can add additional features to an existing class without modifying it

```
#include <iostream>
using namespace std;
class ParentClass {
  //data member
  public:
    int var1 =100;
};
class ChildClass: public ParentClass {
  public:
  int var2 = 500;
};
int main(void) {
  ChildClass obj;
}
```

Now this object obj can use the properties (such as variable var1) of ParentClass.

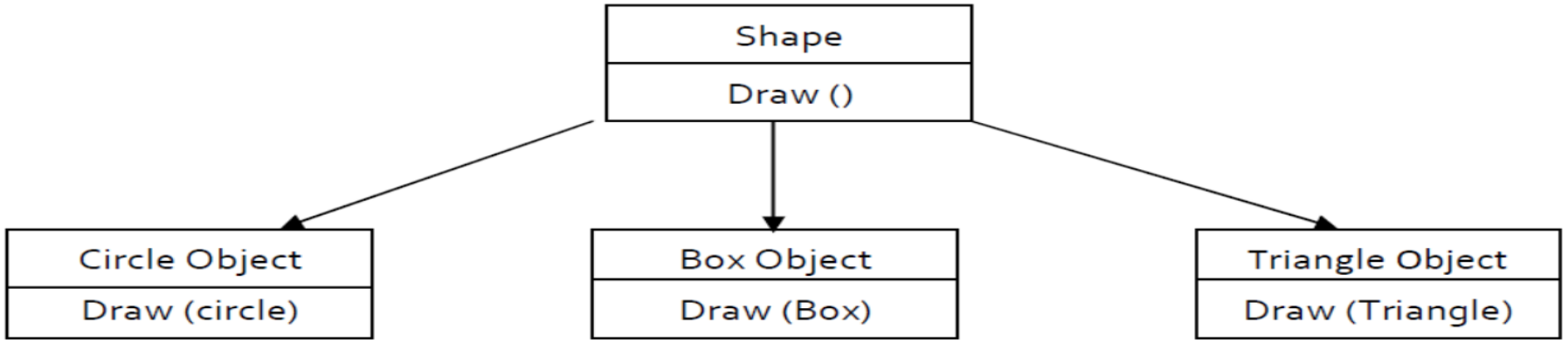
- **Polymorphism**

Polymorphism, a Greek term means to ability to take more than one form.

An operation may exhibits different behaviors in different instances. The behavior depends upon the type of data used in the operation.

For example consider the operation of addition for two numbers; the operation will generate a sum. If the operands are string then the operation would produce a third string by concatenation.

The process of making an operator to exhibit different behavior in different instances is known operator overloading.



Introduction to C++

- C++ is a superset of C.
- Important properties that C++ add on to C are:- Classes, inheritance, function overloading and operator overloading.
- Simple C++ program written as:

```
#include<iostream>     // #include directive cause the pre-processor to add the content of the iostream file to the
                       //program. It contain the declaration of the identifier cout and operator <<.
                       //refer table no. 2.1 (reference book) for more header files
Using Namespace std    //new concept introduce. It defines the scope of the identifier used in the program.  Here std is the
                       namespace where c++ standard class library are defined. //It bring all the identifiers define in std to the
                       current global scope.
Int main()                                  // it return the integer type value to the operating system
{ cout<<" introduction to c++"; // statement or comment (double quotation mark)
Return 0;    // by default return type of all function is int
}
```

- Comment:- //(double slash)

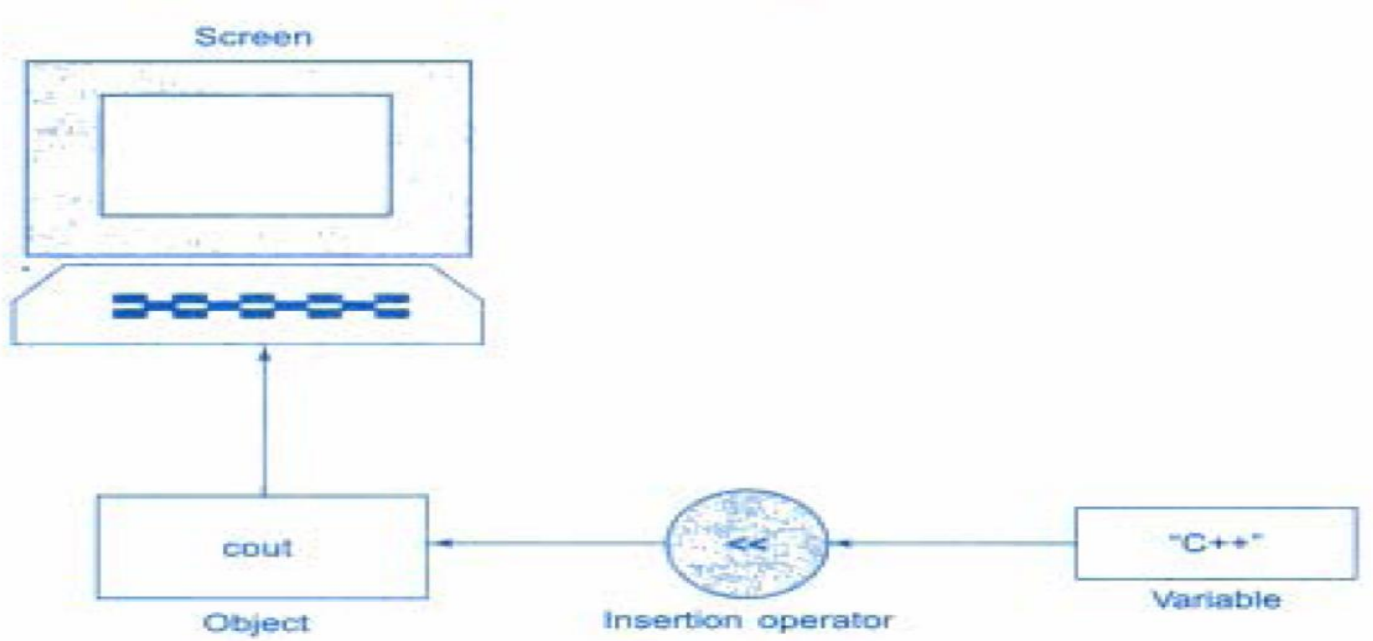
  //(to write the comment)

  multiline comment use /* example to illustrate the use

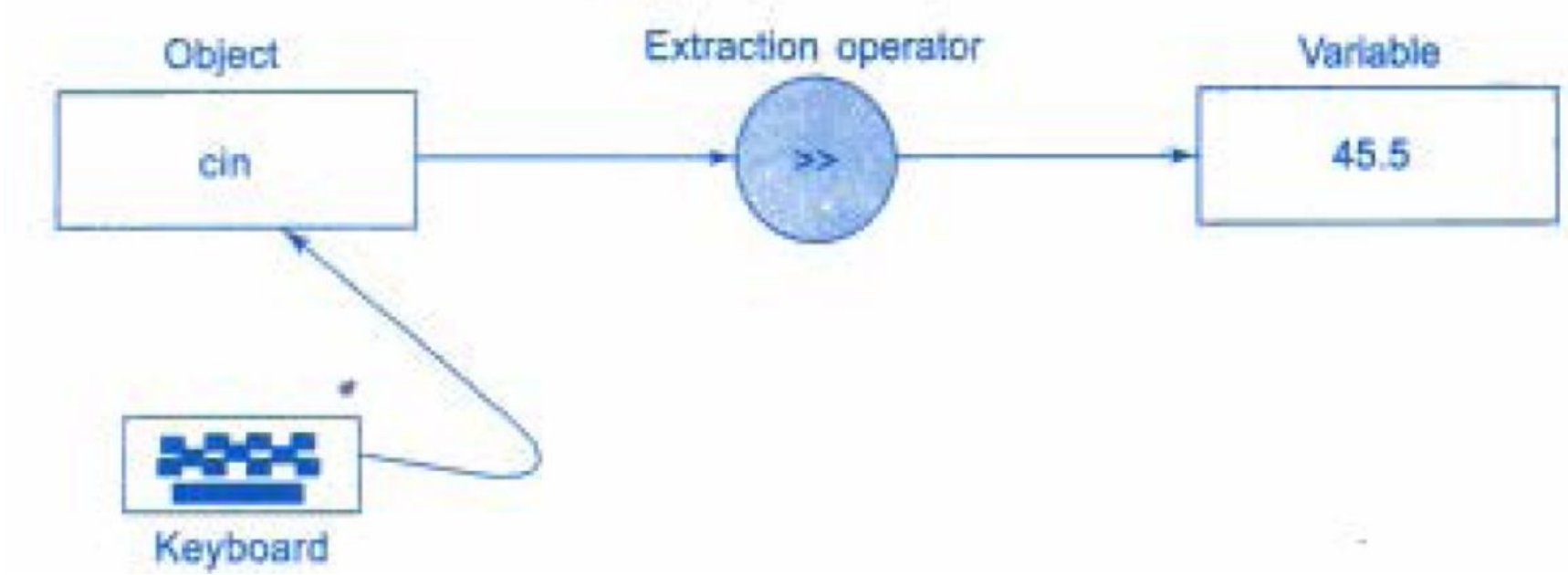
  c++ features */

Output and Input Statement in C++

- **An Output statement** is used to print the output on computer screen. It is a standard predefine object that represent the standard output stream in C++. **cout** is an output statement.
- cout<<**"**Delhi College of Engineering**";** //prints **delhi college of Engineering** on computer screen.
- cout<<**"**x**";** //print **x** on computer screen // using double quotation mark
- cout<<x**;** //prints **value of x** on computer screen.
- cout<<"**\n**"**;** takes the cursor to a newline.
- << (two "less than" signs) is called insertion operator or put to operator. It insert or send the content of the variable on its right to the object on its left



- **An Input statement** is used to take input from the keyboard. **cin** is an input statement.
- cin>>x; takes the value of x from keyboard.
- cin>>x>>y; takes value of x and y from the keyboard.
- Cin is a predefined object in c++ correspond to standard input stream. (keyboard)
- >> (two "greater then sign") is known as extraction or get from operator. It takes the value form the keyboard and assign it to the variable on its right.



- Tokens: (smallest individual units)

1. Keywords (reserved words it is used to make the language more versatile . It cannot be used as user defined program element or program variable eg: int, switch, void, float, else etc.)
2. Identifiers (name of the function, array, class variable etc created by programmer)
3. Constants (fixed value that cannot change during execution of the program)
4. Operators and strings.

Operators:-

1. Arithmetic Operators (+, -, *, /, %)

The five arithmetical operations supported by the C++ language are:

- ✓ Addition (+)
- ✓ Subtraction (-)
- ✓ Multiplication (*)
- ✓ Division (/)
- ✓ Modulo (%)

Operations of addition, subtraction, multiplication and division literally correspond with their respective mathematical operators.

Division Rule

Integer/integer=integer

Integer/float=float

Float/integer=float

Float /float=float

2. Assignment Operator (=)

- ✓ The assignment operator assigns a value to a variable.
- ✓ a = 5;

Shorthand assignment (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

- ➢ a -= 5; is equivalent to a = a - 5;
- ➢ a /= b; is equivalent to a = a / b;
- ➢ price *= units + 1; is equivalent to price = price * (units + 1); and the same for all other operators

3. Relational and equality operators (==, !=, >, <, >=, <=)

- ✓ Here there are some examples:
- ✓ (7 == 5) // evaluates to false.
- ✓ (5 > 4) // evaluates to true.
- ✓ (3 != 2) // evaluates to true.
- ✓ (6 >= 6) // evaluates to true.
- ✓ (5 < 5) // evaluates to false.
- ✓ Of course, instead of using only numeric constants, we can use any valid expression, including variables.
- ✓ Suppose that a=2, b=3 and c=6,
- ✓ (a == 5) // evaluates to false since a is not equal to 5.
- ✓ (a*b >= c) // evaluates to true since (2*3 >= 6) is true.
- ✓ (b+4 > a*c) // evaluates to false since (3+4 > 2*6) is false.
- ✓ ((b=2) == a) // evaluates to true

4. Logical operators (!, &&, ||)

- For example:
- ((5 == 5) && (3 > 6)) // evaluates to false (true && false).
- ((5 == 5) || (3 > 6)) // evaluates to true (true || false).
- ! → NOT, && → AND, || → OR

a	b	a && b
True	True	True
True	False	False
False	True	False
False	False	False

a	b	a || b
True	True	True
True	False	True
False	True	True
False	False	False

5. Increment and Decrement Operator (++, --)

- The increment operator (++) and the decrement operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to - =1, respectively. Thus:

```
c++;
c+=1;
c=c+1;
```

all are same.

- Pre → First increment/decrement then assignment.
- Post → First assignment then increment/decrement.
- Ex: 1. **Find the value of A and B.**

```
B=3;
A=++B;
```

Ans: A contains 4, B contains 4

Ex: 2 Find the value of A and B.

```
B=3;
A=B++;
```

Ans: A contains 3, B contains 4

In Example 1, B is increased before its value is assigned to A. While in Example 2, the value of B is assigned to A and then B is increased.

6. Conditional operator (? :)

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false. Its format is: condition ? result1 : (colon) result2. If condition is true the expression will return result1, if it is not it will return result2.

7==5 ? 4 : 3 // returns 3, since 7 is not equal to 5.

7==5+2 ? 4 : 3 // returns 4, since 7 is equal to 5+2.

5>3 ? a : b // returns the value of a, since 5 is greater than 3.

a>b ? a : b // returns whichever is greater, a or b.

CONTROL STATEMENTS

1. Selection statements

a. if Statement

The **if** statement is C++'s conditional branch statement. It can be used to route program execution through two different paths. Here is the general form of the **if** statement:

```
if (condition)
{
statement1;
}
else
{
statement2;
}
```

```
void main()
{
int age;
cout<<"Enter Age";
   if(age>=35)
cout<<"Old";

else
cout<<"Young";
}
```

Output:

```
Enter age 39
Old
```

b. Switch Statements

The **switch** statement is C++'s multi way branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression. As such, it often provides a better alternative than a large series of **if-else-if** statements. Here is the general form of a **switch** statement:

```
switch (expression)
{
case value1:
// statement sequence
break;
case value2:
// statement sequence
break;
...
case valueN:
// statement sequence
break;
default:
// default statement sequence
```

Ex: switch

```
#include<iostream>
void main()
{
int i
for(int i=0; i<6; i++)
switch(i)
{
case 0:
cout<<"i is zero.";
break;
case 1:
cout<<"i is one.";
break;
case 2:
cout<<"i is two.";
break;
case 3:
cout<<"i is three.";
break;
default:
cout<<"i is greater than 3.";
}
}
```

Output:

```
i is zero.
i is one.
i is two.
i is three.
i is greater than 3.
i is greater than 3.
```

ITERATION STATEMENTS

- C++'s iteration statements are for, while, and do-while. These statements create what we commonly call loops

1. while loop

The while loop is Java's most fundamental looping statement. It repeats a statement or block while its controlling expression is true. Here is its general form:

```
initialization
while (condition)
{
// body of loop
Increment/ decrement
}
```

```
#include<iostream.h>
void main()
{
        int n = 0;
        while (n <10)
        {
        cout<<"C++ is good";
        n++;
        }
}
```

2. do while Loop

If the conditional expression controlling a **while** loop is initially false, then the body of the loop will not be executed at all. However, sometimes it is desirable to execute the body of a **while** loop at least once, even if the conditional expression is false to begin with. The **do-while** loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Its general form is:

initialization

do

{

// body of loop

Increment/ decrement

} while (condition);

```
#include<iostream.h>
void main()
{
        int n = 0;
        do
        {
        cout<<"C++ is good";
        n++;
        } while(n<9);
}
```

3. for loop

The general form of the **for** statement is:

```
for(initialization; condition; iteration)
{
// body
}
```

Ex:

```
for(n=0; n<10; n++)
{
cout<<"C++ is good";
}
```

Jump Statements

C++ offers following jump statements:

break statement: A break statement takes control out of the loop.

continue statement: A continue statement takes control to the beginning of the loop.

goto statement: A goto Statement take control to a desired line of a program.