UNIT-2 Control Flow:

Control Flow: If statement, If Else statement, Nested If, Else, Statements, For Loop, While Loop, Do, While Loop, Break, Switch, Continue, goto. Classes and Objects, Encapsulation, information hiding, abstract data types, Object & classes, attributes, methods, C++ class declaration, Constructors and destructors, Default parameter value, object types, C++ garbage collection, dynamic memory allocation, Metaclass / abstract classes.

Outcome of this unit- Proficiency in writing structured, modular, and object-oriented C++ code.
Improved problem-solving skills through better understanding and application of control flow mechanisms and OOP principles.
Ability to design and implement complex systems using classes, objects, and inheritance.
Understanding of memory management concepts and best practices in C++ programming.
Enhanced software development skills, making you a more competent and versatile programmer.

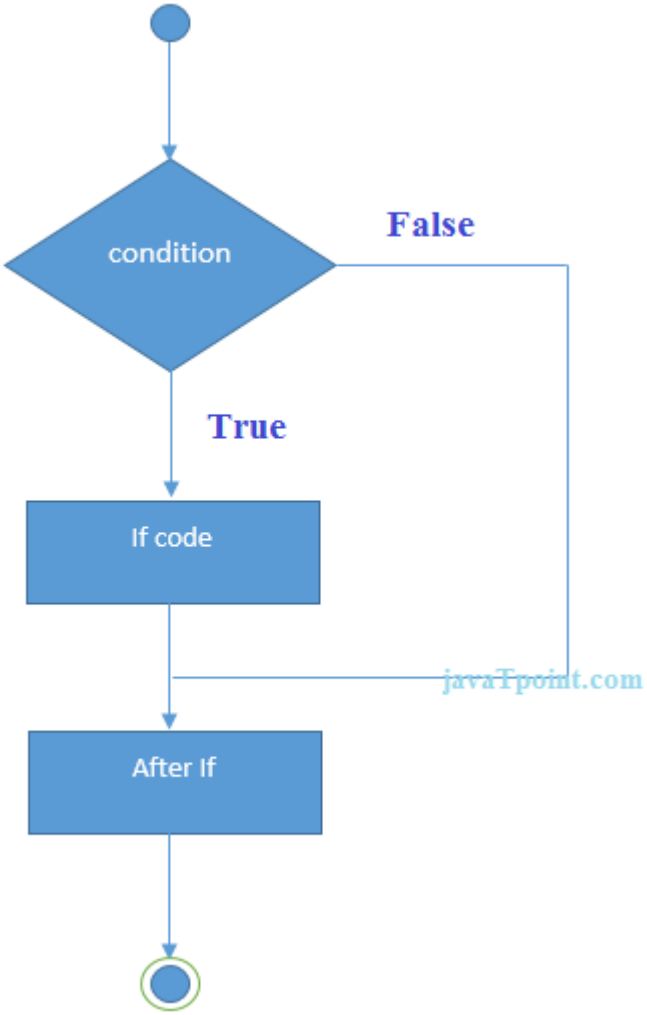
C++ IF Statement

The C++ if statement tests the condition. It is executed if condition is true.

```
1. if(condition){
2. //code to be executed
3. }
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:



C++ If Example

```
1. #include <iostream>
2. using namespace std;
3.
4. int main () {
5.    int num = 10;
6.          if (num % 2 == 0)
7.          {
8.              cout<<"It is even number";
9.          }
10.   return 0;
11.}
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

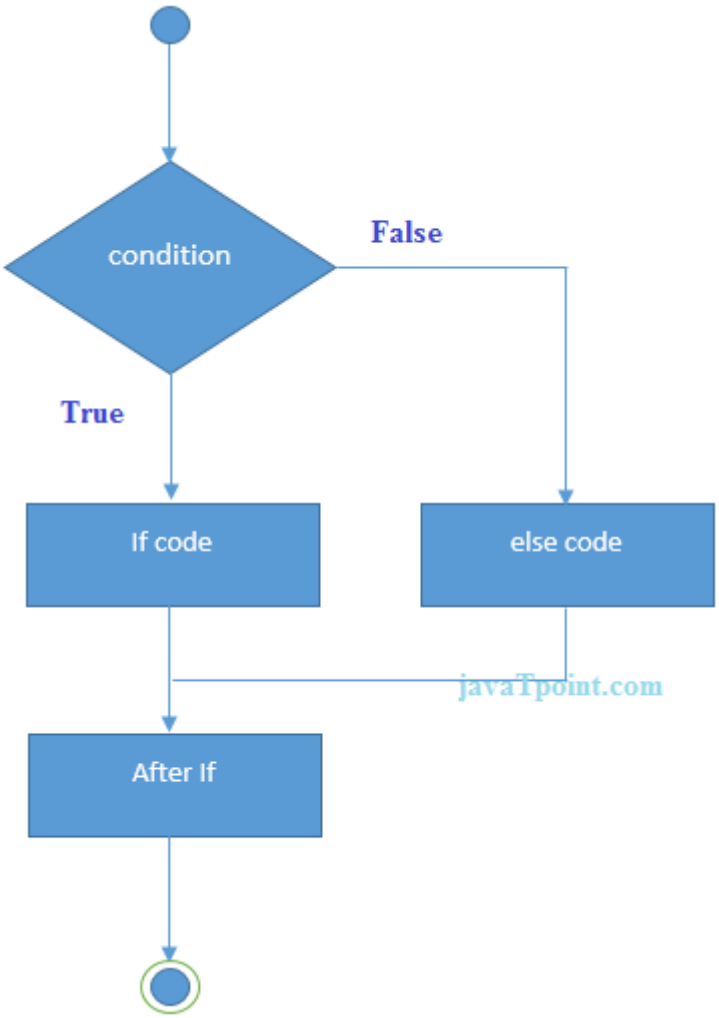
Output:/p>

```
It is even number
```

C++ IF-else Statement

The C++ if-else statement also tests the condition. It executes if block if condition is true otherwise else block is executed.

```
1. if(condition){
2. //code if condition is true
3. }else{
4. //code if condition is false
5. }
```



Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

C++ If-else Example

```
1. #include <iostream>
2. using namespace std;
3. int main () {
4.   int num = 11;
5.         if (num % 2 == 0)
6.         {
7.            cout<<"It is even number";
8.         }
9.         else
10.        {
11.           cout<<"It is odd number";
12.        }
13.  return 0;
14.}
```

Output:

```
It is odd number
```

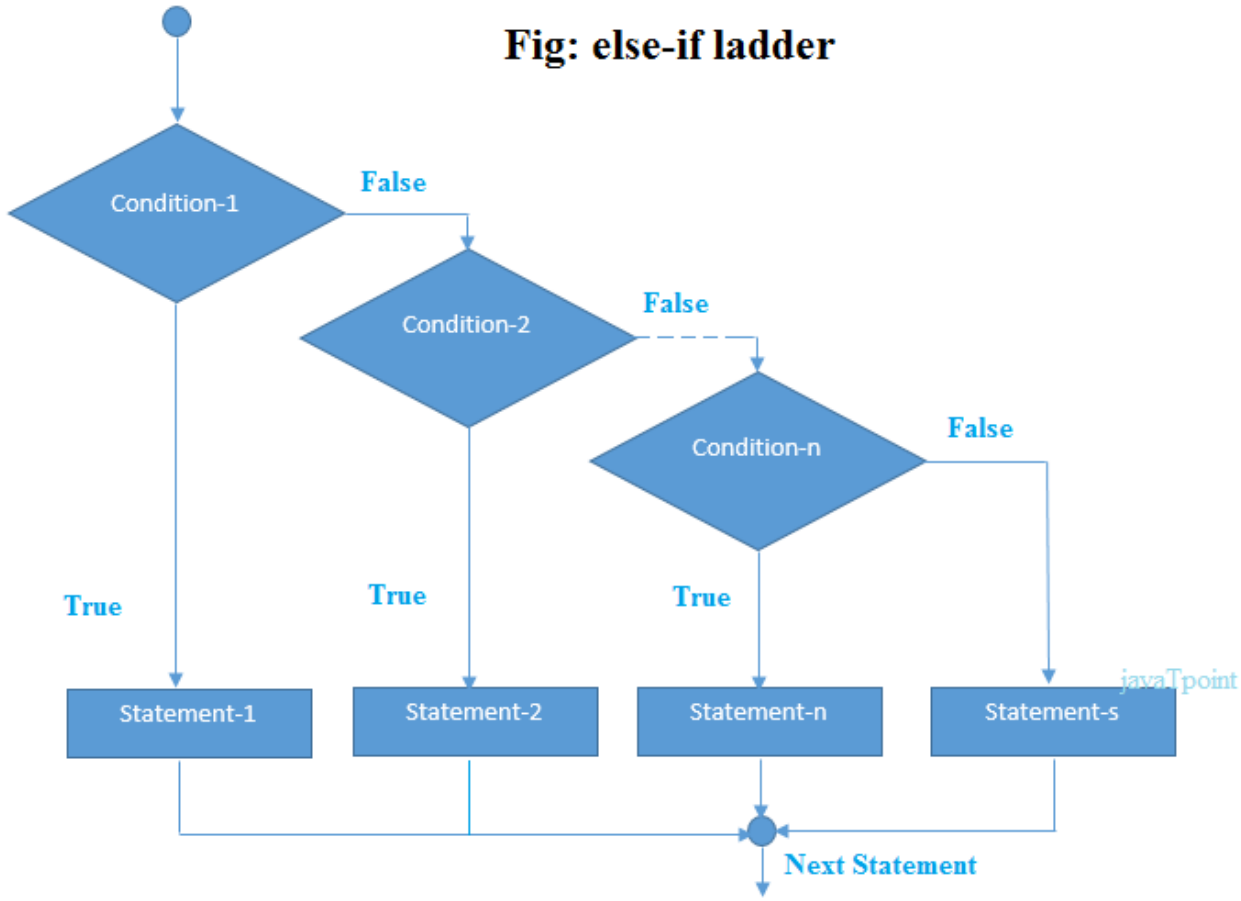
C++ IF-else-if ladder Statement

The C++ if-else-if ladder statement executes one condition from multiple statements.

```
1. if(condition1){
2. //code to be executed if condition1 is true
3. }else if(condition2){
4. //code to be executed if condition2 is true
5. }
6. else if(condition3){
7. //code to be executed if condition3 is true
8. }
9. ...
10. else{
11. //code to be executed if all the conditions are false
12. }
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:



C++ If else-if Example

```
1.  #include <iostream>
2.  using namespace std;
3.  int main () {
4.        int num;
5.        cout<<"Enter a number to check grade:";
6.        cin>>num;
7.           if (num <0 || num >100)
8.           {
9.              cout<<"wrong number";
10.          }
11.          else if(num >= 0 && num < 50){
12.             cout<<"Fail";
13.          }
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
14.         else if (num >= 50 && num < 60)
15.         {
16.            cout<<"D Grade";
17.         }
18.         else if (num >= 60 && num < 70)
19.         {
20.            cout<<"C Grade";
21.         }
22.         else if (num >= 70 && num < 80)
23.         {
24.            cout<<"B Grade";
25.         }
26.         else if (num >= 80 && num < 90)
27.         {
28.            cout<<"A Grade";
29.         }
30.         else if (num >= 90 && num <= 100)
31.         {
32.            cout<<"A+ Grade";
33.         }
34.    }
```

Output:

```
Enter a number to check grade:66
C Grade
```

Output:

```
Enter a number to check grade:-2
wrong number
```

C++ switch

The ***switch statement*** in C++ is a potent ***control structure*** that enables you to run several code segments based on the result of an expression. It offers a sophisticated and effective substitute for utilizing a succession of ***if-else-if statements*** when you have to make a decision between several possibilities.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

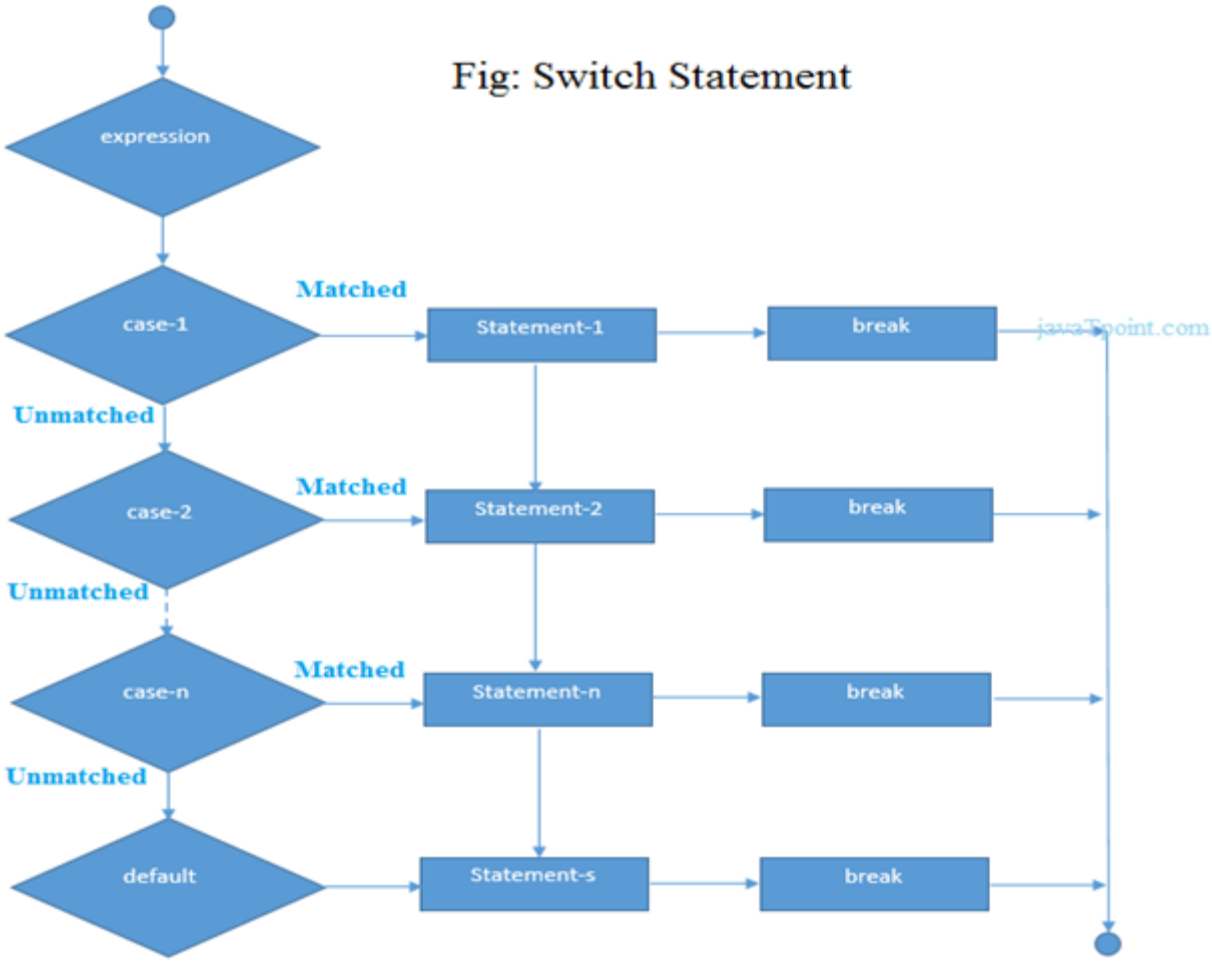
UNIT-2 Control Flow:

The C++ switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C++.

```
1. switch(expression){
2. case value1:
3.  //code to be executed;
4.  break;
5. case value2:
6.  //code to be executed;
7.  break;
8. ......
9.
10. default:
11.  //code to be executed if all cases are not matched;
12.  break;
13. }
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:



C++ Switch Example

```
1.  #include <iostream>
2.  using namespace std;
3.  int main () {
4.        int num;
5.        cout<<"Enter a number to check grade:";
6.        cin>>num;
7.          switch (num)
8.          {
9.             case 10: cout<<"It is 10"; break;
10.            case 20: cout<<"It is 20"; break;
11.            case 30: cout<<"It is 30"; break;
12.            default: cout<<"Not 10, 20 or 30"; break;
13.         }
14.   }
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

Output:

```
Enter a number:
10
It is 10
```

Output:

```
Enter a number:
55
Not 10, 20 or 30
```

C++ For Loop

The C++ for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop than while or do-while loops.

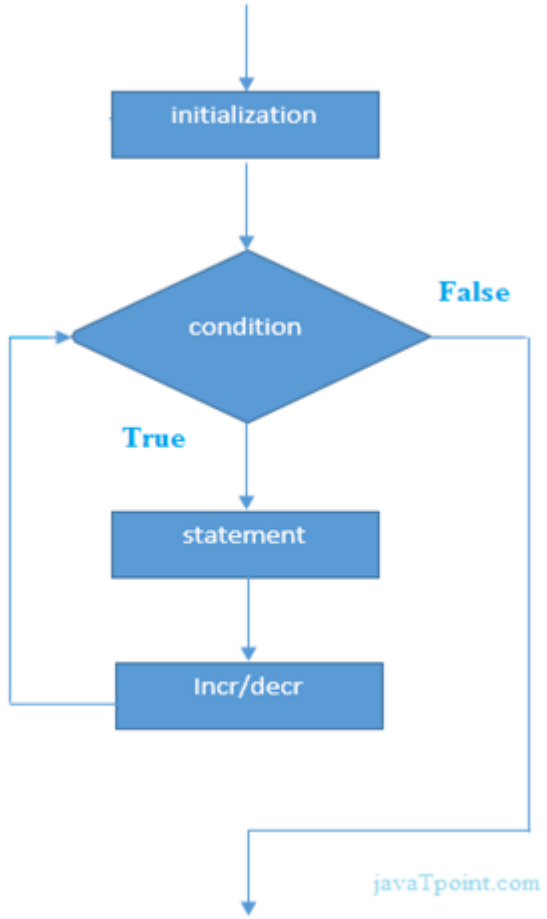
The C++ for loop is same as C/C#. We can initialize variable, check condition and increment/decrement value.

```
1. for(initialization; condition; incr/decr){
2. //code to be executed
3. }
```

Flowchart:

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:



C++ For Loop Example

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.         for(int i=1;i<=10;i++){
5.           cout<<i <<"\n";
6.         }
7.    }
```

Output:

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


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

```
8
9
10
```

C++ Nested For Loop

In C++, we can use for loop inside another for loop, it is known as nested for loop. The inner loop is executed fully when outer loop is executed one time. So if outer loop and inner loop are executed 4 times, inner loop will be executed 4 times for each outer loop i.e. total 16 times.

C++ Nested For Loop Example

Let's see a simple example of nested for loop in C++.

```
#include <iostream>
using namespace std;

int main () {
        for(int i=1;i<=3;i++){
            for(int j=1;j<=3;j++){
           cout<<i<<" "<<j<<"\n";
          }
        }
    }
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

C++ Infinite For Loop

If we use double semicolon in for loop, it will be executed infinite times. Let's see a simple example of infinite for loop in C++.

```
1. #include <iostream>
2. using namespace std;
3.
4. int main () {
5.         for (; ;)
6.          {
7.                  cout<<"Infinitive For Loop";
8.          }
9.     }
```

Output:

```
Infinitive For Loop
Infinitive For Loop
Infinitive For Loop
Infinitive For Loop
Infinitive For Loop
ctrl+c
```

C++ While loop

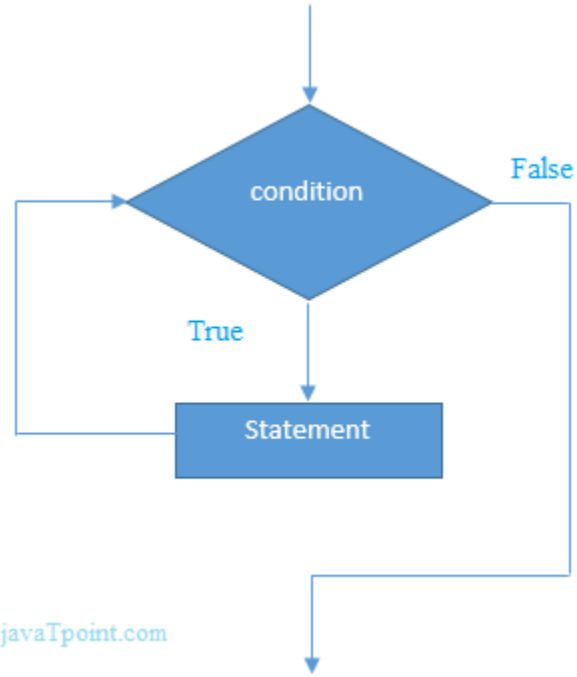
In C++, while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop than for loop.

```
1. while(condition){
2. //code to be executed
3. }
```

Flowchart:

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:



C++ While Loop Example

Let's see a simple example of while loop to print table of 1.

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.  int i=1;
5.          while(i<=10)
6.        {
7.             cout<<i <<"\n";
8.             i++;
9.          }
10.   }
```

Output:

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


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
8
9
10
```

C++ Nested While Loop Example

In C++, we can use while loop inside another while loop, it is known as nested while loop. The nested while loop is executed fully when outer loop is executed once.

Let's see a simple example of nested while loop in C++ programming language.

```
1.  #include <iostream>
2.  using namespace std;
3.  int main () {
4.          int i=1;
5.            while(i<=3)
6.            {
7.               int j = 1;
8.               while (j <= 3)
9.  {
10.             cout<<i<<" "<<j<<"\n";
11.             j++;
12.           }
13.            i++;
14.         }
15.    }
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

UNIT-2 Control Flow:

C++ Infinitive While Loop Example:

We can also create infinite while loop by passing true as the test condition.

```
1. #include <iostream>
2. using namespace std;
3. int main () {
4.         while(true)
5.          {
6.                 cout<<"Infinitive While Loop";
7.          }
8.     }
```

Output:

```
Infinitive While Loop
Infinitive While Loop
Infinitive While Loop
Infinitive While Loop
Infinitive While Loop
ctrl+c
```

C++ Do-While Loop

The C++ do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

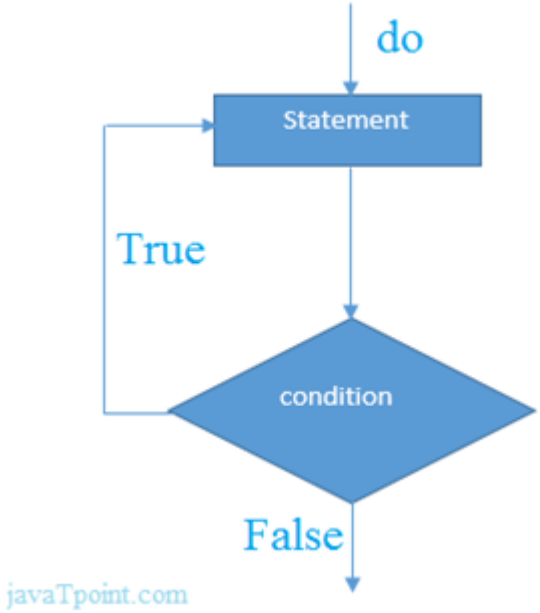
The C++ do-while loop is executed at least once because condition is checked after loop body.

```
1. do{
2. //code to be executed
3. }while(condition);
```

Flowchart:

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:



C++ do-while Loop Example

Let's see a simple example of C++ do-while loop to print the table of 1.

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     int i = 1;
5.         do{
6.             cout<<i<<"\n";
7.             i++;
8.         } while (i <= 10) ;
9. }
```

Output:

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


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

C++ Nested do-while Loop

In C++, if you use do-while loop inside another do-while loop, it is known as nested do-while loop. The nested do-while loop is executed fully for each outer do-while loop.

Let's see a simple example of nested do-while loop in C++.

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     int i = 1;
5.         do{
6.             int j = 1;
7.             do{
8.               cout<<i<<"\n";
9.                j++;
10.            } while (j <= 3) ;
11.            i++;
12.        } while (i <= 3) ;
13.}
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

C++ Infinitive do-while Loop

In C++, if you pass **true** in the do-while loop, it will be infinitive do-while loop.

```
1. do{
2. //code to be executed
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
3. }while(true);
```

C++ Infinitive do-while Loop Example

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.       do{
5.             cout<<"Infinitive do-while Loop";
6.          } while(true);
7. }
```

Output:

```
Infinitive do-while Loop
Infinitive do-while Loop
Infinitive do-while Loop
Infinitive do-while Loop
Infinitive do-while Loop
ctrl+c
```

C++ Break Statement

The C++ break is used to break loop or switch statement. It breaks the current flow of the program at the given condition. In case of inner loop, it breaks only inner loop.

```
1. jump-statement;
2. break;
```

Flowchart:

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

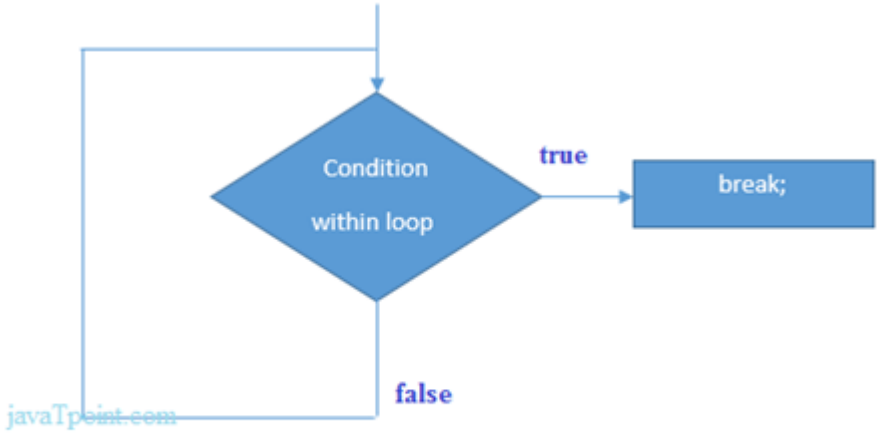


Figure: Flowchart of break statement

C++ Break Statement Example

Let's see a simple example of C++ break statement which is used inside the loop.

```
1.  #include <iostream>
2.  using namespace std;
3.  int main() {
4.      for (int i = 1; i <= 10; i++)
5.        {
6.           if (i == 5)
7.           {
8.              break;
9.           }
10.      cout<<i<<"\n";
11.       }
12. }
```

Output:

```
1
2
3
4
```

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

C++ Break Statement with Inner Loop

The C++ break statement breaks inner loop only if you use break statement inside the inner loop.

Let's see the example code:

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     for(int i=1;i<=3;i++){
6.         for(int j=1;j<=3;j++){
7.             if(i==2&&j==2){
8.                 break;
9.                 }
10.                cout<<i<<" "<<j<<"\n";
11.                }
12.        }
13.}
```

Output:

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

C++ Continue Statement

The C++ continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

```
1. jump-statement;
2. continue;
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

C++ Continue Statement Example

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     for(int i=1;i<=10;i++){
6.         if(i==5){
7.             continue;
8.         }
9.         cout<<i<<"\n";
10.    }
11.}
```

Output:

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

C++ Continue Statement with Inner Loop

C++ Continue Statement continues inner loop only if you use continue statement inside the inner loop.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.  for(int i=1;i<=3;i++){
6.         for(int j=1;j<=3;j++){
7.          if(i==2&&j==2){
8.            continue;
9.                 }
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
10.             cout<<i<<" "<<j<<"\n";
11.                 }
12.         }
13.}
```

Output:

```
1 1
1 2
1 3
2 1
2 3
3 1
3 2
3 3
```

C++ Goto Statement

The C++ goto statement is also known as jump statement. It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label.

It can be used to transfer control from deeply nested loop or switch case label.

C++ Goto Statement Example

Let's see the simple example of goto statement in C++.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5. ineligible:
6.         cout<<"You are not eligible to vote!\n";
7.     cout<<"Enter your age:\n";
8.     int age;
9.     cin>>age;
10.    if (age < 18){
11.         goto ineligible;
12.    }
13.    else
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
14.     {
15.         cout<<"You are eligible to vote!";
16.     }
17. }
```

Output:

```
You are not eligible to vote!
Enter your age:
16
You are not eligible to vote!
Enter your age:
7
You are not eligible to vote!
Enter your age:
22
You are eligible to vote!
```

++ OOPs Concepts

The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.

Object Oriented Programming is a paradigm that provides many concepts such as **inheritance, data binding, polymorphism etc.**

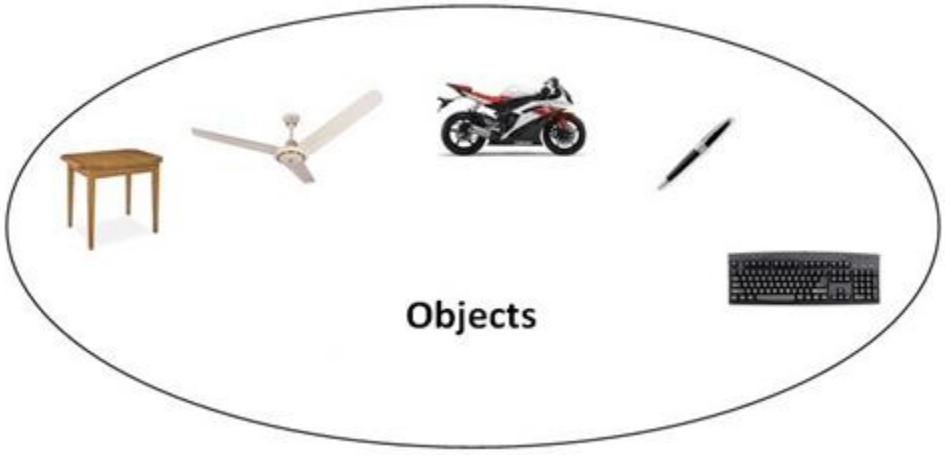
The programming paradigm where everything is represented as an object is known as truly object-oriented programming language. **Smalltalk** is considered as the first truly object-oriented programming language.

OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:



- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

A Class in C++ is the foundational element that leads to Object-Oriented programming. A class instance must be created in order to access and use the user-defined data type's data members and member functions. An object's class acts as its blueprint. Take the class of cars as an example. Even if different names and brands may be used for different cars, all of them will have some characteristics in common, such as four wheels, a speed limit, a range of miles, etc. In this case, the class of car is represented by the wheels, the speed limitations, and the mileage.

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

1. Sub class - Subclass or Derived Class refers to a class that receives properties from another class.
2. Super class - The term "Base Class" or "Super Class" refers to the class from which a subclass inherits its properties.
3. Reusability - As a result, when we wish to create a new class, but an existing class already contains some of the code we need, we can generate our new class from the old class thanks to inheritance. This allows us to utilize the fields and methods of the pre-existing class.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

Different situations may cause an operation to behave differently. The type of data utilized in the operation determines the behavior.

Abstraction

Hiding internal details and showing functionality is known as abstraction. Data abstraction is the process of exposing to the outside world only the information that is absolutely necessary while concealing implementation or background information.For example: phone call, we don't know the internal processing.

In C++, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

Encapsulation is typically understood as the grouping of related pieces of information and data into a single entity. Encapsulation is the process of tying together data and the functions that work with it in object-oriented programming. Take a look at a practical illustration of encapsulation: at a company, there are various divisions, including the sales division, the finance division, and the accounts division. All financial transactions are handled by the finance sector, which also maintains records of all financial data. In a similar vein, the sales section is in charge of all tasks relating to sales and maintains a record of each sale. Now, a scenario could occur when, for some

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

reason, a financial official requires all the information on sales for a specific month. Under the umbrella term "sales section," all of the employees who can influence the sales section's data are grouped together. Data abstraction or concealing is another side effect of encapsulation. In the same way that encapsulation hides the data. In the aforementioned example, any other area cannot access any of the data from any of the sections, such as sales, finance, or accounts.

Dynamic Binding - In dynamic binding, a decision is made at runtime regarding the code that will be run in response to a function call. For this, C++ supports virtual functions.

Advantage of OOPs over Procedure-oriented programming language

1. OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
2. OOPs provide data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
3. OOPs provide ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

Why do we need oops in C++?

There were various drawbacks to the early methods of programming, as well as poor performance. The approach couldn't effectively address real-world issues since, similar to procedural-oriented programming, you couldn't reuse the code within the program again, there was a difficulty with global data access, and so on.

With the use of classes and objects, object-oriented programming makes code maintenance simple. Because inheritance allows for code reuse, the program is simpler because you don't have to write the same code repeatedly. Data hiding is also provided by ideas like encapsulation and abstraction.

Why is C++ a partial oop?

The object-oriented features of the C language were the primary motivation behind the construction of the C++ language.

The C++ programming language is categorized as a partial object-oriented programming language despite the fact that it supports OOP concepts, including classes, objects, inheritance, encapsulation, abstraction, and polymorphism.

UNIT-2 Control Flow:

1) The main function must always be outside the class in C++ and is required. This means that we may do without classes and objects and have a single main function in the application.

It is expressed as an object in this case, which is the first time Pure OOP has been violated.

2) Global variables are a feature of the C++ programming language that can be accessed by any other object within the program and are defined outside of it. Encapsulation is broken here. Even though C++ encourages encapsulation for classes and objects, it ignores it for global variables.

Overloading

Polymorphism also has a subset known as overloading. An existing operator or function is said to be overloaded when it is forced to operate on a new data type.

Conclusion

You will have gained an understanding of the need for object-oriented programming, what C++ OOPs are, and the fundamentals of OOPs, such as polymorphism, inheritance, encapsulation, etc., after reading this course on OOPS Concepts in C++. Along with instances of polymorphism and inheritance, you also learned about the benefits of C++ OOPs.

C++ Object and Class

Since C++ is an object-oriented language, program is designed using objects and classes in C++.

C++ Object

In C++, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.

In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.

Object is a runtime entity, it is created at runtime.

Object is an instance of a class. All the members of the class can be accessed through object.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

Let's see an example to create object of student class using s1 as the reference variable.

```
1. Student s1;  //creating an object of Student
```

In this example, Student is the type and s1 is the reference variable that refers to the instance of Student class.

C++ Class

In C++, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Let's see an example of C++ class that has three fields only.

```
1. class Student
2.  {
3.      public:
4.      int id;  //field or data member
5.      float salary; //field or data member
6.      String name;//field or data member
7.  }
```

C++ Object and Class Example

Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

```
1. #include <iostream>
2. using namespace std;
3. class Student {
4.   public:
5.      int id;//data member (also instance variable)
6.      string name;//data member(also instance variable)
7. };
8. int main() {
9.     Student s1; //creating an object of Student
10.    s1.id = 201;
11.    s1.name = "Sonoo Jaiswal";
```

UNIT-2 Control Flow:

```
12.    cout<<s1.id<<endl;
13.    cout<<s1.name<<endl;
14.    return 0;
15. }
```

Output:

```
201
Sonoo Jaiswal
```

C++ Class Example: Initialize and Display data through method

Let's see another example of C++ class where we are initializing and displaying object through method.

```
1.  #include <iostream>
2.  using namespace std;
3.  class Student {
4.    public:
5.       int id;//data member (also instance variable)
6.       string name;//data member(also instance variable)
7.       void insert(int i, string n)
8.        {
9.            id = i;
10.           name = n;
11.       }
12.      void display()
13.       {
14.           cout<<id<<"  "<<name<<endl;
15.       }
16. };
17. int main(void) {
18.    Student s1; //creating an object of Student
19.    Student s2; //creating an object of Student
20.    s1.insert(201, "Sonoo");
21.    s2.insert(202, "Nakul");
22.    s1.display();
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
23.    s2.display();
24.    return 0;
25. }
```

Output:

```
201   Sonoo
202   Nakul
```

C++ Class Example: Store and Display Employee Information

Let's see another example of C++ class where we are storing and displaying employee information using method.

```
1.  #include <iostream>
2.  using namespace std;
3.  class Employee {
4.    public:
5.       int id;//data member (also instance variable)
6.       string name;//data member(also instance variable)
7.       float salary;
8.       void insert(int i, string n, float s)
9.        {
10.          id = i;
11.          name = n;
12.          salary = s;
13.       }
14.      void display()
15.       {
16.          cout<<id<<"  "<<name<<"  "<<salary<<endl;
17.       }
18. };
19. int main(void) {
20.    Employee e1; //creating an object of Employee
21.    Employee e2; //creating an object of Employee
22.    e1.insert(201, "Sonoo",990000);
23.    e2.insert(202, "Nakul", 29000);
```

UNIT-2 Control Flow:

```
24.    e1.display();
25.    e2.display();
26.    return 0;
27. }
```

Output:

```
201   Sonoo   990000
202   Nakul   29000
```

C++ Constructor

In C++, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C++ has the same name as class or structure.

In brief, A particular procedure called a constructor is called automatically when an object is created in C++. In general, it is employed to create the data members of new things. In C++, the class or structure name also serves as the constructor name. When an object is completed, the constructor is called. Because it creates the values or gives data for the thing, it is known as a constructor.

- Default constructor
- Parameterized constructor

C++ Default Constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

Let's see the simple example of C++ default Constructor.

```
1.  #include <iostream>
2.  using namespace std;
3.  class Employee
4.  {
5.    public:
6.        Employee()
7.        {
8.            cout<<"Default Constructor Invoked"<<endl;
9.        }
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
10. };
11. int main(void)
12. {
13.     Employee e1; //creating an object of Employee
14.     Employee e2;
15.     return 0;
16. }
```

Output:

```
Default Constructor Invoked
Default Constructor Invoked
```

C++ Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

Let's see the simple example of C++ Parameterized Constructor.

```
1.  #include <iostream>
2.  using namespace std;
3.  class Employee {
4.    public:
5.       int id;//data member (also instance variable)
6.       string name;//data member(also instance variable)
7.       float salary;
8.       Employee(int i, string n, float s)
9.        {
10.          id = i;
11.          name = n;
12.          salary = s;
13.       }
14.      void display()
15.       {
16.          cout<<id<<"  "<<name<<"  "<<salary<<endl;
17.       }
18. };
19. int main(void) {
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
20.    Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Em
       ployee
21.    Employee e2=Employee(102, "Nakul", 59000);
22.    e1.display();
23.    e2.display();
24.    return 0;
25. }
```

Output:

```
101   Sonoo   890000
102   Nakul   59000
```

What distinguishes constructors from a typical member function?

1. Constructor's name is the same as the class's
2. Default There isn't an input argument for constructors. However, input arguments are available for copy and parameterized constructors.
3. There is no return type for constructors.
4. An object's constructor is invoked automatically upon creation.
5. It must be shown in the classroom's open area.
6. The C++ compiler creates a default constructor for the object if a constructor is not specified (expects any parameters and has an empty body).

What are the characteristics of a constructor?

1. The constructor has the same name as the class it belongs to.
2. Although it is possible, constructors are typically declared in the class's public section. However, this is not a must.
3. Because constructors don't return values, they lack a return type.
4. When we create a class object, the constructor is immediately invoked.
5. Overloaded constructors are possible.
6. Declaring a constructor virtual is not permitted.
7. One cannot inherit a constructor.
8. Constructor addresses cannot be referenced to.
9. When allocating memory, the constructor makes implicit calls to the new and delete operators.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

What is a copy constructor?

A member function known as a copy constructor initializes an item using another object from the same class-an in-depth discussion on Copy Constructors.

Every time we specify one or more non-default constructors (with parameters) for a class, we also need to include a default constructor (without parameters), as the compiler won't supply one in this circumstance. The best practice is to always declare a default constructor, even though it is not required.

A reference to an object belonging to the same class is required by the copy constructor.

```
1. Sample(Sample &t)
2. {
3. id=t.id;
4. }
```

C++ Destructor

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).

Note: C++ destructor cannot have parameters. Moreover, modifiers can't be applied on destructors.

C++ Constructor and Destructor Example

Let's see an example of constructor and destructor in C++ which is called automatically.

```
1. #include <iostream>
2. using namespace std;
3. class Employee
4. {
5.   public:
6.       Employee()
7.       {
8.           cout<<"Constructor Invoked"<<endl;
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
9.        }
10.       ~Employee()
11.       {
12.           cout<<"Destructor Invoked"<<endl;
13.       }
14. };
15. int main(void)
16. {
17.    Employee e1; //creating an object of Employee
18.    Employee e2; //creating an object of Employee
19.    return 0;
20. }
```

Output:

```
Constructor Invoked
Constructor Invoked
Destructor Invoked
Destructor Invoked
```

Data Abstraction in C++

- Data Abstraction is a process of providing only the essential details to the outside world and hiding the internal details, i.e., representing only the essential details in the program.
- Data Abstraction is a programming technique that depends on the seperation of the interface and implementation details of the program.
- Let's take a real life example of AC, which can be turned ON or OFF, change the temperature, change the mode, and other external components such as fan, swing. But, we don't know the internal details of the AC, i.e., how it works internally. Thus, we can say that AC seperates the implementation details from the external interface.
- C++ provides a great level of abstraction. For example, pow() function is used to calculate the power of a number without knowing the algorithm the function follows.

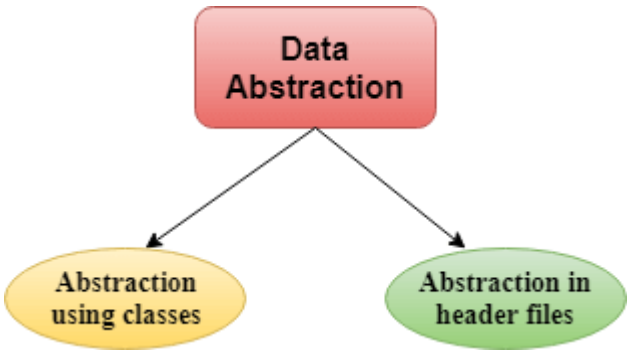
Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

In C++ program if we implement class with private and public members then it is an example of data abstraction.

Data Abstraction can be achieved in two ways:

- Abstraction using classes
- Abstraction in header files.



Abstraction using classes: An abstraction can be achieved using classes. A class is used to group all the data members and member functions into a single unit by using the access specifiers. A class has the responsibility to determine which data member is to be visible outside and which is not.

Abstraction in header files: An another type of abstraction is header file. For example, pow() function available is used to calculate the power of a number without actually knowing which algorithm function uses to calculate the power. Thus, we can say that header files hides all the implementation details from the user.

Access Specifiers Implement Abstraction:

- **Public specifier:** When the members are declared as public, members can be accessed anywhere from the program.
- **Private specifier:** When the members are declared as private, members can only be accessed only by the member functions of the class.

Let's see a simple example of abstraction in header files.

// program to calculate the power of a number.

```
1. #include <iostream>
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
2. #include<math.h>
3. using namespace std;
4. int main()
5. {
6.  int n = 4;
7.    int power = 3;
8.    int result = pow(n,power);        // pow(n,power) is the  power function
9.    std::cout << "Cube of n is : " <<result<< std::endl;
10.   return 0;
11. }
```

Output:

```
Cube of n is : 64
```

In the above example, pow() function is used to calculate 4 raised to the power 3. The pow() function is present in the math.h header file in which all the implementation details of the pow() function is hidden.

Let's see a simple example of data abstraction using classes.

```
1. #include <iostream>
2. using namespace std;
3.  class Sum
4. {
5. private: int x, y, z; // private variables
6. public:
7. void add()
8. {
9. cout<<"Enter two numbers: ";
10. cin>>x>>y;
11. z= x+y;
12. cout<<"Sum of two number is: "<<z<<endl;
13. }
14. };
15. int main()
16. {
17. Sum sm;
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

UNIT-2 Control Flow:

```
18. sm.add();
19. return 0;
20. }
```

Output:

```
Enter two numbers:
3
6
Sum of two number is: 9
```

In the above example, abstraction is achieved using classes. A class 'Sum' contains the private members x, y and z are only accessible by the member functions of the class.

Advantages Of Abstraction:

- Implementation details of the class are protected from the inadvertent user level errors.
- A programmer does not need to write the low level code.
- Data Abstraction avoids the code duplication, i.e., programmer does not have to undergo the same tasks every time to perform the similar operation.
- The main aim of the data abstraction is to reuse the code and the proper partitioning of the code across the classes.
- Internal implementation can be changed without affecting the user level code.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com