C++ Access Specifiers

Public

Private

Protected

By – Mohammad Imran

Access Specifier

Introduction

Data hiding is an important concept of Object-Oriented Programming, implemented with some Access modifiers' help. It is also known as Access Specifier.

Access Specifiers in a class decide the accessibility of the class members, like variables or methods in other classes. That is, it will decide whether the class members or methods will get directly accessed by the blocks present outside the class or not, depending on the type of Access Specifier.

By – Mohammad Imran

Access Specifier

Types

There are three types of access modifiers in C++

- ✓ Public
- ✓ Private
- ✓ Protected

By – Mohammad Imran

Access Specifier

Public

This keyword is used to declare the functions and variables public, and any part of the entire program can access it. The members and member methods declared public can be accessed by other classes and functions.

The public members of a class can be accessed from anywhere in the program using the **(.)** with the object of that class.

By – Mohammad Imran

Access Specifier | Public | Example

```
class Test
{
  public:
   int x;
  public:
   void display()
   {
      cout << "Hello" << endl;
   }
};
```

NOTE: Variable x and function display is public scope it means can be access anywhere outside the class body.

Access Specifier

Private

The **private keyword** is used to create private variables or private functions. The private members can only be accessed from within the class. Only the member functions or the friend functions are allowed to access the private data of a class or the methods of a class.

Note -

- ✓ Protected and Private data members or class methods can be accessed using a function only if that function is declared as the friend function.
- ✓ We can use the keyword friend to ensure the compiler understands and make the data accessible to that function.

By – Mohammad Imran

Access Specifier

Protected

The protected keyword is used to create protected variables or protected functions. The protected members can be accessed within and from the derived / child class.

Note - A class created or derived from another existing class (base class) is known as a derived class. The base class is also known as a superclass. It is created and derived through the process of inheritance.

By – Mohammad Imran

Access Specifier

Example - 1

```
class Employee
{
  private:
   int employeeSalary;
  public:
   int employeeId;
   string employeeName;
   string getEmployeeName()
   {
      return employeeName;
   }

  protected:
   void setEmployeeSalary(int n)
   {
      employeeSalary = n;
      return;
   }
};
```

OUTPUT

By – Mohammad Imran

Inheritance in C++

- Single Inheritance — 01
- Multiple Inheritance — 02
- Multilevel Inheritance — 03
- Hierarchical Inheritance — 04
- Hybrid Inheritance — 05

By – Mohammad Imran

Inheritance

Introduction

Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.

In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

By – Mohammad Imran

Inheritance

Reusability

Code reusability: Now you can reuse the members of your parent class. So, there is no need to define the member again. So less code is required in the class.

We use inheritance in C++ for the reusability of code from the existing class. C++ strongly supports the concept of reusability. Reusability is yet another essential feature of OOP(Object Oriented Programming).

It is always good to reuse something that already exists rather than trying to create the one that is already present, as it saves time and increases reliability.

By – Mohammad Imran

Inheritance

Syntax

```
class Derived_className() : access_specifier Base_className
{
   Statements;
}
```

NOTE: Colon (:) is used for inheritance.

Example

```
class Base
{
  public:
   function_Name();
};
```

```
class Derived : public Base
{
  public:
     function_Name();
}
```

By – Mohammad Imran

Inheritance

Mode

There are three modes of inheritance:

- ✓ Public Mode
- ✓ Protected Mode
- ✓ Private Mode

By – Mohammad Imran

Mode of Inheritance

Public Mode

In the public mode of inheritance, when a child class is derived from the base or parent class, then the public member of the base class or parent class will become public in the child class also, in the same way, the protected member of the base class becomes protected in the child class, and private members of the base class are not accessible in the derived class.

By – Mohammad Imran

Mode of Inheritance

Public Mode | **Example**

```
#include <bits/stdc++.h>
using namespace std;
class Parent
{
  public:
   int a;
  private:
   int b;
  protected:
   int c;
};
class Child1 : public Parent
{
    // Data members.
    // Member functions.
};
```

```
class Child2 : public Parent
{
    // Data members.
    // Member functions.
};

class Child3 : public Parent
{
    // Data members.
    // Member functions.
};
```

By – Mohammad Imran

Mode of Inheritance

Public Mode | **Example**

```
int main()
{
   Child1 x;
   Child2 y;
   Child3 z;

   cout << x.a << endl; // Accessible.
   cout << x.b << endl; // Not accessible.
   cout << x.c << endl; // Not accessible.

   cout << y.a << endl; // Accessible.
   cout << y.b << endl; // Not accessible.
   cout << y.c << endl; // Not accessible.

   cout << z.a << endl; // Accessible.
   cout << z.b << endl; // Not accessible.
   cout << z.c << endl; // Not accessible.
}
```

By – Mohammad Imran

Mode of Inheritance

Protected Mode

In protected mode, when a child class is derived from a base class or parent class, then both public and protected members of the base class will become protected in the derived class, and private members of the base class are again not accessible in the derived class. In contrast, protected members can be easily accessed in the derived class.

By – Mohammad Imran

Mode of Inheritance

Protected Mode

Example

```
#include <bits/stdc++.h>
using namespace std;
class Parent
{
  public:
   int a;
  private:
   int b;
  protected:
   int c;
};
class Child1 : protected Parent
{
    // Data members.
    // Member functions.
};
```

```
class Child2 : protected Parent
{
    // Data members.
    // Member functions.
};

class Child3 : protected Parent
{
    // Data members.
    // Member functions.
};
```

By – Mohammad Imran

```
int main()
{
   Child1 x;
   Child2 y;
   Child3 z;

   cout << x.a << endl; // Not accessible.
   cout << x.b << endl; // Not accessible.
   cout << x.c << endl; // Not accessible.

   cout << y.a << endl; // Not accessible.
   cout << y.b << endl; // Not accessible.
   cout << y.c << endl; // Not accessible.

   cout << z.a << endl; // Not accessible.
   cout << z.b << endl; // Not accessible.
   cout << z.c << endl; // Not accessible.
}
```

Mode of Inheritance

Private Mode

In private mode, when a child class is derived from a base class, then both public and protected members of the base class will become private in the derived class, and private members of the base class are again not accessible in the derived class.

By – Mohammad Imran

Mode of Inheritance

Access Table

Access Specifier	Own Class	Derived Class	Main Function
Public	Yes	Yes	Yes
Protected	Yes	Yes	No
Private	Yes	No	No

By – Mohammad Imran

Mode of Inheritance

Access Table

Base Class Member Access Specifier	Type of Inheritance		
	Public	**Protected**	**Private**
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not Accessible (Hidden)	Not Accessible (Hidden)	Not Accessible (Hidden)

By – Mohammad Imran

Inheritance

Types

C++ supports five types of inheritance:

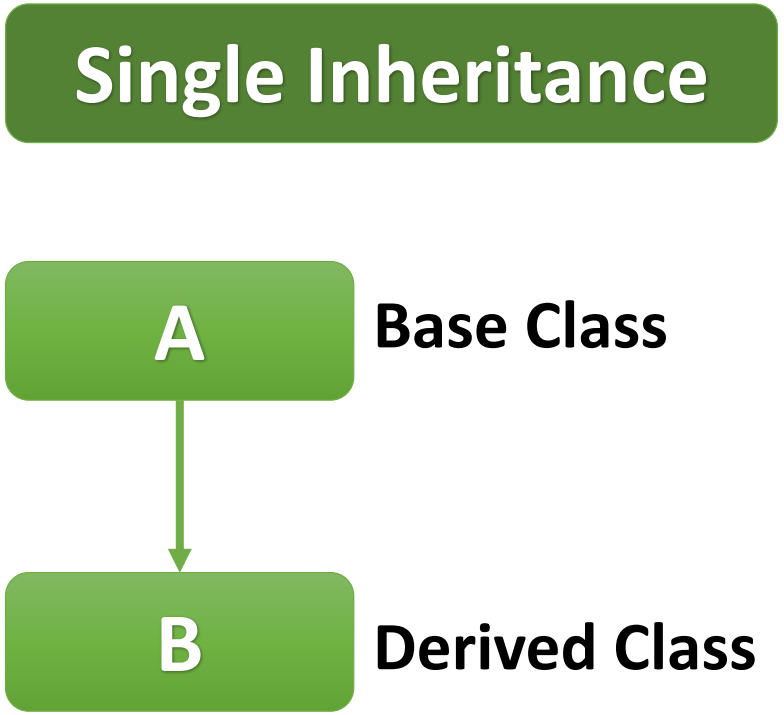
- ✓ Single Inheritance
- ✓ Multiple Inheritance
- ✓ Hierarchical Inheritance
- ✓ Multilevel Inheritance
- ✓ Hybrid Inheritance

By – Mohammad Imran

Types of Inheritance

Single Inheritance

When the derived class inherits only one base class, it is known as Single Inheritance.



By – Mohammad Imran

Single Inheritance

Example - 2

```
#include<iostream>
using namespace std;
class Base
{
  public:
   float salary;
   void basic()
   {
      salary = 900;
   }
};
class Derived : public Base
{
  public:
   float bonus;
   void gross()
   {
      bonus = 100;
   }
   void sum()
   {
      cout << "Total Salary = “;
      cout << (salary + bonus) << endl;
   }
};
```

By – Mohammad Imran

Single Inheritance

Example - 2

```
int main()
{
   Derived d;
   d.basic();
   d.gross();
   cout << "Salary = " << d.salary << endl;
   cout << "Bonus = " << d.bonus << endl;
   d.sum();
   return 0;
}
```

OUTPUT

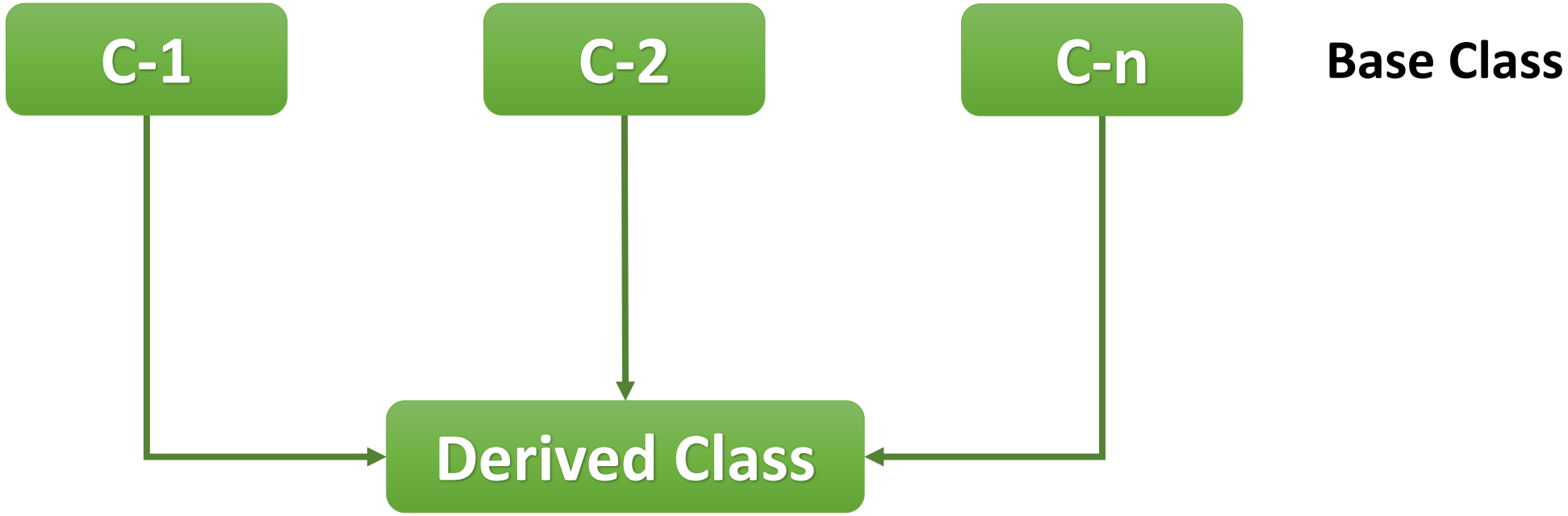
```
Salary = 10000
Bonus  = 2000
Total Salary = 12000
```

By – Mohammad Imran

Types of Inheritance

Multiple Inheritance

When a derived class(child class) inherits more than one base class(parent class), it is called multiple inheritance.



By – Mohammad Imran

Multiple Inheritance

Example - 3

```
#include<iostream>
using namespace std;
class A
{
  protected:
     int a;
  public:
    void get_a(int n)
    {
        a = n;
    }
};
```

```
class B
{
  protected:
     int b;
  public:
    void get_b(int n)
    {
        b = n;
    }
};
```

By – Mohammad Imran

Multiple Inheritance

Example - 3

```
class C : public A, public B
{
  public:
   void display()
   {
      cout << "a = " << a << endl;
      cout << "b = " << b << endl;
      cout << "Sum = " << a+b;
   }
};
```

OUTPUT

```
a = 10
b = 20
Sum = 30
```

```
int main()
{
      C c;
      c.get_a(10);
      c.get_b(20);
      c.display();
      return 0;
}
```

By – Mohammad Imran

Ambiguity in Inheritance

Ambiguity can be occurred in using the multiple inheritance when a function with the same name occurs in more than one base class.

By – Mohammad Imran

Ambiguity in Single Inheritance

Example - 4

```
#include<iostream>
using namespace std;
class A
{
  public:
   void display()
   {
      cout << "Class A" << endl;
   }
};
```

```
class B : public A
{
  public:
   void display()
   {
      cout<< "Class B" << endl;
   }
};
```

By – Mohammad Imran

Ambiguity in Single Inheritance

Example - 4

```
int main()
{
   B b;
   b.A :: display();
   b.display();
   return 0;
}
```

OUTPUT

```
Class A
Class B
```

By – Mohammad Imran

Ambiguity in Multiple Inheritance

Example - 5

```
include <iostream>
using namespace std;
class A
{
  public:
   void display()
   {
      cout << "Class A" << endl;
   }
};
```

```
class B : public A
{
  public:
   void display()
   {
      cout<< "Class B" << endl;
   }
};
```

By – Mohammad Imran

Ambiguity in Multiple Inheritance

Example - 5

```
class C : public A, public B
{
  public:
   void view()
   {
      A::display();
      B::display();
      cout << "Class C" << endl;
   }
};
```

```
int main()
{
   C c;
   c.view();
   return 0;
}
```

OUTPUT

```
Class A
Class B
Class C
```

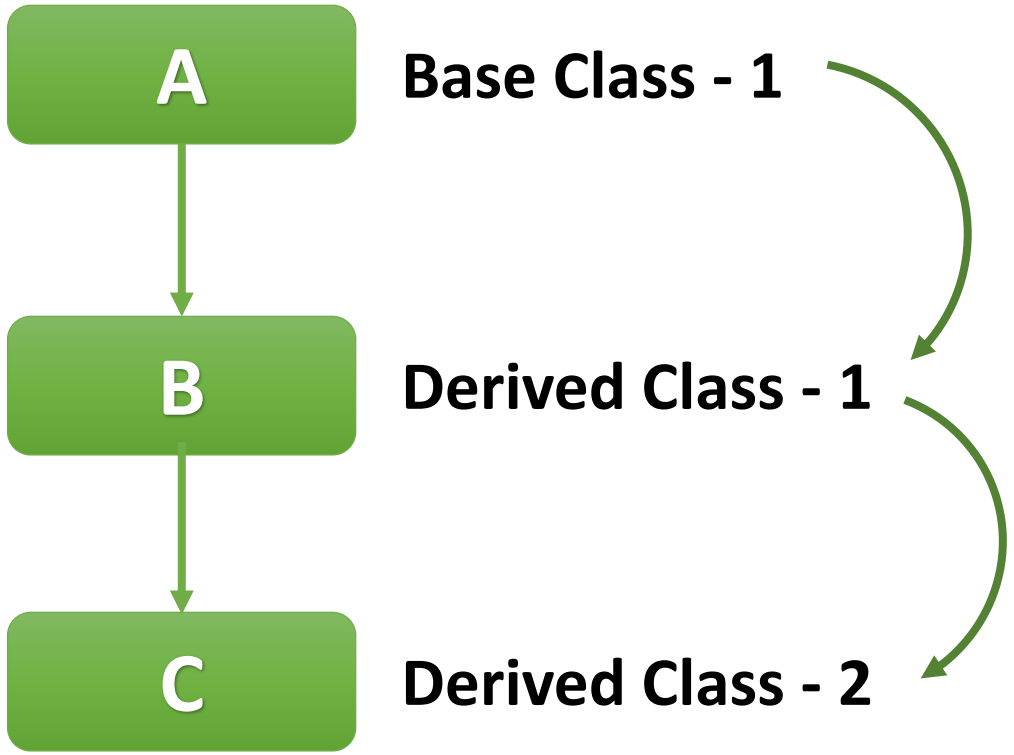
By – Mohammad Imran

Types of Inheritance

Multilevel Inheritance

Multilevel inheritance is a process of deriving a class from another derived class.

When a derived (child) class inherits the base class and acts as the base class (parent class) to the other class, it is called Multilevel Inheritance. There can be any number of levels.



By – Mohammad Imran

Multilevel Inheritance

Example - 6

```
#include <iostream>
using namespace std;
class Animal
{
  public:
   void eat()
   {
      cout << "Eating" << endl;
   }
};
```

```
class Dog: public Animal
{
  public:
   void bark()
   {
      cout << "Barking" << endl;
   }
};
```

By – Mohammad Imran

Multilevel Inheritance

Example - 6

```
class BabyDog: public Dog
{
  public:
   void weep()
   {
      cout << "Weeping...";
   }
};
```

```
int main(void)
{
      BabyDog bd;
      bd.eat();
      bd.bark();
      bd.weep();
      return 0;
}
```

OUTPUT

```
Eating
Barking
Weeping...
```

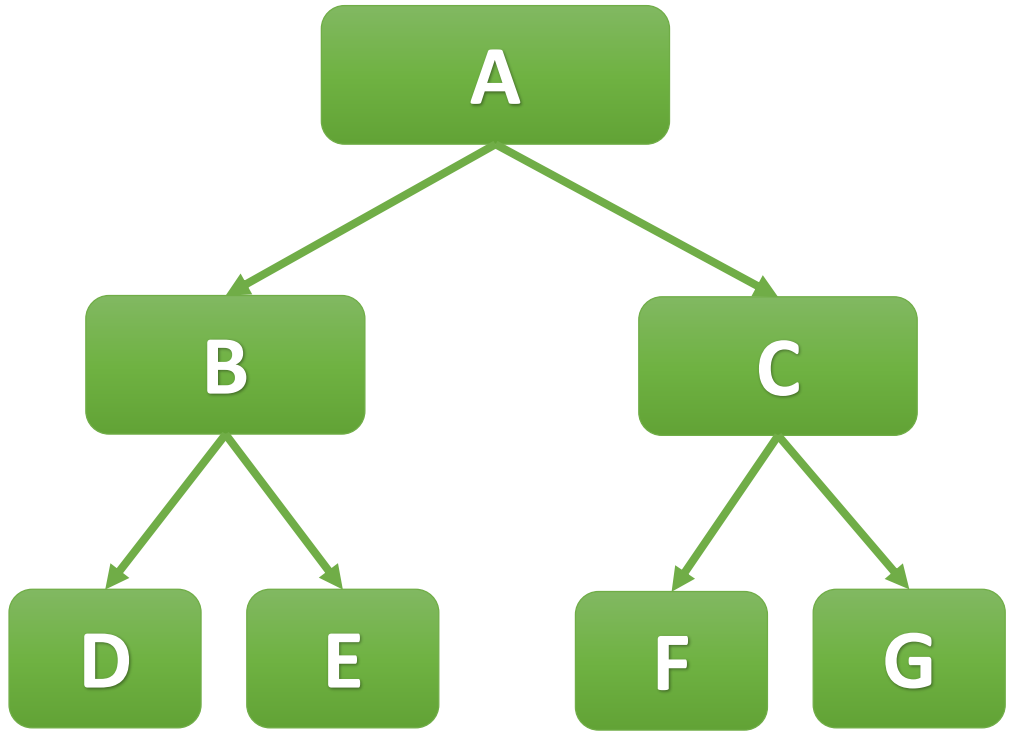
By – Mohammad Imran

Types of Inheritance

Hierarchical Inheritance

When more than one class is inherited from a single base class, we call it Hierarchical Inheritance. The child classes inherit the features of the parent class. It allows code reusability and improves the readability of code.

As the base class features are reused as often as needed, it improves maintainability and reduces development costs.



By – Mohammad Imran

Hierarchical Inheritance

Syntax

```
class Parent
{
    // Data members.
    // Member functions.
};

class child1 : visibility_mode Parent
{
    // Data members.
    // Member functions.
};

class child2 : visibility_mode Parent
{
    // Data members.
    // Member functions.
};
```

By – Mohammad Imran

Hierarchical Inheritance

Parent : This is the base class, which contains the common characteristics of the derived classes. The derived classes inherit features from this class.

child1 : First derived class inheriting properties from the base class.

child2 : Second derived class inheriting properties from the base class.

visibility_mode : Visibility modes control the accessibility of the base class by its derived classes.

By – Mohammad Imran

Hierarchical Inheritance

Example - 7

```
#include <iostream>
using namespace std;
class DataEntry
{
  public:
   int n1, n2;
   void inputData()
   {
      cout << "Enter n1 : ";
      cin >> n1;
      cout << "Enter n2 : ";
      cin >> n2;
   }
};
```

By – Mohammad Imran

Hierarchical Inheritance

Example - 7

```
class Addition : public DataEntry
{
  public:
   void sum()
   {
      cout << "Sum = " << n1 + n2 << endl;
   }
};

class Product : public DataEntry
{
  public:
   void multi()
   {
      cout << "Product = " << n1 * n2 << endl;
   }
};
```

By – Mohammad Imran

Hierarchical Inheritance

Example - 7

```
int main()
{
      Addition ad;
      ad.inputData();
      ad.sum();

      Product pd;
      pd.inputData();
      pd.multi();
      return 0;
}
```

OUTPUT

```
Enter n1 : 20
Enter n2 : 50
Sum = 70

Enter n1 : 25
Enter n2 : 4
Product = 100
```

By – Mohammad Imran

Hierarchical Inheritance

Example - 8

```
#include <iostream>
using namespace std;
class Shape
{
  public:
   int a;
   int b;
   void get_data(int n, int m)
   {
      a = n;
      b = m;
   }
};
```

```
class Rectangle : public Shape
{
  public:
   int rect_area()
   {
      int result = a*b;
      return result;
   }
};
```

By – Mohammad Imran

Hierarchical Inheritance

Example - 8

```
class Triangle : public Shape
{
  public:
   int triangle_area()
   {
      float result = 0.5*a*b;
      return result;
   }
};
int main()
{
   Rectangle r;
   Triangle t;
   int l, b, base, h;
   cout << "Enter length : ";
   cin >> l;
   cout << "Enter breadth : ";
   cin >> b;
   r.get_data(l,b);
   int m = r.rect_area();
   cout << "Area of the rectangle is :
   " << m << endl;
   cout << "Enter base : ";
   cin >> base;
   cout << "Enter height : ";
   cin >> h;
   t.get_data(base,h);
   float n = t.triangle_area();
   cout <<"Area of the triangle is : "
   << n;
   return 0;
}
```

By – Mohammad Imran

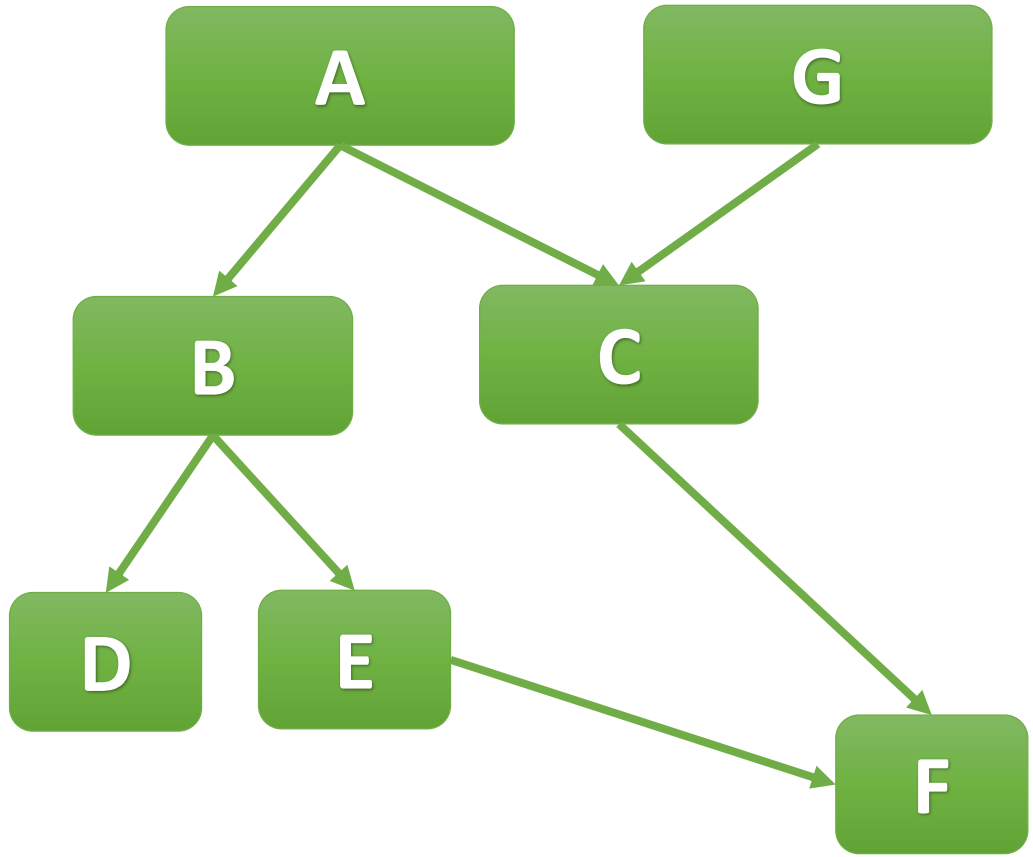
Types of Inheritance

Hybrid Inheritance

When we combine more than one type of **inheritance**, it is called hybrid inheritance in C++.

It is also referred to as a **multipath inheritance** because many types of inheritances get involved.

For example, multiple inheritances can be combined with the single or multilevel inheritance.



By – Mohammad Imran

Hybrid Inheritance

Syntax

```
class A
{
   // data members and member functions()
};
class B: public A  // single inheritance
{
   // data members and member functions();
};
class C
{
   // data members and member functions();
};
class D: public B, public C   // multiple inheritance
{
   // data members and member functions();
};
```

By – Mohammad Imran

Hybrid Inheritance

Example - 9

```
#include<iostream>
using namespace std;
class World
{
  public:
   World()
   {
      cout << "This is
      World!\n";
   }
};
```

```
class Continent: public World
{
  public:
   Continent()
   {
      cout << "This is
      Continent\n";
   }
};
```

By – Mohammad Imran

Hybrid Inheritance

Example - 9

```
class Country
{
  public:
   Country()
   {
      cout << "This is Country" << endl;
   }
};
class India: public Continent, public Country
{
  public:
   India()
   {
      cout << "This is India!";
   }
};
```

```
int main()
{
   India myworld;
   return 0;
}
```

OUTPUT

```
This is World
This is Continent
This is Country
This is India
```

By – Mohammad Imran

Objects and Memory

Introduction

In computer programming, memory allocation is a fundamental concept. Its play a vital role in optimizing and managing the memory usage of computer. In object-oriented programming (OOP), memory allocation has a more significant role as objects, as these are the basic building blocks of OOP languages such as Java, C++, Python, and C#. In this article, we are going to learn about what memory allocation for objects means, how it works, and its implications in computer programming.

By – Mohammad Imran

Memory Allocation for Objects

In object-oriented programming (OOP), the memory allocation for an object means it is the process to reserve the memory block in the memory of the computer for storing an object during the runtime. In object-oriented programming (OOP), objects are also known as instances. These instances represent real-world entities such as a person, a car, or a bank account. When an object is created, the memory needs to be allocated to store its data members (variables) and member functions (methods) so that these objects can be accessed and manipulated during program execution.

By – Mohammad Imran

Memory Allocation for Objects

- ✓ Objects are no different from simple data types.
- ✓ For example, following code where we are going to use an array of objects to clear your concept:

The memory space is allocated to the data members of a class only when an object of the class is declared, and not when the data members are declared inside the class. Since a single data member can have different values for different objects at the same time, every object declared for the class has an individual copy of all the data members.