18ES611 Embedded System Programming

Sarath tv

The infamous **Hello world** program

When learning a new language, the first program people usually write is one that salutes the world :)

Here is the Hello world program in C++.

```
#include <iostream.h>
int main() {
 cout << "Hello world!";
return 0;
}
```

Basics of C++

Input output

Variables

Loops

Variable declaration

`type variable-name;`

Meaning: variable <variable-name> will be a variable of type <type>

Where type can be:

```
int          //integer
double             //real number
char               //character
```

Example:

```
int a, b, c;
double x;
int sum;
char my-character;
```

Output statements

```
cout << variable-name;
```

Meaning: print the value of variable <variable-name> to the user

```
cout << "any message ";
```

Meaning: print the message within quotes to the user

```
cout << endl;
```

Meaning: print a new line

Example:

```
cout << a;
cout << b << c;
cout << "This is my character: " << my-character << " he he he"
     << endl;
```

Input statements

```
cin >> variable-name;
```

Meaning: read the value of the variable called <variable-name> from the user

Example:

```
cin >> a;
cin >> b >> c;
cin >> x;
cin >> my-character;
```

Practice

Control statements

Loops

Classes

Objects

Methods- different ways of defining functions

Members

Access specifiers
- For members
- For inheritance

Constructors

Features

OOP

Stands for "Object-Oriented Programming." OOP (not Oops!) refers to a **programming methodology based on objects**, instead of just functions and procedures.

These **objects are organized into classes**, which allow individual objects to be group together.

The focus of OOP languages is not on structure, **but on *modeling data*.**

Programmers code using "**blueprints**" of data models called *classes*.

Examples of OOP languages include C++, Visual Basic.NET and Java.

Class

A Class is **a user defined data-type which has Data Members and Member Functions**.

Data members are the **Data Variables** and **Member Functions are the functions used to manipulate these variables** and together these **data members and member functions defines the properties and behavior of the objects in a Class**.

In object-oriented programming , a class is a template definition of the methods and variables in a particular kind of object . Thus**, an object is a specific instance of a class; it contains real values instead of variables.**

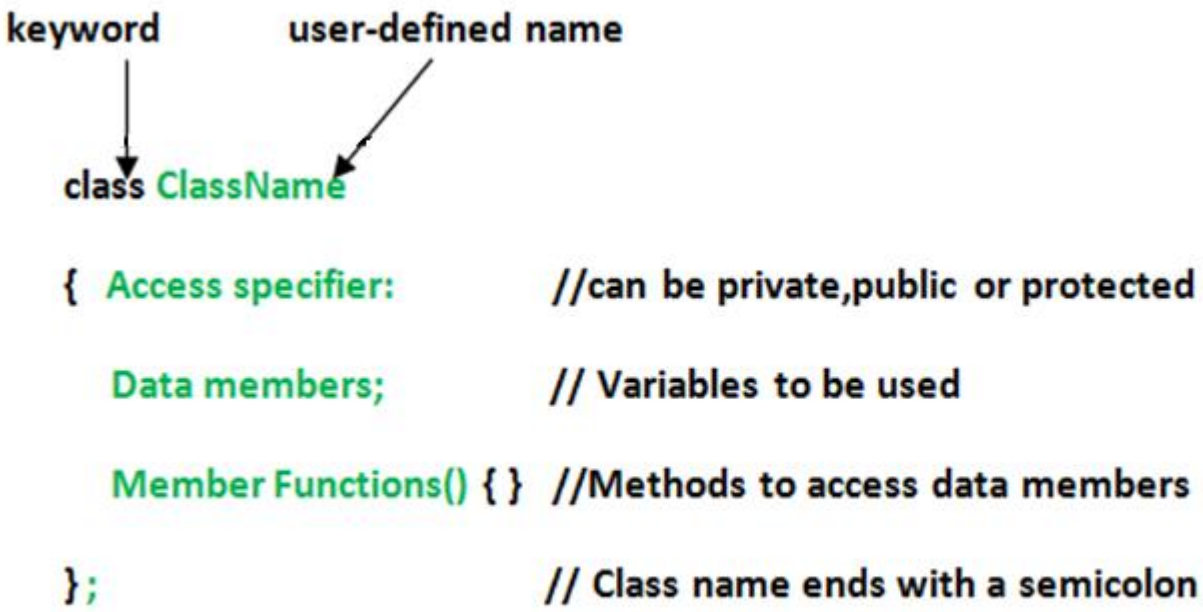
A class is like a blueprint for an object.

For Example: Consider the Class of **Cars**. There may be many cars with different names and brand but all of them will **share some common properties** like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and *wheels, speed limits, mileage are their properties.*

In C++, the concept of structure has been generalized in an object-oriented sense:

- Classes are types representing groups of similar instances
- Each instance has certain fields that define it (instance variables)
- Instances also have functions that can be applied to them (represented as function fields) -- called methods
- The programmer can limit access to parts of the class (to only those functions that need to know about the internals)

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end



Access Specifiers

Public - keyword makes data and functions public. Public data and functions can be accessed out of the class.

Private - keyword makes data and functions private. Private data and functions can be accessed only from inside the same class.

Protected- We will come to this later…… !!!!!!!!!!!!!!!!!!!!!!!!!

Mix of different access specifiers for different functions and members

Object

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated

Declaring Objects: When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:

```
ClassName ObjectName;
```

Access data member and member function in C++

You can access the data members and member functions by using a . (dot) operator.

It is important to note that, the private members can be accessed only from inside the class.

Accessing data members and member functions: The data members and member functions of class can be accessed using the **dot('.')** operator with the object. For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .

Accessing Data Members

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member. This access control is given by Access modifiers in C++. There are three access modifiers : **public, private and protected**.

Create a class

Use multiple access specifiers

Create data member

Create member functions

Create object

Access data members .

Create methods for class

Example

```
class Test{
    private:
        int data1;
        float data2;
    public:
        void function1()
        {   data1 = 2;  }
        float function2()
        {
            data2 = 3.5;
            return data2;
        }
};

int main() {
  Test o1, o2;
}
```

```
// C++ program to demonstrate
// accessing of data members

using namespace std;
class ESP
{
	// Access specifier
	public:

	// Data Members
	string name;

	// Member Functions()
	void printname()
	{
	cout << "name is: " << name;
	}
};

int main() {

	// Declare an object of class
	 ESP obj1;

	// accessing data member
	obj1.name = “16ES601-ESP";

	// accessing member function
	obj1.printname();
	return 0;
}
```

```
// C++ program to demonstrate
// accessing of data members

#include <bits/stdc++.h>
using namespace std;
class ESP
{
    // Access specifier
    public:

    // Data Members
    string name;

    // Member Functions()
    void printname()
    {
       cout <<  My name is: " << name;
    }
};

int main() {

    // Declare an object of class ESP
    ESP obj1;

    // accessing data member
    obj1.name = "Abhi";

    // accessing member function
    obj1.printname();
    return 0;
}
```

Defining member functions

```
// C++ program to demonstrate function
// declaration outside class

#include <bits/stdc++.h>
using namespace std;
class ESP
{
    public:
    string name;
    int id;

void printname();  // not defined

    // printid is defined inside class defination
    void printid()
    {
        cout << "Your  id is: " << id;
    }
};

// Definition of printname using scope resolution operator
 ::
void ESP::printname()
{
    cout << "name is: " << name;
}
int main() {

    ESP obj1;
    obj1.name = "xyz";
    obj1.id=15;

    // call printname()
    obj1.printname();
    cout << endl;

    // call printid()
    obj1.printid();
    return 0;
```

Constructors

Initialize object with default values

Compiler identifies a member function as constructor by it name and return type.

Name same as Class . And no return type. And always public

Constructor () : var1(var1_default), var2(var2_default){

}

Constructor overloading .. Same name different number of arguments

Default copy constructor

Object initialization Using Constructors

Constructor

A constructor is a special type of member function that initialises an object automatically when it is created.

Compiler identifies a given member function is a constructor **by its name and the return type**.

Constructor has the **same name as that of the class and it does not have any return type**. Also, the constructor is always **public**.

if you want to **execute some code immediately** after an object is created, you **can place the code inside the body of the constructor.**

```
class temporary
{
private:
      int x;
      float y;
public:
      // Constructor
      temporary(): x(5), y(5.5)
      {
            // Body of constructor
      }
      ... ..  ...
};

int main()
{
      Temporary t1;
      ... .. ...
}
```

How constructor works?

In the above pseudo code, **temporary()** is a constructor.

When an object of class temporary is created, the constructor is called automatically, and x is initialized to 5 and y is initialized to 5.5.

You can also initialize the data members inside the constructor's body as below. However, this method is not preferred.

```
temporary()
{
    x = 5;
    y = 5.5;
}
// This method is not preferred.
```

Features of OOP

Abstraction

Encapsulation

Inheritance

Polymorphism

OOP - Encapsulation

Incorporation into a class of data & operations in one package

Data can only be accessed through that package

“Information Hiding”

Encapsulation is defined as wrapping up of data and information under a single unit.

Encapsulation is a process of combining data and function into a single unit like capsule. This is to avoid the access of private data members from outside the class.

To achieve encapsulation, we make all data members of class private and create public functions, using them we can get the values from these data members or set the value to these data members.

```
// c++ program to explain
// Encapsulation

#include<iostream>
using namespace std;

class Encapsulation
{
    private:
        // data hidden from outside world
        int x;

    public:
        // function to set value of
        // variable x
        void set(int a)
        {
            x =a;
        }
        // function to return value of
        // variable x
        int get()
        {
            return x;
        }
};

// main function
int main()
{
    Encapsulation obj;

    obj.set(5);

    cout<<obj.get();
    return 0;
}
```

In the above program the variable **x** is made private. This variable can be accessed and manipulated only using the functions get() and set() which are present inside the class. Thus we can say that here, the variable x and the functions get() and set() are binded together which is nothing but encapsulation.

Role of access specifiers in encapsulation

The process of implementing encapsulation can be sub-divided into two steps:

The data members should be labeled as private using the **private** access specifiers

The member function which manipulates the data members should be labeled as public using the **public** access specifier

OOP - Polymorphism

Creating methods which describe the way to do some general function (Example: The “drive” method in the automobile class)

Polymorphic methods can adapt to specific types of objects.

Polymorphism is a feature using which an object behaves differently in different situation.

In function overloading we can have more than one function with same name but different numbers, type or sequence of arguments.

Polymorphism in C++

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. Real life example of polymorphism, a person at a same time can have different characteristic. Like a man at a same time is a father, a husband, a employee. So a same person posses have different behavior in different situations. This is called polymorphism.

Polymorphism is considered as one of the important features of Object Oriented Programming.

```
#include <bits/stdc++.h>

using namespace std;
class ESP
{
	public:

	// function with 1 int
parameter
	void func(int x)
	{
		cout << "value of x
is " << x << endl;
	}

	// function with same name
but 1 double parameter
	void func(double x)
	{
		cout << "value of x
is " << x << endl;
	}

	// function with same name
and 2 int parameters
	void func(int x, int y)
	{
		cout << "value of x
and y is " << x << ", " << y <<
endl;
	}
};

int main() {

	ESP obj1;

	// Which function is called
will depend on the parameters
passed
	// The first 'func' is
called
	obj1.func(7);

	// The second 'func' is
called
	obj1.func(9.132);

	// The third 'func' is
called
	obj1.func(85,64);
	return 0;
}
```

Abstraction in C++

Data abstraction is one of the most essential and important feature of object oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Consider a real life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

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

class implementAbstraction
{
	private:
		int a, b;

	public:

		// method to set values of
		// private members
		void set(int x, int y)
		{
			a = x;
			b = y;
		}

		void display()
		{
		cout<<"a = " <<a << endl;
cout<<"b = " << b << endl;
		}
};
```

```
int main()
{
	implementAbstraction obj;
	obj.set(10, 20);
	obj.display();
	return 0;
}
```

OOP - Inheritance

Allows programmers to create new classes based on an existing class

Methods and attributes from the parent class are inherited by the newly-created class

New methods and attributes can be created in the new class, but don’t affect the parent class’s definition

Inheritance in C++

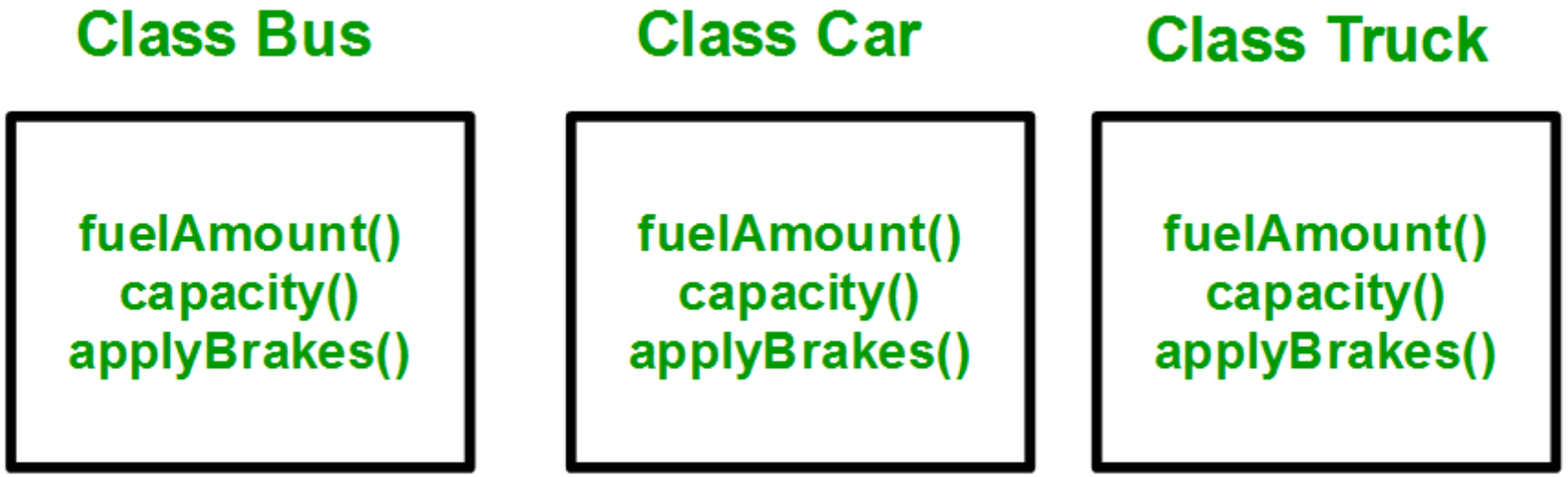
The capability of a class to derive properties and characteristics from another class is called **Inheritance**. Inheritance is one of the most important feature of Object Oriented Programming.

Sub Class: The class that inherits properties from another class is called Sub class or Derived Class.

Super Class: The class whose properties are inherited by sub class is called Base Class or Super class.

Why and when !!!!

Consider a group of vehicles. You need to create classes for Bus, Car and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be same for all of the three classes. If we create these classes avoiding inheritance then we have to write all of these functions in each of the three classes as shown in below figure:

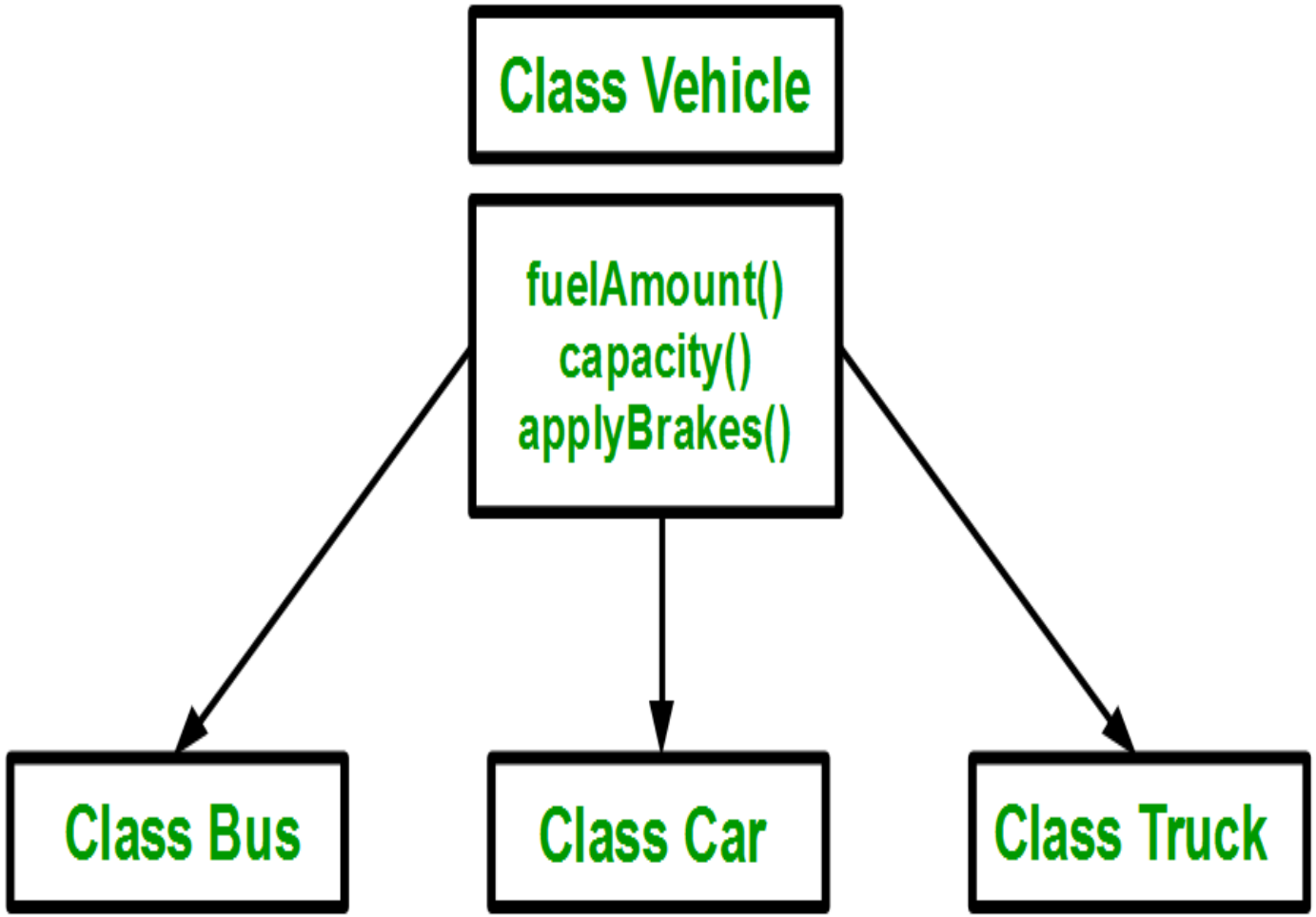


You can clearly see that above process results in duplication of same code 3 times.

This increases the chances of error and data redundancy.

To avoid this type of situation, inheritance is used.

If we create a class Vehicle and write these three functions in it and inherit the rest of the classes from the vehicle class, then we can simply avoid the duplication of data and increase re-usability.



Implementing inheritance in C++:

For creating a sub-class which is inherited from the base class we have to follow the below syntax.

Syntax:

```
class subclass_name : access_mode
base_class_name
{
  //body of subclass
};
```

subclass_name is the name of the sub class,
access_mode is the mode in which you want to inherit this sub class for example: public, private etc. and
base_class_name is the name of the base class from which you want to inherit the sub class.

```
// C++ program to demonstrate implementation
// of Inheritance

#include <bits/stdc++.h>
using namespace std;

//Base class
class Parent
{
    public:
      int id_p;
};

// Sub class inheriting from Base Class(Parent)
class Child : public Parent
{
    public:
      int id_c;
};

//main function
int main()
   {

        Child obj1;

        // An object of class child has all data members
        // and member functions of class parent
        obj1.id_c = 7;
        obj1.id_p = 91;
        cout << "Child id is " <<  obj1.id_c << endl;
        cout << "Parent id is " <<  obj1.id_p << endl;

        return 0;
   }
```

Modes of Inheritance

Public mode: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.

Protected mode: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.

Private mode: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.

Single Inheritance and Multiple Inheritance

Inheritance and Access Specifiers

Inheritance in C++ can be one of the following types:

- Private Inheritance
- Public Inheritance
- Protected inheritance

Member Access Rules

First and most important rule Private members of a class are never accessible from anywhere except the members of the same class.

Public Inheritance

All Public members of the Base Class become Public Members of the derived class &

All Protected members of the Base Class become Protected Members of the Derived Class.

i.e. No change in the Access of the members

Private Inheritance

All Public members of the Base Class become Private Members of the Derived class &

All Protected members of the Base Class become Private Members of the Derived Class.

Protected Inheritance

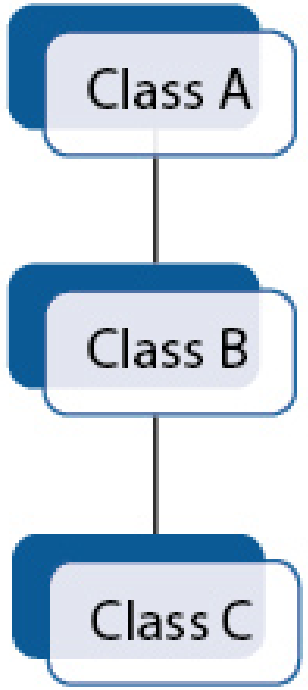
All Public members of the Base Class become Protected Members of the derived class &

All Protected members of the Base Class become Protected Members of the Derived Class.

```
// C++ Implementation to show
that a derived class
// doesn’t inherit access to
private data members.
// However, it does inherit a
full parent  object
class A
{
public:
      int x;
protected:
      int y;
private:
      int z;
};

class B : public A
{
      // x is public
      // y is protected
      // z is not accessible from
B
};

  Class C : protected A
{
      // x is protected
      // y is protected
      // z is not accessible from
C
};

class D : private A // 'private'
is default for classes
{
      // x is private
      // y is private
      // z is not accessible from
D
};
```

```
class A
{
... .. ...
};
class B: public A
{
... .. ...
};
class C: public B
{
... ... ...
};
```

You can derive a class from the base class but you can also derive a class from the derived class. This form of inheritance is known as **Multilevel inheritance**

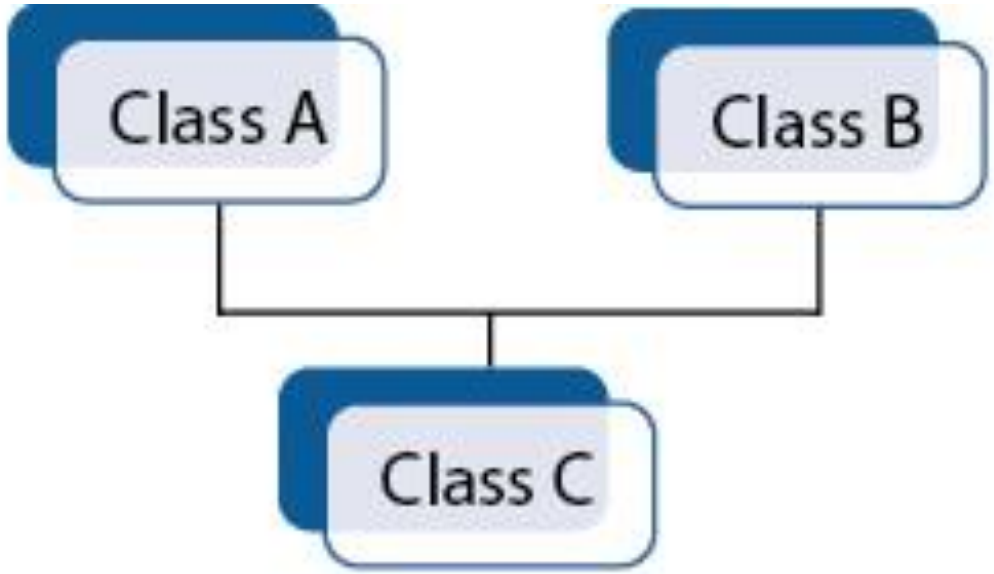


Multilevel Inheritance

C++ Multiple Inheritance

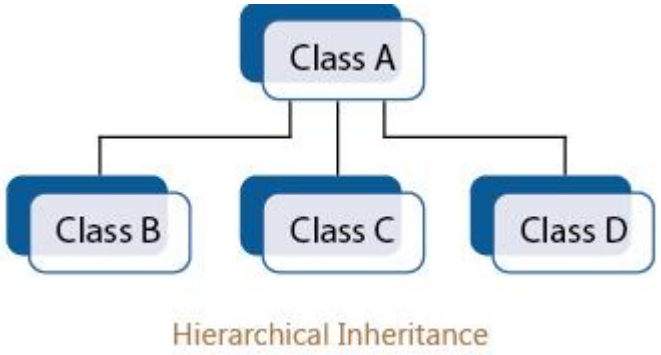
- A class can be derived from more than one Base class

```
class C: public A, public B {
};
```



Multiple Inheritance

If more than one class is inherited from the base class, it's known as **Hierarchical inheritance.**

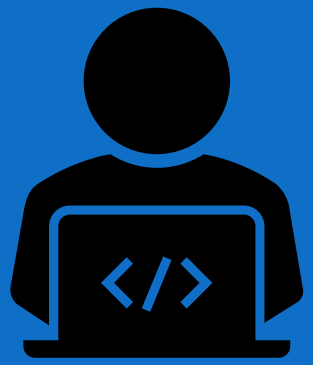


Hierarchical Inheritance

```
class base_class {

      ... .. ...

}


class first_derived_class: public base_class
{

      ... .. ...

}


class second_derived_class: public base_class
{

      ... .. ...

}


class third_derived_class: public base_class
{

      ... .. ...

}
```

Write C++ programs to show

- Single inheritance
- Multiple inheritance
- Multilevel inheritance
- Hierarchical inheritance



THANK YOU!!!!!