UNIT III

OBJECT-ORIENTED PROGRAMMING CONCEPTS

UNITIII

OBJECT-ORIENTED PROGRAMMING CONCEPTS

- **Topics to be discussed,**

➢ Inheritance

➢ Constructors and Destructors in Derived Classes

➢ Polymorphism and Virtual Functions

UNITIII

OBJECT-ORIENTED PROGRAMMING CONCEPTS

- **Topics to be discussed,**

- **Inheritance**
- Constructors and Destructors in Derived Classes
- Polymorphism and Virtual Functions

Inheritance

- Inheritance is **one of the most important feature of Object Oriented Programming**.
- The capability of a class to derive properties and characteristics from another class is called **Inheritance**.
- It allows user to create a **new class (derived class)** from an **existing class(base class)**.
- Inheritance makes the code reusable.
- When we inherit an existing class, all its methods and fields become available in the new class, hence code is reused.
- The idea of inheritance implements the **is a relationship**.
- For example, mammal IS-A animal, dog **IS-A** mammal hence dog IS-A animal as well and so on.

Inheritance – Cont'd

- **Sub Class:** The class which inherits properties of other class is called Child or Derived or Sub class. The derived class is the specialized class for the base class.
- **Super Class:** The class whose properties are inherited by other class is called the Parent or Base or Super class.
- **NOTE:- All members of a class except Private, are inherited.**
- A class can be derived from **more than one classes**, which means it can inherit data and functions from multiple base classes.

Inheritance – Cont'd

- **Syntax of Inheritance**

  class Derivedclass_name : access-specifier Baseclass_name

  {

  // body of subclass

  };

  - **Derivedclass_name** is the name of the derived class
  - **access-specifier** is the mode in which we want to inherit this sub class, i.e **public**, **protected**, or **private**.
  - **Base_class_name** is the name of the base class from which we want to inherit the sub class.
- If the access-specifier is not used, then it is **private** by default.
- **Note**: A derived class doesn't inherit ***access*** to private data members. However, it does inherit a full parent object, which contains any private members which that class declares.

Inheritance – Cont'd

- **Modes of Inheritance:**
- There are 3 modes of inheritance.
  - **Public Mode**: If we derive a subclass 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 the derived class.
  - **Protected Mode**: If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.
  - **Private Mode**: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.
- **Note:** *The private members in the base class cannot be directly accessed in the derived class, while protected members can be directly accessed.*

Inheritance – Cont'd

- **Example:**

```
1. class Derived : private Base        //private derivation
        {           }
2. class Derived : public Base         //public derivation
        {           }
3. class Derived : protected Base  //protected derivation
        {           }
4. class Derived : Base          //private derivation by default
        {         }
```

Note:

- When a base class is privately inherited by the derived class, public members of the base class becomes the private members of the derived class and therefore, the public members of the base class can only be accessed by the member functions of the derived class. They are inaccessible to the objects of the derived class.
- On the other hand, when the base class is publicly inherited by the derived class, public members of the base class also become the public members of the derived class. Therefore, the public members of the base class are accessible by the objects of the derived class as well as by the member functions 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
};
```

public derivation Example

```
#include <iostream>
using namespace std;
class Base
{
    int a;//Not Inheritable
    public:
        int b;//Inheritable
    void setAB(int x,int y)
    {
        a=x;b=y;
    }
    int getA()
    {
        return a;
    }
    void showA()
    {
        cout << "\na:"<<a<<endl;
    }
};
```

```
class Derived : public Base
{
    int c;

public:
    void mul()
    {
        c=b*getA();
    }

    void show()
    {

        cout <<"b: "<< b;
        cout << "\nc: "<< c;
    }
};
int main()
{
    Derived d;
    d.setAB(2,5);
    d.showA();
    d.mul();
    d.show();

    d.b=100;
    d.mul();
    d.showA();
    d.show();
    return 0;
}
```

Output:

```
a:2
b: 5
c: 10
a:2
b: 100
c: 200
```

private derivation Example

```
#include <iostream>
using namespace std;
class Base
{
    int a;//Not Inheritable
    public:
        int b;//Inheritable
    void setAB(int x,int y)
    {
        a=x;b=y;
    }
    int getA()
    {
        return a;
    }
    void showA()
    {
        cout << "\na:"<<a<<endl;
    }
};
class Derived : private Base
{
    int c;

public:
    void mul()
    {
        c=b*getA();
    }

    void show()
    {
        cout <<"\nb: "<< b;
        cout << "\nc: " << c;
    }
};
```

```
int main()
{
    Derived d;
    d.setAB(2,5);
    d.mul();
    d.showA();
    d.show();

    d.b=100;
    d.mul();
    d.show();
    return 0;
}
```

Output:

File	Line	Message
		=== Build file: "no target" in "no project" (compiler: unknown) ===
F:\2024\CS320...		In function 'int main()':
F:\2024\CS320...	41	error: 'void Base::setAB(int, int)' is inaccessible within this context
F:\2024\CS320...	8	note: declared here
F:\2024\CS320...	41	error: 'Base' is not an accessible base of 'Derived'
F:\2024\CS320...	43	error: 'void Base::showA()' is inaccessible within this context
F:\2024\CS320...	16	note: declared here
F:\2024\CS320...	43	error: 'Base' is not an accessible base of 'Derived'
F:\2024\CS320...	46	error: 'int Base::b' is inaccessible within this context
F:\2024\CS320...	7	note: declared here
		=== Build failed: 5 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

Inheritance – Cont'd

- **Access Control and Inheritance:**

  class derived-class: access-specifier base-classA

  – derived class can access all the non-private members of its base class.

  – Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

- **A derived class can inherit all base class methods except**:
  - Constructors, destructors and copy constructors of the base class.
  - Overloaded operators of the base class.
  - The friend functions of the base class.

- Create a class **shape** with ***width*** and ***height*** as its data members and setWidth() and setHeight() as member functions which assigns the arguments received to width and height data members respectively. Create a class called Rectangle which inherits from Shape and has a member function getArea() which returns the area by finding the product of width and height. Create an object of Rectangle. Assign values to width and height and calculate area.

Inheritance - Example

```
#include <iostream>
using namespace std;
class Shape // Base class
{
protected:
  int width;
  int height;
public:
  void setWidth(int w)
  {
    width = w;
  }
  void setHeight(int h)
  {
    height = h;
  }
};
class Rectangle: public Shape// Derived class
 {
  public:
  int getArea()
  {
    return (width * height);
  }
};
```

```
int main()
{
  Rectangle Rect;
  Rect.setWidth(5);
  Rect.setHeight(7);
  cout << "Total area: " << Rect.getArea() << endl;
  return 0;
}
```

Output:

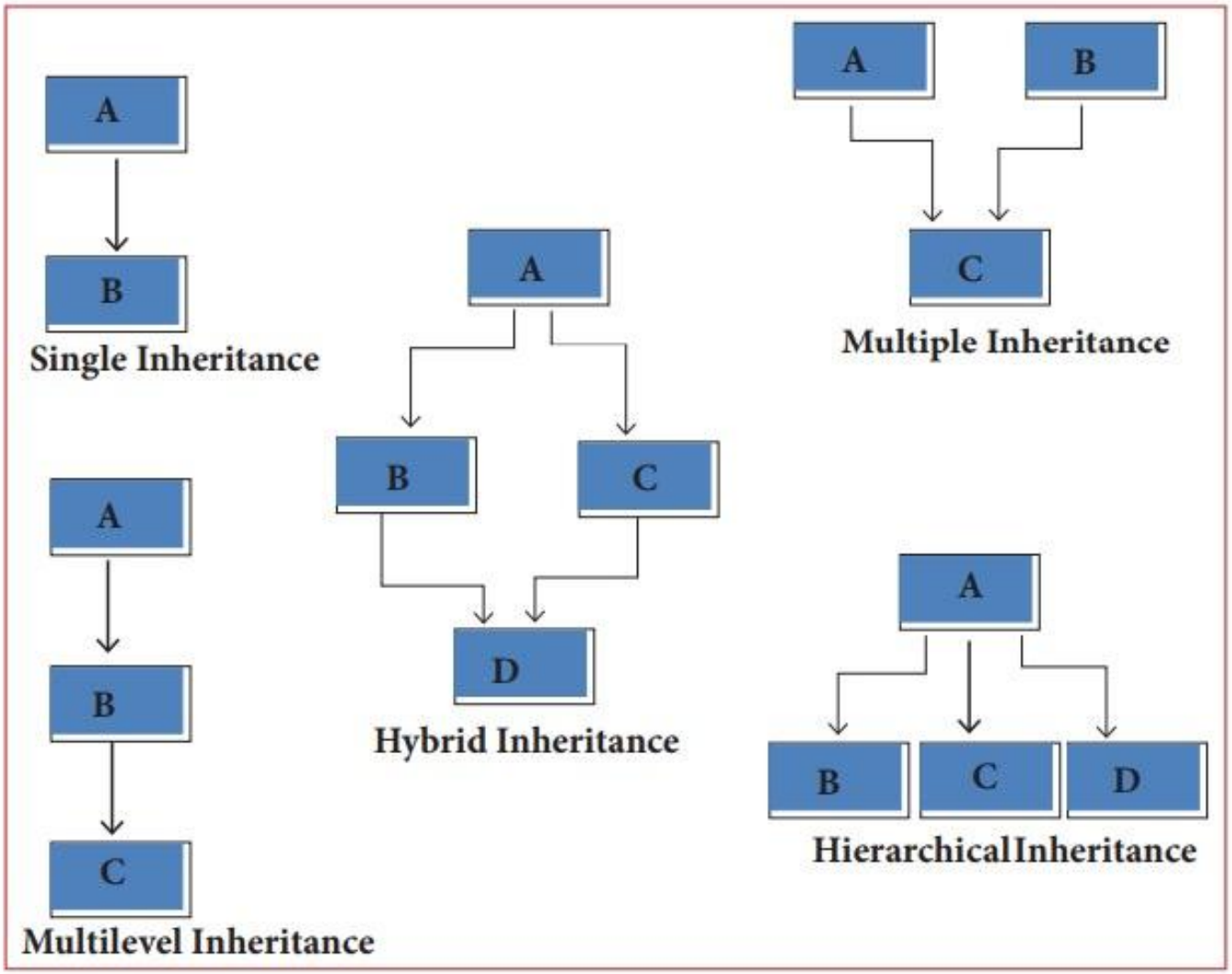
```
Total area: 35
```

- Create a class StudentInfo with name, age and gender and its data members, getInfo() and putInfo() as member functions to get data and display data respectively. Create a derived class studentResult with total (for 5 subjects), percentage and grade as data members and getMarks(), calcGrade() and displayResult() as member functions and inherits from StudentInfo. Get all the details of a student and print the same.

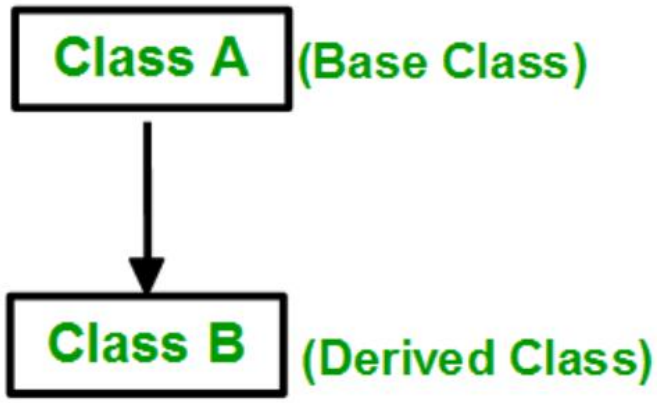
Inheritance – Cont'd

- **Types of Inheritance:**
  - **Single Inheritance** – In this type of inheritance one derived class inherits from only one base class. It is the most simplest form of Inheritance.
  - **Multiple Inheritance** – In this type of inheritance a single derived class may inherit from two or more than two base classes.
  - **Hierarchical Inheritance** – In this type of inheritance, multiple derived classes inherits from a single base class.
  - **Multilevel Inheritance** – In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is sub class for the other.
  - **Hybrid Inheritance/ Multipath (also known as Virtual Inheritance)** – a sub class follows multiple types of inheritance while deriving properties from the base or super class

Types of Inheritance



Inheritance – Cont'd
(Single Inheritance in C++)




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

- In this type of inheritance one derived class inherits from only one base class.
- It is the most simplest form of Inheritance.
- All other types of Inheritance are a combination or derivation of Single inheritance.

Single Inheritance - Example

```
#include<iostream>
using namespace std;
class father
{
   public:
    void house()
    {
        cout<<"Have 2BHK House."<<endl;
    }
};
class son:public father
{
    public:
    void car()
    {
        cout<<"Have Audi Car."<<endl;
    }
};
int main()
{
    son o;
    o.house();
    o.car();
    return 0;
}
```

Output:

```
Have 2BHK House.
Have Audi Car.
```

Inheritance – Cont'd (Single Inheritance in C++)

- **Ambiguity in Single Inheritance in C++**
  - If parent and child classes have same named method, parent name and scope resolution operator (::) is used.
  - This is done to distinguish the method of child and parent class since both have same name.

parent and child classes have same named method

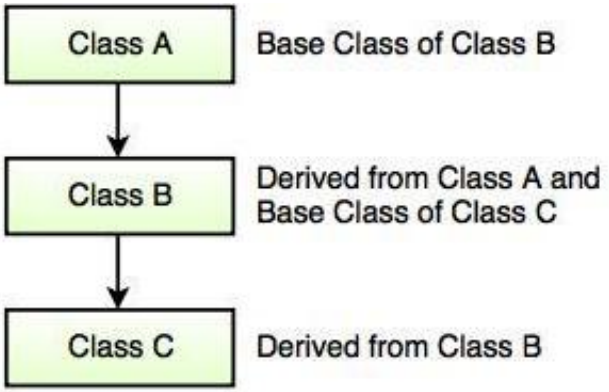
```
#include <iostream>
using namespace std;
class staff
{
    protected:
        string name;
        int code;
    public:
        void getdata();
};
class typist: public staff
{
    private:
        int speed;
    public:
        void getdata();
        void display();
};
void staff::getdata()
{

    cout<<"Name:";
    cin>>name;
    cout<<"Code:";
    cin>>code;
}
```

```
void typist::getdata()
{
    cout<<"Speed:";
    cin>>speed;
}
void typist::display()
{
    cout<<"Name:"<<name<<endl;
    cout<<"Code:"<<code<<endl;
    cout<<"Speed:"<<speed<<endl;
}

int main()
{
    typist t;
    cout<<"Enter data"<<endl;
    t.staff::getdata();
    t.getdata();
    cout<<endl<<"Display data"<<endl;
    t.display();
    return 0;
}
```

Output:

```
Enter data
Name:Ajay
Code:112
Speed:123

Display data
Name:Ajay
Code:112
Speed:123
```

Inheritance – Cont'd
(Multilevel Inheritance in C++)




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

- In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class.
- The Super class for one, is sub class for the other.
- When a class is derived from a class which is also derived from another class, such inheritance is called Multilevel Inheritance.
- **The level of inheritance can be extended to any number of level depending upon the relation.**
- Multilevel inheritance is similar to relation between grandfather, father and child.

Multilevel Inheritance Example

```
#include <iostream>
using namespace std;
class A //Base Class : class A
{
    private:
        int a;
    public:
        void set_a(int val_a)
        {
            a=val_a;
        }
        void disp_a(void)
        {
            cout << "Value of a: " << a << endl;
        }
};
//Here Class B is base class for class C
//and Derived class for class A
class B: public A
{
    private:
        int b;
    public:
        //assign value of a from here
        void set_b(int val_a, int val_b)
        {
            //assign value of a by calling function of class A
            set_a(val_a);
            b=val_b;
        }
        void disp_b(void)
        {
            //display value of a
            disp_a();
            cout << "Value of b: " << b << endl;
        }
};
```

```
//Here class C is derived class and B is Base class
class C: public B
{
    private:
        int c;
    public:
        //assign value of a from here
        void set_c(int val_a, int val_b,int val_c)
        {
            /*** Multilevel Inheritance ***/
            //assign value of a, b by calling function of class B and Class A
            //here Class A is inherited on Class B, and Class B in inherited on Class B
            set_b(val_a,val_b);
            c=val_c;
        }
        void disp_c(void)
        {
            //display value of a and b using disp_b()
            disp_b();
            cout << "Value of c: " << c << endl;
        }
};
int main()
{
    //create object of final class, which is Class C
    C objC;

    objC.set_c(10,20,30);
    objC.disp_c();

    return 0;
}
```

Output:

```
Value of a: 10
Value of b: 20
Value of c: 30
```

Multilevel Inheritance Example

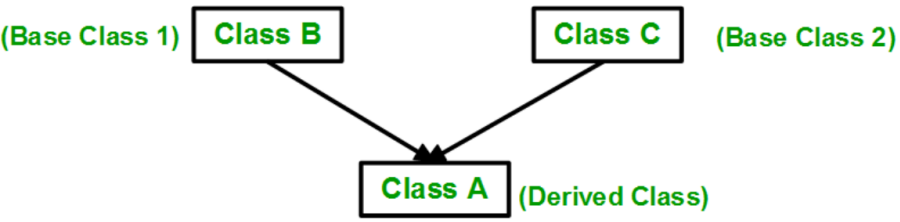
```
#include<iostream>
using namespace std;
class AddData  //Base Class
{
    protected:
        int subjects[3] ;
    public:
        void accept_details()
        {
                cout<<"\n Enter Marks for Three Subjects ";
                cout<<"\n ------------------------------ \n";
                cout<<"\n English : ";
                cin>>subjects[0];
                cout<<"\n Maths : ";
                cin>>subjects[1];
                cout<<"\n History : ";
                cin>>subjects[2];
        }
};
//Class Total - Derived Class.
//Derived from class AddData and Base class of class Percentage
class Total : public AddData
{
    protected:
        int total;
    public:
        void total_of_three_subjects()
        {
                total = subjects[0] + subjects[1] + subjects[2];
        }
};
```

```
//Class Percentage - Derived Class.
// Derived from class Total
class Percentage : public Total
{
    private:
        float per;
    public:
        void calculate_percentage()
        {
                per=total/3.0;
        }
        void show_result()
        {
                cout<<"\n ------------------------------ \n";
                cout<<"\n Percentage of a Student : "<<per;
        }
};
int main()
{
        Percentage p;
        p.accept_details();
        p.total_of_three_subjects();
        p.calculate_percentage();
        p.show_result();
        return 0;
}
```

Output:

```
Enter Marks for Three Subjects
------------------------------

English : 98

Maths : 99

History : 98

------------------------------

Percentage of a Student : 98.3333
```

Inheritance – Cont'd
(Multiple Inheritance in C++)




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

- In C++ programming, a class can be derived from **more than one parents**.
- When a class is derived from **two or more** base classes, such inheritance is called **Multiple Inheritance**.
- Multiple Inheritance in C++ allow us to combine the features of several existing classes into a single class.
- **Syntax:**

class subclass_name : access_mode base_class1, access_mode base_class2, ...
{
  // body of subclass
};

Multiple Inheritance Example

```
#include<iostream>
using namespace std;
class student
{
    protected:
        int rno, m1, m2;
    public:
        void getData()
        {
            cout << "Enter the Roll no :";
            cin>>rno;
            cout << "Enter the two marks    :";
            cin >> m1>>m2;
        }
};
class sports
{
    protected:
        int sm; // sm = Sports mark
    public:
        void getsm()
        {
            cout << "\nEnter the sports mark :";
            cin>>sm;
        }
};
class result : public student, public sports
{
    int tot, avg;
    public:
        void display()
        {
            tot = (m1 + m2 + sm);
            avg = tot / 3;
            cout << "\n\nRoll No: " << rno;
            cout << "\nTotal: " << tot;
            cout << "\nAverage: " << avg;
        }
};
int main()
{
    result obj;
    obj.getData();
    obj.getsm();
    obj.display();
}
```

Output:

```
Enter the Roll no :111
Enter the two marks    :89 98

Enter the sports mark :80


Roll No: 111
Total: 267
Average: 89
```

Ambiguity in Multiple Inheritance

```
class base1
{
  public:
     void someFunction( )
     { .... ... .... }
};
class base2
{
    void someFunction( )
     { .... ... .... }
};
class derived : public base1, public base2
{

};
int main()
{
    derived obj;
    obj.someFunction()// Error!
}
```

This problem can be solved using scope resolution(::) function to specify which function to class either base1 or base2

- In multiple inheritance, a single class is derived from two or more parent classes.
- So, there may be a possibility that two or more parents have **same named member function**.
- If the object of child class needs to access one of the same named member function then it results in **ambiguity**.
- The **compiler is confused** as method of which class to call on executing the call statement.

```
int main()
{
   obj.base1::someFunction( );  // Function of base1 class is called
   obj.base2::someFunction();   // Function of base2 class is called.
}
```

Multiple Inheritance Example

```
#include<iostream>
using namespace std;
class A
{
    protected:
    int x;
    void get()
    {
        cout<<"Enter value of x: ";
        cin >> x;
    }
};
class B
{
    protected:
    int y;
    void get()
    {
        cout<<"Enter value of y: ";
        cin >> y;
    }
};
```

```
//C is derived from class A and class B
class C : public A, public B
{
    public:
    void getData()
    {
        A::get();
        B::get();
    }
    void sum()
    {
        cout << "Sum = " << x + y;
    }
};

int main()
{
     C obj1; //object of derived class C
     obj1.getData();
     obj1.sum();
     return 0;
}
```

Output:

```
Enter value of x: 4
Enter value of y: 5
Sum = 9
```

- Create two classes named Mammals and MarineAnimals. Create another class named BlueWhale which inherits both the above classes. Now, create a function in each of these classes which prints "I am mammal", "I am a marine animal" and "I belong to both the categories: Mammals as well as Marine Animals" respectively. Now, create an object for each of the above class and try calling
  1 - function of Mammals by the object of Mammal
  2 - function of MarineAnimal by the object of MarineAnimal
  3 - function of BlueWhale by the object of BlueWhale
  4 - function of each of its parent by the object of BlueWhale