Chapter-10

INHERITANCE

➢ Introduction:
- Inheritance is another important aspect of object oriented programming.
- C++ allows the user to create a new class (derived class) from an existing class (base class).

➢ Inheritance:
- Inheritance is the capability of one class to inherit properties from another class.
- Base Class: It is the class whose properties are inherited by another class. It is also called Super class.
- Derived Class: It is the class that inherits the properties from base class. It is also called Sub class.

➢ Need of Inheritance:
- Suppose X is a class already defined and we need to redefine another class Y has same properties of X and in addition its own.
- Suppose if we use direct option without using inheritance, it has following problems.
  - Code written in X is repeated again in Y which leads to unnecessary wastage of memory.
  - Testing to be done separately for both class X and class Y leads to waste of time.
- The above problem can be solved by using the concept of inheritance.
- If we use the code of X even in Y without rewriting it. The class Y inherits all the properties of X.
- The class X is called base class and the class Y is called derived class.

➢ The main advantages of Inheritance are:
  - Reusing existing code
  - Faster development time
  - Easy to maintain
  - Easy to extend
  - Memory Utilization

➢ The main disadvantage of Inheritance are:
  - Inappropriate use of inheritance makes programs more complicated.
  - Calling member functions using objects creates more compiler overheads.
Defining Derived Classes:

- A derived class is a composite class – it inherits members from the base class and adds member of its own.

- The general form of the derived class is given below.

Here,

- **class** → Keyword
- **derived_class_name** → Name of the derived class
- **:** → Shows the derivation from the base class.
- **Visibility Mode** → Specifies the type of derivation
- **base_class_name** → Name of the base class.
- The use of a constructor can be cleverly done especially in those problems where it is necessary to initialize certain data members compulsorily.

Example:

<table>
<thead>
<tr>
<th>Public Derived Class</th>
<th>Private Derived Class</th>
<th>Protected Derived Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>class father //Base class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
<td></td>
</tr>
<tr>
<td>private:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>char name[10];</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>char caste[10];</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int age;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void readdata( );</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| };
| class son : public father |
| { |
| private: |
| char gender[5]; |
| public: |
| void display( ); |
| };
| class father //Base class |
| { |
| private: |
| char name[10]; |
| public: |
| char caste[10]; |
| int age; |
| void readdata( ); |
| };
| class son : private father |
| { |
| private: |
| char gender[5]; |
| public: |
| void display( ); |
| };
| class father //Base class |
| { |
| private: |
| char name[10]; |
| public: |
| char caste[10]; |
| int age; |
| void readdata( ); |
| };
| class son : protected father |
| { |
| private: |
| char gender[5]; |
| public: |
| void display( ); |
| };

Visibility mode:

- Visibility mode can be public, private or protected. The private data of base class cannot be inherited.

  - **public**: If inheritance is done in public mode, public members of the base class become the public member of derived class and protected member of base class become the protected member of derived class.
private: If inheritance is done in a private mode, public and protected members of base class become the private members of derived class.

protected: If inheritance is done in a protected mode, public and protected members of base class become the protected members of the base class.

<table>
<thead>
<tr>
<th>Visibility Mode</th>
<th>public</th>
<th>private</th>
<th>protected</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>public</td>
<td>private</td>
<td>protected</td>
</tr>
<tr>
<td>private</td>
<td>Not inherited</td>
<td>Not inherited</td>
<td>Not inherited</td>
</tr>
<tr>
<td>protected</td>
<td>protected</td>
<td>private</td>
<td>protected</td>
</tr>
</tbody>
</table>

### Public Inheritance:

- When a base class is inherited as public, all public members of the base class become public members of derived class.
- The private members of the base class remain private to that class, and are not accessible by members of the derived class.
- Example: A program illustrates public inheritance.

```cpp
#include<iostream.h>
#include<cstdio.h>

class shape //Base Class
{
    public:
        int side1, side2;
};
class rectangle : public shape //Derived Class
{
    public:
        int area;
        void compute()
        {
            area = side1 * side2;
        }
};
void main()
{
    rectangle R; // R is the object of derived class
    R.side1 = 5; // Data directly accessed by object
    R.side2 = 6;
    R.compute();
    cout<< "Area of the rectangle = " <<R.area;
}
```

**OUTPUT:**

Area of the rectangle = 30
Private Inheritance:

- When a base class is inherited as private, then all public and protected members of the base class become private members of derived class.
- This means that they are still accessible by member of the derived class, but cannot be accessed by other parts of the program.
- Example: A program illustrates private inheritance.

```cpp
#include<iostream.h>
#include<conio.h>

class shape //Base Class
{
    protected:
        int side1, side2;
    public:
        int area;
        void compute()
        {
            area = side1 * side2;
        }
};

class rectangle : private shape //Derived Class
{
    public:
        void readdata()
        {
            cout << " Enter the input first side :";
            cin>>side1;
            cout << " Enter the input second side :";
            cin>>side2;
        }
        void display()
        {
            compute(); // Calling base class
            cout<< "Area of the rectangle = " <<area;
        }
};

void main()
{
    rectangle R;
    R.readdata();
    R.display();
}
```

OUTPUT:
Enter the input first side : 9
Enter the input second side : 5
Area of the rectangle = 45
Protected Inheritance:

- When a base class is inherited as protected, then all public and protected members of the base class become protected members of derived class.
- The private data members of base class are not visible to derived class.
- They can only be accessed through public and protected member functions of base class.
- Example: A program illustrates protected inheritance with the base class having protected and public members.

```cpp
#include<iostream.h>
#include<conio.h>

class shape //Base Class
{
    protected:
        int side1, side2;
    public:
        int compute( )
        {
            return(side1 * side2);
        }
};
class rectangle : protected shape //Derived Class
{
    public:
        void readdata( )
        {
            cout << " Enter the input first side :";
            cin>>side1;
            cout << " Enter the input second side :";
            cin>>side2;
        }
        void display( )
        {
            cout<< "Area of the rectangle = " <<compute( );
        }
};
void main( )
{
    rectangle R;
    R.readdata( );
    R.display( );
}
```

OUTPUT:

Enter the input first side : 7
Enter the input second side : 8
Area of the rectangle = 56
Types of Inheritance:

- Based on the relationship, inheritance can be classified into five forms:
  - Single Inheritance
  - Multilevel Inheritance
  - Multiple Inheritance
  - Hierarchical Inheritance
  - Hybrid Inheritance

**Single Inheritance:**

- *Single Inheritance is the process of creating a new class from existing class base class.*
- It is very common in inheritance that a class is derived from the base class.
- The data members and member function of the base class are data member and member function of the derived class.

A derived class with single inheritance is declared as follows:

```cpp
class Base_Class
{
    ................
};
class Derived_class : public Base_class
{
    ................
};
```

**Example: Program to illustrate single level inheritance.**

```cpp
#include<iostream.h>
#include<conio.h>
class base
{
    private:
        int rollno;
        char name[10];
    public:
        void read( )
```
```cpp
{    cout << " Enter Roll Number and Name " <<endl;
cin >> rollno >> name;
}
void display( )
{
    cout << " Roll No : " << rollno <<endl;
cout << " Name : " << name <<endl;
}

};
class derived : public base
{
private:
    int m1, m2, t;
public:
    void read1( )
    {
        cout << " Enter Maths and Computer marks " <<endl;
cin >> m1 >> m2;
t = m1 + m2;
    }
    void display1( )
    {
        cout << " Maths : " << m1 <<endl;
cout << " Computer : " << m2 <<endl;
cout << "Total Marks : " << t <<endl;
    }
};
void main( )
{
    derived obj;
crscr( );
obj.read();
obj.read1();
obj.display();
obj.display1();
getch( );

```

**OUTPUT:**
Enter Roll Number and Name
1234 RAM
Enter Maths and Computer marks
80 90
Roll No : 1234
Name : RAM
Maths : 80
Computer : 90
Total Marks : 170

**Multilevel Inheritance:**

- **Derivation of a class from another derived class is called multilevel inheritance.**
- In the figure class A is the base class for class AB and class AB is the base class for class ABC.
- The class AB provides a link for the inheritance between A and ABC, and is known as
A derived class with multilevel inheritance is declared as follows:

```cpp
class A {
     public:
     void displayA() {
         cout << "Base class A" << endl;
     }
};
class AB : public A {
     public:
};
class ABC : public AB {
     public:
};
```

**Example: Program to illustrate multilevel inheritance.**

```cpp
#include<iostream.h>
#include<conio.h>
class A {
     public:
     void displayA() {
         cout << "Base class A" << endl;
     }
};
class AB : public A {
     public:
};
class ABC : public AB {
     public:
};
```
void displayAB( )
{
    cout << " Intermediate Base class AB" << endl;
    cout << " Derived from A" << endl;
}
};
class ABC : public AB
{
    public:
        void displayABC( )
        {
            cout << " Derived Class ABC" << endl;
            cout << " Derived from AB" << endl;
        }
    void output( )
    {
        displayA( );
        displayAB( );
        displayABC( );
    }
};
void main( )
{
    ABC obj;
    clrscr( );
    obj.output( );
    getch( );
}

**Multiple Inheritance:**

- A class can be derived from more than one base class is known as multiple inheritance.

```
A

B

C

MULTIPLE INHERITANCE

Base-1

Base-2

Derived
```

- A derived class with multiple inheritance is declared as follows:

class A  //Base Class A
{
    ........
};
class B //Base Class B
{
    
};

class C //Base Class C
{
    
};
class Derived_Class : public A, private B, protected C
{
    
    //Members of derived class
};

- **Hierarchical Inheritance:**
  - *If a number of classes are derived from a single base class, it is called as hierarchical inheritance.*
  - Hierarchical model exhibits top down approach by breaking up a complex class into simpler class.

- **Hybrid Inheritance:**
  - *Hybrid Inheritance is combination of Hierarchical and multilevel inheritance.*
Virtual Base Classes:
- When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited.
- Such a base class is known as *virtual base class*.
- This can be achieved by preceding the base class name with the word virtual.
- Example:
  
  ```cpp
  class A
  {
    ---------------------
    ---------------------
  }
  class B : virtual public A
  {
    ---------------------
    ---------------------
  }
  class C : virtual public A
  {
    ---------------------
    ---------------------
  }
  class D : public B, public C
  {
    ---------------------
    ---------------------
  }
  ```

Abstract Class:
- An abstract class is one that is not used to create objects.
- An abstract class is designed only to act as a base class (to be inherited by other classes).

Constructor and Destructors in derived classes:
- A *destructor* is a special member function that is executed when an object of that class is destroyed.
• Destroying an object means, de-allocating all the resources such as memory that was allocated for the object by the constructor.

• It will have like constructor, the name same as that of the class but preceded by a tilde (~).

• **Example: Program to illustrate the use of destructors in C++.**

```cpp
class Base
{
    public:
        Base( ); //Default constructor
        {
            cout<<"Inside Base Constructor"<<endl;
        }
        ~ Base( ); //Destructor
        {
            cout<<"Inside Base Destructor"<<endl;
        }
};

class Derived : public Base
{
    public:
        Derived( ); //Default constructor
        {
            cout<<"Inside Derived Constructor"<<endl;
        }
        ~ Derived( ); //Destructor
        {
            cout<<"Inside Derived Destructor"<<endl;
        }
};

void main( )
{
    Derived x;
    a.display( );
}
```

**OUTPUT:**
Inside Base Constructor
Inside Derived Constructor
Inside Derived Destructor
Inside Base Destructor

**Important Questions**
5 Marks Question:

1. What is Inheritance? Explain any two types of Inheritance.
2. What are visibility modes? Explain.
3. What is the difference between public, private and protected access specifiers?
4. Explain single inheritance with a suitable C++ program.
5. What is Multilevel Inheritance? Explain with a suitable program example.
6. What is virtual base class? Give example.
7. What are the advantages of Inheritance? (Any five)