Object Oriented Programming
Lecture 7

Inheritance

Inheritance

- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class (or parent class), and the new class is referred to as the **derived** class (or child class).
- IS-A relationship

Motivation and benefits of Inheritance

- Reusability: This also provides an opportunity to reuse the code functionality and fast implementation time.
- Maintainability: Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.

Base and Derived Classes

- A class can be derived from another class, which means it inherits data and functions from that base class. To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form

class derived-class: access-specifier base-class

Where access-specifier is one of **public, protected,** or **private**, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

```
// Base class
class Shape
{
protected:
int width;
int height;
public:
void setWidth(int w)
{       width = w;
}
void setHeight(int h)
{       height = h;
}
};
// Derived class
class Rectangle: public Shape
{   public:
int getArea()
{        return (width * height);
}
};
```

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

//Output:
//Total area: 35
```

Access Control and Inheritance

- A 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.
- A derived class inherits all base class methods with the following exceptions −
  - Constructors and destructors of the base class
  - Overloaded operators of the base class
  - The friend functions of the base class

Type of Inheritance

- When deriving a class from a base class, the base class may be inherited through **public, protected** or **private** inheritance. The type of inheritance is specified by the access-specifier.
- We hardly use **protected** or **private** inheritance, but **public** inheritance is commonly used. While using different type of inheritance, following rules are applied −
- **Public Inheritance** − When deriving a class from a **public** base class, **public** members of the base class become **public** members of the derived class and **protected** members of the base class become **protected** members of the derived class. A base class's **private** members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class.
- **Protected Inheritance** − When deriving from a **protected** base class, **public** and **protected** members of the base class become **protected** members of the derived class.
- **Private Inheritance** − When deriving from a **private** base class, **public** and **protected** members of the base class become **private** members of the derived class

Constructor and Destructor calling sequence

- In inheritance, the base class constructor is called before the child class constructor.
- The destructors are called in the reverse order of constructors' calls.

```
class Shape {
 private:
     int width;
     int height;
   public:
   Shape(){ cout<<"Shape constructor called:
\n"; }
     void setWidth(int w) {
        width = w;  }
     void setHeight(int h) {
        height = h; }
     int getWidth() {
        return width; }
     int getHeight() {
       return height; }
       ~Shape(){
   cout<<"Shape destructor called: \n“;}
};
```

```
class Rectangle: public Shape {
   public:
  Rectangle(){
   cout<<"Rectangle constructor called: \n";}
      int getArea() {
         return(getWidth() * getHeight()); }
        ~Rectangle(){
   cout<<"Rectangle destructor called: \n";
                              }
};

main() {
   Rectangle Rect;
   Rect.setWidth(5);
   Rect.setHeight(7);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() <<
endl;
}
```

Class exercise

- Write a program that defines six classes named: Laboratory, Patient, Medical_Complex, Clinic, Diagnostic_Center. The classes Clinic and Diagnostic_Center are inherited from the class Medical_Complex. The Medical_Complex class should have the data members for address, administrator's name, number of doctors, starting and closing timings. The Clinic class must have a member function to allot time slot to the patients where each patient should be allotted a time of 30 minutes. The time slots should not be allotted beyond the timings of the clinic. The Clinic class has the relation of aggregation with Patient class. The Diagnostic_Center class should have the data member of number of laboratories. The laboratory class has the data member of type of reports the various laboratory generate such as x-rays, CBC etc. Write the setters to set the inputs of all data members of classes and getters to display on screen. The Diagnostic_Center and Clinic classes should have display method to display the values of the data members.

References

- C++ How to Program
  By Deitel & Deitel
- The C++ Programming Language
  By Bjarne Stroustrup
- Object oriented programming using C++ by Tasleem Mustafa, Imran Saeed, Tariq Mehmood, Ahsan Raza
- https://www.tutorialspoint.com/cplusplus
- http://ecomputernotes.com/cpp/introduction-to-oop
- https://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm
- https://www.guru99.com/c-loop-statement.html
- www.w3schools.com
- https://www.geeksforgeeks.org/