Practical Lecture : inheritance 2

Quick Recap

Let’s take a quick recap of previous lecture –

- Inheritance basics – base class , dervied class
- Type of inheritance- simple, multi-level, multiple and hierarchical

Today’s

Today we are going to cover -

- Access specifier (private, protected, public) , Protected members
- Modes (private, protected, public inheritance)
- Overriding member functions,
- Order of execution of constructors and destructors,
- Resolving ambiguities in inheritance,
- Virtual base class.

Let’s Get Started-

Quick recap

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 inheritance:

- 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
- So there will be 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

Implementing inheritance

- 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
};
```

- Here, 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.
- base_class_name is the name of the base class from which you want to inherit the sub class.

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.

Let us understand it with example: First understand how private and public members of base class are affected by modes of inheritance

Practice Question - revisited

```
class student
{
      int rollno;
public:
      student() {rollno=1;}
};
class test: public student  //here  public is mode of inheritance
{
      float  marks;
  public:
      test() { marks=40;}
      void display(void);
};
```

Practice Question

```
void test::display()
{
	//cout<<"Rollno ="<<rollno<<endl; //not accessible here as private in
base
	cout<<"Marks ="<<marks<<endl;
}
int main()
{
	test t1;
	t1.display();
	return 0;
}
```

Output:
Marks =40

Note that we always create objects of derived class and access all the

Few Questions

- What is the mode of inheritance in test (derived class)?
- Can we access rollno in main()? Why? If not, then what is the solution?
- Can we access rollno in test class? Why? If not, then what is the solution?
- Can we access marks in main()?

Few Questions

- What is the mode of inheritance in test (derived class)?
- public

- Can we access rollno in main()? If not, then what is the solution?
- No. because it is private member of student, not accessible outside class. Make it public or access using public methods.

- Can we access rollno in test class? If not, then what is the solution?
- No. because it is private member of student, not accessible outside class, not even derived class. Solution is to make it public or access using public methods.

- Can we access marks in main()?
- No. because it is private member of test, not accessible outside class. Make it public or access using public methods. Here we can access it

Making private members public

```
class student
{
public:
      int rollno;
public:
      student() {rollno=1;}
};
class test: public student
{
public:
      float  marks;
public:
      test() { marks=40;}
      void display(void);
};
```

Making private members public

```
void test::display()
{
    cout<<"Rollno ="<<rollno<<endl; //accessible here as public in base
    cout<<"Marks ="<<marks<<endl;
}
int main()
{
    test t1;
    cout<<"rollno= "<<t1.rollno<<" Marks = "<<t1.marks<<endl;
    t1.display(); //not required now
    return 0;
}
```

Output:

Marks =40

Making private members accesible in derived class

- Solution of making private members public works, but it is against the principle of OOP – Data hiding or encapsulation. Hence you should not make data members of a class public.

- Then what is the solution: how to make base class members accessible in derived class?

- Answer is using by making them protected.

- **Protected members:** The protected members are the members in the base class which can be accessed directly in the derived class. The private members in the base class cannot be directly accessed in the derived class, while protected members can be directly accessed.

Practice Question – protected members

```
class student
{
protected:
        int rollno;
public:
        student() {rollno=1;}
};
class test: public student
{
        float  marks;
  public:
        test() { marks=40;}
        void display(void);
};
```

Practice Question

```
void test::display()
{
	cout<<"Rollno ="<<rollno<<endl; // accessible here as protected in
base
	cout<<"Marks ="<<marks<<endl;
}
int main()
{
	test t1;
	t1.display();
	return 0;
}
```

Output:

Rollno=1

Marks =40

How Modes of inheritance impact the members

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)

Impact on members of modes of inheritance

- **Private members** : Irrespective of mode (type) of inheritance, the private members are not accessible outside the class (not even in main, or further derived classes)

- **Protected members:** If mode of inheritance is public or protected, protected members of base class remain protected in derived class, if mode is private, protected members become private

- **public members:** : If mode of inheritance is public , public members will remain public in derived class . In case of protected mode of inheritance, public members become protected in derived class, if mode is private, public members become private in derived class which cannot be inherited further

- **Note: Teachers are expected to code an example and change the mode from private, protected and public and show the impact on**

Practice Question- observe the inheritance

```
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
};
```

Practice Question- observe the inheritance

```
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
};
```

MCQ

When the inheritance is private, the private members of the base class are _____ in the derived class

A. Inaccessible
B. Accessible
C. Protected
D. Private

MCQ

When the inheritance is private, the private members of the base class are _____ in the derived class

A. Inaccessible
B. Accessible
C. Protected
D. Private

Answer: option A

MCQ

MCQ

Predict the error/ output

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

class Base
{
protected:
    int a;
public:
    Base() {a = 0;}
};
 class Derived1:  protected  Base
{
protected:
    int b;
};
```

Predict the error/output

```
class Derived2:  private Derived 1
{
private:
    int c;
};
class DerivedDerived: public Derived2
{
public:
    void show()  {   cout << a <<endl<<b<<endl<<c;   }
};
int main(void)
{
    DerivedDerived d;
    d.show();
    return 0;
}
```

Predict the error/output

```
main.cpp: In member function ‘void DerivedDerived::show()’:
main.cpp:28:30: error: ‘int Base::a’ is protected within this context
    void show()  {   cout << a <<endl<<b<<endl<<c; }
                             ^
main.cpp:7:9: note: declared protected here
    int a;
        ^
main.cpp:28:40: error: ‘int Derived1::b’ is protected within this context
    void show()  {   cout << a <<endl<<b<<endl<<c; }
                                       ^
main.cpp:15:9: note: declared protected here
    int b;
        ^
main.cpp:28:49: error: ‘int Derived2::c’ is private within this context
    void show()  {   cout << a <<endl<<b<<endl<<c; }
                                             ^
main.cpp:22:9: note: declared private here
    int c;
```

Assignment

Create two classes Cuboid and CubiodVol. Cuboid with three data fields-
length, *width* and *height* of *int* types. The class should have d*isplay()* method, to print the length, *width* and *height* of the cuboid separated by space. The*CuboidVol* class is derived from Cuboid class. The class should have *read_input()* method, to read the values of length, *width* and *height* of the Cuboid. The *CuboidVol* class should also the *displayVol()* method to print the volume of the Cuboid (length * width * height).

Output expected:
If length = 12, width = 10 and height = 2
Volume of the cuboid is = (length * width * height)
= 12 * 10 * 2
= 240

Note: Assume necessary data wherever required

Assignment

Use the concept of multi-level inheritance. Create a class student with roll number as a member.
Create 2 classes:
Test: containing the marks of a student in 5 subjects inheriting class student (having roll number of the student).
Result: containing the function Display() to compute the total and average and then displaying the output as Roll number, total and average which are space separated.

Note: Assume necessary data wherever required

Assignment

Create a class shape with attributes as length and breath of float type. Create derived classes rectangle , circle to calculate area of them. Have display methods in both of these derived classes to display the areas calculated .

Note: Assume necessary data wherever required

Overriding member functions

Earlier we have discussed **function overloading** where same function takes various forms.

The function name is same , but the parameter list changes

Now let is see the concept of function overriding

If the member function in defined in both the derived class and the based class with the same name and same number/type of parameters, then the concept is called as **function overriding**

The function in derived class overrides the function in base class.

It is the redefinition of base class function in its derived class with same signature i.e return type and parameters.

Overriding member functions

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

class Base {
   public:
    void print() {
        cout << "Base Function" << endl;
    }
};

class Derived : public Base {
   public:
    void print() {
        cout << "Derived Function" << endl;
    }
};
```

Overriding member functions

```
int main() {
Base base1;
base1.print();
return 0;
}
```

Output: Base Function

Had we called the print() function from an object of the Base class, the function would not have been overridden.

Overriding member functions

```
int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```

Output: Derived Function

Here, the same function print() is defined in both Base and Derived classes. So, when we call print() from the Derived object derived1, the print() from Derived is executed by overriding the function in Base. The function was overridden because we called the function from an object of the Derived class.

Access Overriding member functions using ::

Consider above Base and Derived class

```
int main() {
    Derived derived1, derived2;
    derived1.print();

    // access print() function of the Base class
    derived2.Base::print();

    return 0;
}
```

Output:
Derived Function
Base Function

The base class function can be accessed using scope resolution operator.

Access Overriding member functions using :: - another way

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

class Base {
   public:
    void print() {
        cout << "Base Function" << endl;
    }
};

class Derived : public Base {
   public:
    void print() {
        cout << "Derived Function" << endl;
        Base::print(); //call overridden function
    }
};
```

Access Overriding member functions using :: - another way

Consider above Base and Derived class

```
int main() {
    Derived derived1, derived2;
    derived1.print();
    return 0;
}
```

Output:
Derived Function
Base Function

Notice the code Base::print();, which calls the overridden function inside the Derived class.

MCQ

```
Class A {       public : fun(); }
Class B {       public : fun(); }
int main() {
   A a;
   B b;
   // line 3 - call fun() that belongs to base class
}
```

What is the correct way to call a overridden base class function from main() at line 3?

A. a.fun();
B. b.fun();
C. b.A::fun();
D. a.B::fun();

MCQ

```
Class A {       public : fun(); }
Class B {       public : fun(); }
int main() {
   A a;
   B b;
   // line 3 - call fun() that belongs to base class
}
```

What is the correct way to call a overridden base class function from main() at line 3?

A. a.fun();
B. b.fun();
C. b.A::fun();
D. a.B::fun();

Answer: option C

MCQ

What is the true about overloading and overriding a function?

1. Function overloading is function with same name but different parameters
2. Function overriding is function with same name, same parameters and same return type
3. Function overloading is function with same name, same parameters and different return type
4. Function overriding is function with different name but same number of parameters

A. 1 & 2
B. 3 & 4
C. 1 & 4
D. 2 & 3

MCQ

What is the true about overloading and overriding a function?

1. Function overloading is function with same name but different parameters
2. Function overriding is function with same name, same parameters and same return type
3. Function overloading is function with same name, same parameters and different return type
4. Function overriding is function with different name but same number of parameters

A. 1 & 2
B. 3 & 4
C. 1 & 4
D. 2 & 3

Answer: option A

MCQ

What is the true about function overriding?

1. It is function with same name , but it can differ in parameter list and types
2. It can only be implemented using inheritance
3. It is the function with same name , same parameter list and same return type
4. It is same as function overloading except function name is different in function overriding.

A. 1 & 2
B. 3 & 4
C. 1 & 4
D. 2 & 3

Answer: option D


Any
Questions ??

Thank You!

See you guys in next class.