FAST

National University of Computer and Emerging Sciences Peshawar

OOP Lab # 13

C++ (Inheritance)

Computer Instructor: Muhammad Abdullah Orakzai

DEPARTMENT OF COMPUTER SCIENCE

C++ Programming

الذى علم بالقلم. علم الانسان ما لم يعلم.

Contents

Function Overriding in C++

❖ As we know, inheritance is a feature of OOP that allows us to create derived classes from a base class. The derived classes inherit features of the base class.

❖ Suppose, the same function is defined in both the derived class and the based class. Now if we call this function using the object of the derived class, the function of the derived class is executed.

❖ This is known as **function overriding** in C++. The function in derived class overrides the function in base class.

Function Overriding Example 1

```cpp
// C++ program to demonstrate function overriding

#include <iostream>
using namespace std;

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

Function Overriding Example 1…

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

int main() {
    Derived derived1;
    //Base base;
    //base.print();
    derived1.print();
    return 0;
}
```
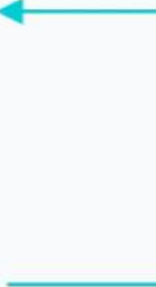
Output:
Derived Function

Function Overriding Example 1…

❖ 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.

Working of function overriding in C++

```cpp
class Base {
    public:
      void print() {
          // code
      }
};

class Derived : public Base {
    public:
      void print() {
          // code
      }
};

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

Function Overriding Example 1…

❖ As we can see, the function was overridden because we called the function from an object of the Derived class.

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

// Call function of Base class

```
Base base1;
base1.print();
```

// Output: Base Function

Access Overridden Function in C++

❖ To access the overridden function of the base class, we use the scope resolution operator **::** .

❖ We can also access the overridden function by using a pointer of the base class to point to an object of the derived class and then calling the function from that pointer.

Example 2: C++ Access Overridden Function to the Base Class

```cpp
// C++ program to access overridden function
// in main() using the scope resolution operator
::

#include <iostream>
using namespace std;

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

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

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

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

    return 0;
}
```

Output:
Derived Function
Base Function

Here, this statement
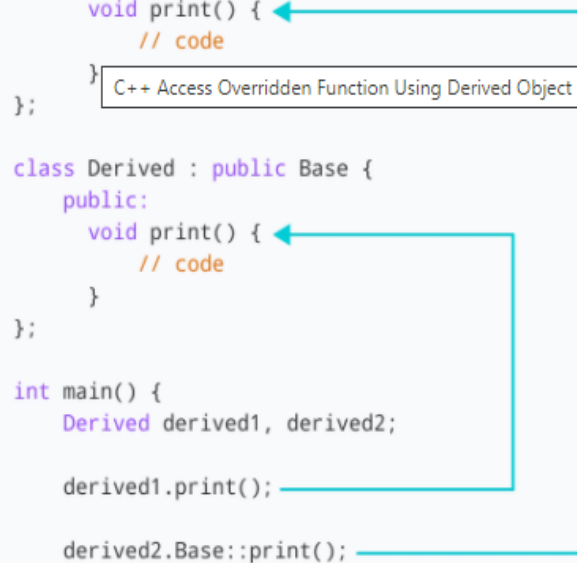
derived2.Base::print();

accesses the print() function of the Base class.

Access overridden function using object of derived class in C++

```cpp
class Base {
    public:
    void print() {
        // code
    }
};

class Derived : public Base {
    public:
    void print() {
        // code
    }
};

int main() {
    Derived derived1, derived2;

    derived1.print();

    derived2.Base::print();

    return 0;
}
```

C++ Access Overridden Function Using Derived Object

```cpp
// C++ program to call the overridden function
// from a member function of the derived class

#include <iostream>
using namespace std;

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

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

        // call overridden function
        Base::print();
    }
};

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

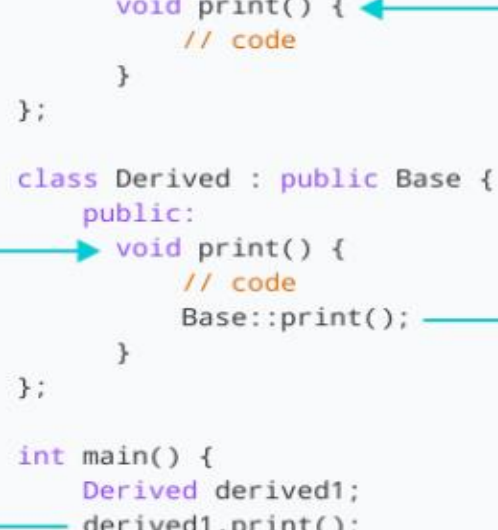Output:
Derived Function
Base Function

In this program, we have called the overridden function inside the **Derived** class itself.

```cpp
class Derived : public Base {

  public:

   void print() {

     cout << "Derived Function" << endl;

     // call overridden function

     Base::print();

   }

};
```

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

16

```cpp
class Base {
    public:
     void print() {
          // code
     }
};

class Derived : public Base {
    public:
     void print() {
          // code
          Base::print();
     }
};

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

```cpp
// C++ program to access overridden function using pointer
// of Base type that points to an object of Derived class

#include <iostream>
using namespace std;

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

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

int main() {
    Derived derived1;

    // pointer of Base type that points to derived1
    Base* ptr = &derived1;

    // call function of Base class using ptr
    ptr->print();
    return 0;
}
```

Output:
Base Function

19

❖ In this program, we have created a pointer of **Base** type named **ptr**. This pointer points to the **Derived** object **derived1**.

// pointer of Base type that points to derived1

   Base* ptr = &derived1;

❖ When we call the **print()** function using **ptr**, it calls the overridden function from Base.

// call function of Base class using ptr

   ptr->print();

❖ This is because even though ptr points to a Derived object, it is actually of Base type. So, it calls the member function of Base.

❖ In order to override the Base function instead of accessing it, we need to use **virtual functions** in the Base class.

Types of Inheritance

Inheritance is one of the core feature of an object-oriented programming language. It allows software developers to derive a new class from the existing class. The derived class inherits the features of the base class (existing class).

There are various models of inheritance in C++ programming.

1) Single Inheritance

2) Multilevel Inheritance

3) Multiple Inheritance

4) Hierarchical Inheritance

5) Hybrid Inheritance
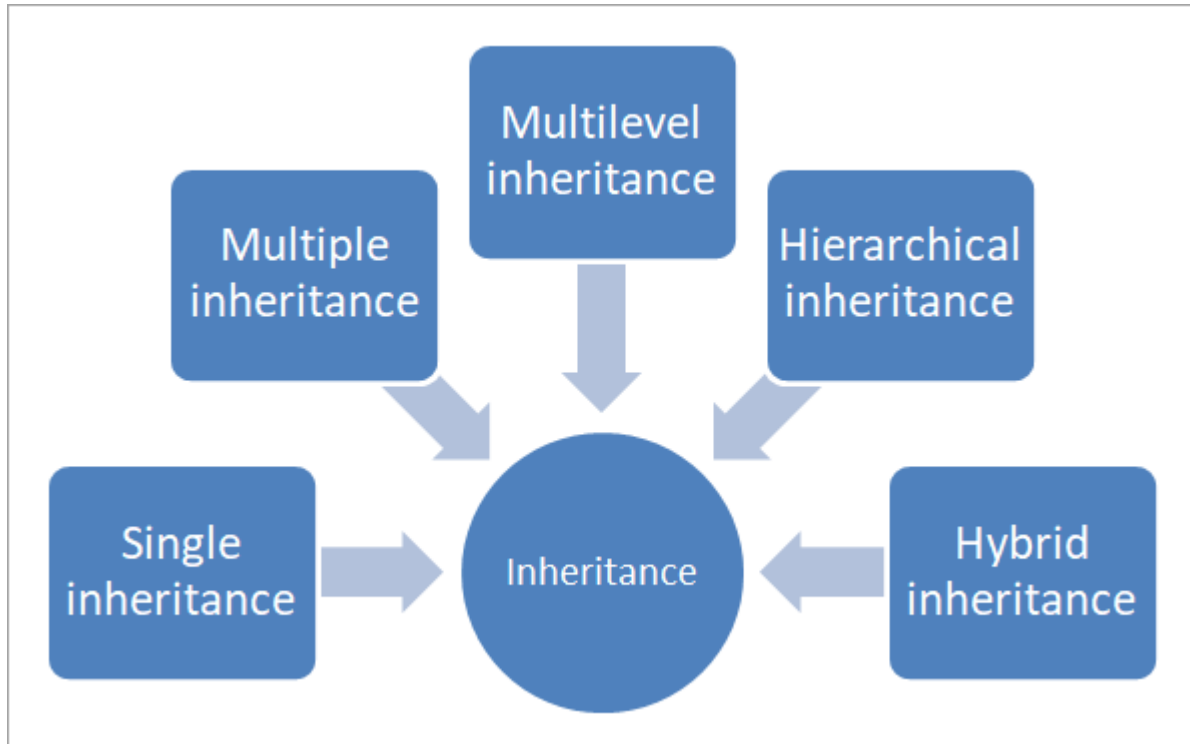
1) Single Inheritance

❖ It is the inheritance hierarchy wherein one derived class inherits from one base class.

❖ There is only one Super Class and Only one Sub Class Means they have one to one Communication between them
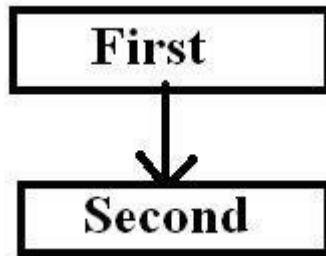
2) Multilevel Inheritance

❖ A Derived class can also be inherited by another class Means in this When a Derived Class again will be inherited by another Class then it creates a Multiple Levels.

❖ It is the inheritance hierarchy wherein subclass acts as a base class for other classes.

❖ In C++ programming, not only 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.

2) Multilevel Inheritance…

❖ It is implemented by defining at least three classes. Each derived class must have a kind of relationship with its immediate base class.
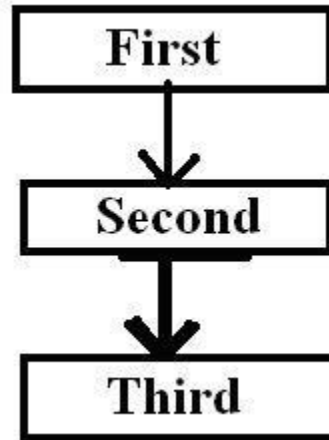
```
class A {

… .. …

};

class B: public A {

 … .. …

};

class C: public B {

 … … …

};
```

Here, class B is derived from the base class A and the class C is derived from the derived class B.

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

class A {
    public:
      void display() {
          cout<<"Base class content.";
      }
};
```

```
class B : public A {};

class C : public B {};

int main() {
    C obj;
    obj.display();
    return 0;
}
```

Output:
Base class content.

Example 1: C++ Multilevel Inheritance…

❖ In this program, class C is derived from class B (which is derived from base class A).

❖ The obj object of class C is defined in the main() function.

❖ When the display() function is called, display() in class A is executed. It's because there is no display() function in class C and class B.

❖ The compiler first looks for the display() function in class C. Since the function doesn't exist there, it looks for the function in class B (as C is derived from B).

❖ The function also doesn't exist in class B, so the compiler looks for it in class A (as B is derived from A).

❖ If display() function exists in C, the compiler overrides display() of class A (because of member function overriding).

Example 2: C++ Multilevel Inheritance

```cpp
#include<iostream>
using namespace std;
class person {
  private:
    char name[20];
    long int phno;
  public:
    void read() {
     cout<<"Enter name and phno = ";
     cin>>name>>phno;
    }
    void show() {
     cout<<"\nName = "<<name;
     cout<<"\nPhone number = "<<phno;
    }
}; // end of person class
```

Example 2: C++ Multilevel Inheritance…

```cpp
class student : public person {
  private:
    int rollno;
    char course[20];
  public:
    void read() {
      person::read();//Access person's read()
      cout<<"Enter rollno and course=";
      cin>>rollno>>course;
    }
    void show() {
      person::show(); //Access person's show()
      cout<<"\nRollno = "<<rollno;
      cout<<"\nCourse ="<<course;
    }
}; // end of student class
```

```cpp
class exam : public student {
 private:
    int m[4];
    double per;
 public:
    void read() ;
    void cal();
    void show() ;
};  // end of exam class
```

Example 2: C++ Multilevel Inheritance…

```cpp
// Function definitions
  void exam::read() {
    student::read();//Access student's read()
    cout<<"Enter marks :";
    for(int i=0;i<4;i++)
      cin>>m[i];
  }
  void exam::cal() {
    int tot_marks=0;
    for(int i=0;i<4;i++)
      tot_marks+=m[i];
      per=double(tot_marks)/4;
  }
```

```
void exam::show() {
    student::show();//Access student's show()
    cout<<"\nMarks :";
    for(int i=0;i<4;i++)
      cout<<m[i]<<"\t";
      cout<<"\nPercentage = "<<per;
      }
//main function
int main() {
   exam e1;
   e1.read();
   e1.cal();
   e1.show();
   return 0;
}
```

Output:
Name = Aisha
Phone number = 57868882
Rollno = 1222
Course =Computer
Marks :88   77   66   98
Percentage = 82.25

Explanation: The above program implements multilevel inheritance. The program inputs the students basic and academic information calculates the result and displays all the students' information. The program consists of three classes, namely person, student and exam. The class person is the base class. The class student is derived from person base class, which acts as a base class for the derived class exam. The exam class inherits the features of student class directly and features of person class indirectly. In main(),the statement

<div align="center">

e1.read();

e1.show();

</div>

Invokes the respective member functions of exam class as e1 is the object of this class. Look carefully at the definition of read() and show() of the exam class, and you will find that both the member functions access same-named member function of the student class directly and person class indirectly. The statement,

e1.cal();

Invokes the member function cal() of exam class for calculating the result of a student.

3) Multiple Inheritance

❖ It is the inheritance hierarchy wherein one derived class inherits from multiple base class(es).

❖ So far, we have been deriving a class from a single base class. However, it is also possible to derive a class from two or more unreleased base classes. When we derive a class from two or more base classes, this form of inheritance is known as Multiple Inheritance. It allows the new class to inherit the functionality of two or more well developed and tested base classes, thus reusing predefined classes' features.

❖ Like Single Inheritance, each derived class in multiple inheritances must have a kind of relationship with its base classes.

3) Multiple Inheritance…

❖ This form of inheritance is useful in those situations in which derived class represents a combination of features from two or more base classes.

❖ **For Example**: In an educational institute, a member can act like a teacher and a student. He may be a student of higher class as well as teaching lower classes.
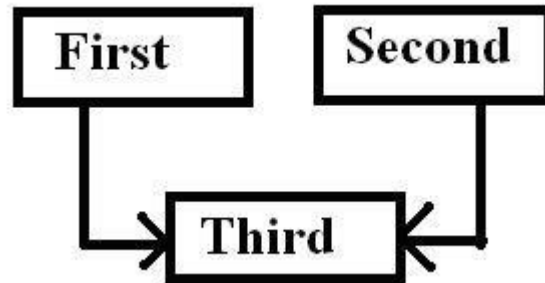
Multiple Inheritance Syntax

The syntax for declaring a derived class which inherited from more than one base classes is:

class der_class_name : **access_specifier** base1 , **access_specifier** base2, ... {

 //Implementation
};

Here access_specifier may be either public, private or protected. Each base class from which class is derived must be specified to the colon's right after the access specifier and must be separated by commas.

3) Multiple Inheritance Syntax…

❖ The Fig. shows multiple inheritances in which class Third is derived from class First and class Second, which are independent of each other.

❖ Here, class Third inherits all the members of its base classes First and Second. The following program segment shows how a class Third is derived from two base classes First and Second publicly.
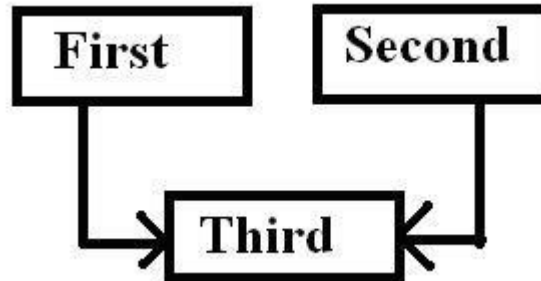
```
class Third: public First, public Second

{

//Implementation

};
```

❖ The inheritance rules and the usages of access specifier for each base class member remain the same as in single inheritance. The access to each base class member is independent of the access to the members of other base classes using which you derive your class.

```cpp
#include<iostream>
using namespace std;
class base1 {
  protected:
    int x;
  public:
    void readx() {
      cout<<"Enter value of x : ";
      cin>>x;
    }
    void showx()
    {
        cout<<"x= "<<x;
    }
};   // end of class base1
```

```
class base2 {
  protected:
    int y;
  public:
    void ready() {
      cout<<"Enter value of y: ";
      cin>>y;
    }
    void showy()
      {
          cout<<"\ny= "<<y;
      }
}; // end of class base2
```

```
class der : public base1,public base2 {
  private:
    int z;
  public:
    void add() {
      z=x+y; //Accessing protected members of base classes
    }
    void showz ()
      {
          cout<<"\nz = "<<z;
      }
}; // end of class der1
```

```
int main() {
  der d1;
  d1.readx();//calling readx() of base1
  d1.ready();//calling ready() of base2
  d1.add();//calling add() of der

  d1.showx();//calling showx() of base1
  d1.showy();//calling showy() of base2
  d1.showz();//calling showz() of der
  return 0;
}
```

Output:
Enter value of x : 22
Enter value of y: 11
x= 22
y= 11
z = 33

46

Explanation: In this program, we define a class base1 containing a protected data member x and public member functions readx() and showx() for inputting and displaying the value of x. Similarly, another class base2 is defined that contains a data member y and two public member function ready() and showy(). The class der is inherited from both the base classes base1 and base2 publicly through multiple inheritances. As the class der is derived publicly from both the base class base1 and base2, the derived class object can access public base el and base2 in the main(). In the main() , the statement d1.readx() ; invokes the member function readx() of the base1 class for inputting value of x. Similarly the statement, d1.ready() ; invokes the ready() of base2 for inputting value of y. The statement d1.add(); invokes the member function add() of the der class to perform addition of values of protected data members x and y as they can be accessed directly in the der class. The remaining statement displays the values of x, y and z.

Ambiguity in Multiple Inheritance

❖ The most obvious problem with multiple inheritance occurs during function overriding.

❖ Suppose, two base classes have a same function which is not overridden in derived class.

❖ If you try to call the function using the object of the derived class, compiler shows error. It's because compiler doesn't know which function to call.

❖ For example,

Ambiguity in Multiple Inheritance…

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

class base1 {
  public:
      void someFunction( )
      {
          cout<<"Function of base1";
      }
};  // end of base1 class
class base2 {
  public:
    void someFunction( )
    {
        cout<<"Function of base1";
    }
}; // end of base2 class
```

Ambiguity in Multiple Inheritance…

```
class derived : public base1, public base2 {
    // code of derived class
};

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

❖ This problem can be solved using the scope resolution function to specify which function to class either **base1** or **base2**

```
int main() {
    derived obj;

    obj.base1::someFunction( );  // Function of base1 class is called

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

4) Hierarchical Inheritance

❖ If more than one class is inherited from the base class, it's known as hierarchical inheritance. In hierarchical inheritance, all features that are common in child classes are included in the base class.

❖ For example, Physics, Chemistry, Biology are derived from Science class. Similarly, Dog, Cat, Horse are derived from Animal class.
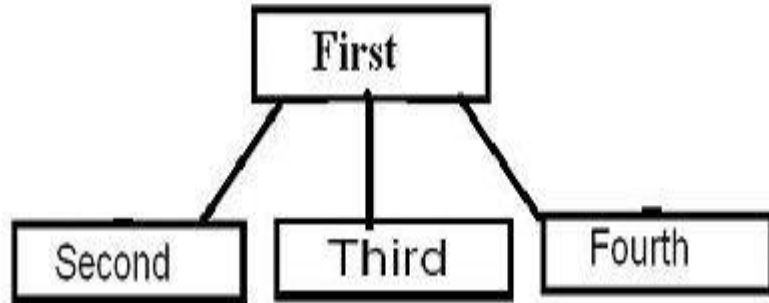
Syntax of Hierarchical Inheritance

```
class base_class {
    ... .. ...
}

class first_derived_class: public base_class {
    ... .. ...
}

class second_derived_class: public base_class {
    ... .. ...
}

class third_derived_class: public base_class {
    ... .. ...
}
```

```cpp
// C++ program to demonstrate hierarchical in
heritance
#include <iostream>
using namespace std;

// base class
class Animal {
   public:
    void info() {
        cout << "I am an animal." << endl;
    }
};
```

```cpp
// derived class 1
class Dog : public Animal {
   public:
    void bark() {
        cout << "I am a Dog. Woof woof." << endl;
    }
};

// derived class 2
class Cat : public Animal {
   public:
    void meow() {
        cout << "I am a Cat. Meow." << endl;
    }
};
```

```cpp
int main() {
    // Create object of Dog class
    Dog dog1;
    cout << "Dog Class:" << endl;
    dog1.info();  // Parent Class function
    dog1.bark();

    // Create object of Cat class
    Cat cat1;
    cout << "\nCat Class:" << endl;
    cat1.info();  // Parent Class function
    cat1.meow();

    return 0;
}
```

Output:

Dog Class:
I am an animal.
I am a Dog. Woof woof.

Cat Class:
I am an animal.
I am a Cat. Meow.

Here, both the **Dog** and **Cat** classes are derived from the **Animal** class. As such, both the derived classes can access the **info()** function belonging to the **Animal** class.

Example 2: Hierarchical Inheritance in C++ Programming

❖ The following program implements hierarchical Inheritance. Here, we define a base class person. Then we derive two classes teacher and student. The derived classes inherit the common member's name, phno, read(), show() from the base class person, and have its members.

❖ The public read() and show() of the base class redefined in each derived class to input and display teacher and student information.

```cpp
#include<iostream>
using namespace std;
class person {
  private:
    char name[20];
    long int phno;
  public:
    void read() {
      cout<<"Enter name and phno = ";
      cin>>name>>phno;
    }
    void show() {
      cout<<"\nName = "<<name;
      cout<<"\nPhone number = "<<phno;
    }
}; //Base class specification ends
```

```
class student : public person {
  private:
    int rollno;
    char course[20];
  public:
    void read() {
      person::read();
      cout<<"Enter rollno and course=";
      cin>>rollno>>course;
    }
    void show() {
      person::show();
      cout<<"\nRollno = "<<rollno;
      cout<<"\nCourse = "<<course;
    }
}; //Derived class specification ends
```

```cpp
class teacher : public person {
   private:
     char dept_name[10];
     char qual[10];
   public:
     void read() {
       person::read();
       cout<<"Enter dept_name and qualification = ";
       cin>>dept_name>>qual;
       }
     void show() {
       person::show();
       cout<<"\nDepartment name = "<<dept_name;
       cout<<"\nQualification = "<<qual;
       }
}; //Derived class specification ends
```

```cpp
int main() {
    student s1;
    cout<<"Enter student Information\n";
    s1.read();
    cout<<"Displaying Student Information ";
    s1.show();
    teacher t1;
    cout<<"\nEnter teacher Information\n";
    t1.read();
    cout<<"Displaying teacher Information ";
    t1.show();
    return 0;
}
```

Output:
```
Enter student Information
Enter name and phno = Ausaf 0344
Enter rollno and course=44 OOP

Displaying Student Information
Name = Ausaf
Phone number = 344
Rollno = 44
Course = OOP

Enter teacher Information
Enter name and phno = Abdullah
0332
Enter dept_name and qualification = Computer MS

Displaying teacher Information
Name = Abdullah
Phone number = 332
Department name = Computer
Qualification = MS
```
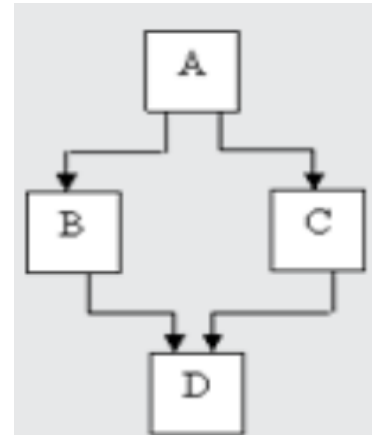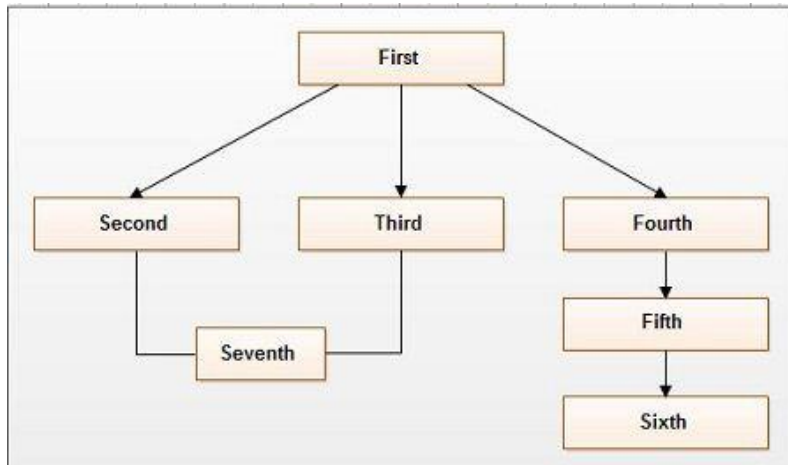
Explanation: In main (), we first creates an object s1 of class student through which we call the read() and show() member functions of student's class for inputting and displaying student information .

Then similarly, we create an object t1 of teacher class for inputting and displaying teacher information.

5) Hybrid Inheritance

The inheritance hierarchy that reflects any legal combination of other four types of inheritance.

This is a Mixture of two or More Inheritance and in this Inheritance a Code May Contains two or Three types of inheritance in Single Code.

References

- https://beginnersbook.com/2017/08/cpp-data-types/

- http://www.cplusplus.com/doc/tutorial/basic_io/

- https://www.w3schools.com/cpp/default.asp

- https://www.javatpoint.com/cpp-tutorial

- https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/?ref=lbp

- https://www.programiz.com/

- https://ecomputernotes.com/cpp/

THANK YOU