LAB NO. 09

INHERITANCE IN C++

Lab Objectives

Following are the lab objectives:

1. Inheritance in C++

Instructions

- This is individual Lab work/task.
- Complete this lab work within lab timing.
- Discussion with peers is not allowed.
- Copy paste from Internet will give you **negative marks**.
- Lab work is divided into small tasks, complete all tasks sequentially.

C++ Inheritance

In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.

In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

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. We can summarize the different access types according to - who can access them in the following way –

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

A derived class inherits all base class methods with the following exceptions −

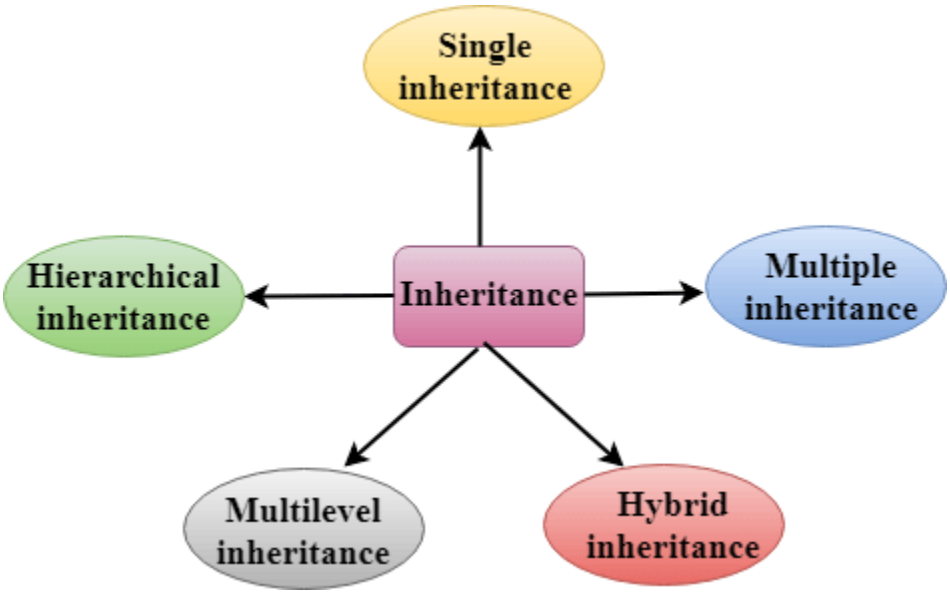
- Constructors, destructors and copy constructors of the base class.
- Overloaded operators of the base class.
- The friend functions of the base class.

Types of Inheritance

C++ supports five types of inheritance:

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance

- Hybrid inheritance



Derived Classes

A Derived class is defined as the class derived from the base class.

The Syntax of Derived class:

```
class derived_class_name:visibility-mode base_class_name
{
   // body of the derived class.
}
```

Where,

derived_class_name: It is the name of the derived class.

visibility mode: The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.

Type of Inheritance

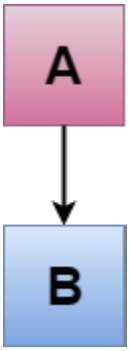
When deriving a class from a base class, the base class may be inherited through **public, protected** or **private** inheritance. The type of inheritance is specified by the access-specifier as explained above.

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.

C++ Single Inheritance

Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.



Where 'A' is the base class, and 'B' is the derived class.

C++ Single Level Inheritance Example: Inheriting Fields

When one class inherits another class, it is known as single level inheritance. Let's see the example of single level inheritance which inherits the fields only.

```
#include <iostream>
using namespace std;
 class Account {
   public:
   float salary = 60000;
 };
   class Programmer: public Account {
   public:
```

```
  float bonus = 5000;
  };
int main(void) {
    Programmer p1;
    cout<<"Salary: "<<p1.salary<<endl;
    cout<<"Bonus: "<<p1.bonus<<endl;
  return 0;
}
```

Output:

```
Salary: 60000
Bonus: 5000
```

In the above example, Employee is the **base** class and Programmer is the **derived** class.

C++ Single Level Inheritance Example: Inheriting Methods

```
#include <iostream>
using namespace std;
class A
{
  int a = 4;
  int b = 5;
  public:
  int mul()
  {
    int c = a*b;
    return c;
  }
};

class B : private A
{
  public:
  void display()
  {
```

```
        int result = mul();
        std::cout <<"Multiplication of a and b is : "<<result<< std::endl;
    }
};
int main()
{
   B b;
   b.display();

    return 0;
}
```

Output:

```
Multiplication of a and b is : 20
```

In the above example, class A is privately inherited. Therefore, the mul() function of class 'A' cannot be accessed by the object of class B. It can only be accessed by the member function of class B.

C++ Multilevel Inheritance

Multilevel inheritance is a process of deriving a class from another derived class.

C++ Multi Level Inheritance Example

When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C++. Inheritance is transitive so the last derived class acquires all the members of all its base classes.

Let's see the example of multi level inheritance in C++.

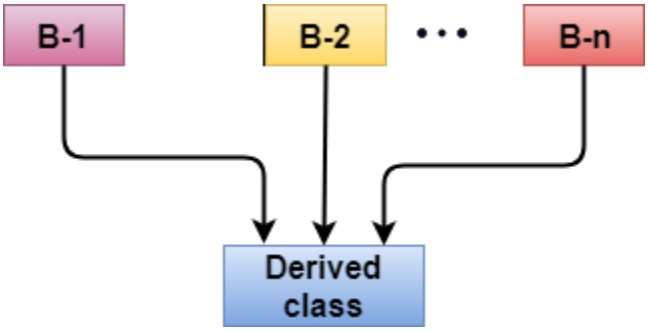
```
#include <iostream>
using namespace std;
 class Animal {
   public:
 void eat() {
    cout<<"Eating..."<<endl;
 }
   };
   class Dog: public Animal
   {
      public:
     void bark(){
    cout<<"Barking..."<<endl;
     }
   };
   class BabyDog: public Dog
   {
      public:
     void weep() {
    cout<<"Weeping...";
     }
   };
int main(void) {
    BabyDog d1;
    d1.eat();
    d1.bark();
     d1.weep();
     return 0;
}
```

Output:

```
Eating...
Barking...
Weeping...
```

C++ Multiple Inheritance

Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.



Syntax of the Derived class:

```
class D : visibility B-1, visibility B-2, ?
{
   // Body of the class;
}
```

Let's see a simple example of multiple inheritance.

```
#include <iostream>
using namespace std;
class A
{
   protected:
    int a;
   public:
   void get_a(int n)
   {
      a = n;
   }
```

```
};
class B
{
    protected:
    int b;
    public:
    void get_b(int n)
    {
        b = n;
    }
};
class C : public A,public B
{
   public:
    void display()
    {
        std::cout << "The value of a is : " <<a<< std::endl;
        std::cout << "The value of b is : " <<b<< std::endl;
        cout<<"Addition of a and b is : "<<a+b;
    }
};
int main()
{
   C c;
   c.get_a(10);
   c.get_b(20);
   c.display();

    return 0;
}
```

Output:

```
The value of a is : 10
The value of b is : 20
Addition of a and b is : 30
```

In the above example, class 'C' inherits two base classes 'A' and 'B' in a public mode.

Ambiquity Resolution in Inheritance

Ambiguity can be occurred in using the multiple inheritance when a function with the same name occurs in more than one base class.

Let's understand this through an example:

```
#include <iostream>
using namespace std;
class A
{
   public:
   void display()
   {
      cout << "Class A" <<endl;
   }
};
class B
{
   public:
   void display()
   {
      cout << "Class B" <<endl;
   }
};
class C : public A, public B
{
   void view()
   {
      display();
   }
};
int main()
{
```

```
    C c;
    c.display();
    return 0;
}
```

Output:

```
error: reference to 'display' is ambiguous
        display();
```

- The above issue can be resolved by using the class resolution operator with the function. In the above example, the derived class code can be rewritten as:

```
class C : public A, public B
{
    void view()
    {
        A :: display();        // Calling the display() function of class A.
        B :: display();        // Calling the display() function of class B.

    }
};
```

An ambiguity can also occur in single inheritance.

Consider the following situation:

```
class A
{
  public:
void display()
{
  cout<<?Class A?;
}
} ;
class B
{
 public:
```

```
 void display()
{
 cout<<?Class B?;
}
} ;
```

In the above case, the function of the derived class overrides the method of the base class. Therefore, call to the display() function will simply call the function defined in the derived class. If we want to invoke the base class function, we can use the class resolution operator.

```
int main()
{
   B b;
  b.display();              // Calling the display() function of B class.
  b.B :: display();       // Calling the display() function defined in B class.
}
```

Lab Tasks

Q1). Write a class Employee that contains attributes of employee id and his salary. The class contains member functions to input and show the attribute. Write a child class Manager that inherits from Employee class. The child class has attributes of manager id and his department. It also contains the member functions to input and show its attribute.

Q2). 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

Q3). Create a C++ program in which we can calculate the total marks of each student of a class in OOP, data structure and database and the average marks of the class. The number of students in the class are entered by the user. Create a class named Marks with data members for roll number, name and marks. Create three other classes inheriting the Marks class, OOP, data structure and database, which are used to define marks in individual subject of each student.