Experiment No: 5 Date:

Aim: To study fundamentals of Inheritence

Theory:

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows one class (the derived or child class) to inherit properties and behavior from another class (the base or parent class). This promotes code reuse and facilitates the creation of hierarchies of related classes.

1. Syntax of Inheritance:

In C++, you specify inheritance using a colon `:` followed by the access specifier and the name of the base class. For example:

```
class DerivedClass : accessSpecifier BaseClass {
    // ...
};
```

Access specifiers can be `public`, `protected`, or `private`, determining how members of the base class are accessible in the derived class.

2. Types of Inheritance:

- Public Inheritance: In this type of inheritance, public members of the base class become public members of the derived class, protected members become protected, and private members remain inaccessible. This is the most common form of inheritance.

- Protected Inheritance: The public and protected members of the base class become protected members of the derived class. Private members remain inaccessible.

- Private Inheritance: Both public and protected members of the base class become private members of the derived class. This is rarely used and is more akin to composition than inheritance.

3. Derived Class Constructors:

- When an object of a derived class is created, the constructor of the base class is called first, followed by the constructor of the derived class.

- If the base class has multiple constructors, you can specify which one to call in the initialization list of the derived class.

4. Access Specifiers:

- Public members of the base class remain public in the derived class.
- Protected members of the base class remain protected in the derived class.
- Private members of the base class are not directly accessible in the derived class.

5. Overriding Functions:

- A derived class can provide a specific implementation for a function defined in the base class. This is known as function overriding.

- The function in the derived class must have the same signature (i.e., same name and parameter list) as the one in the base class.

6. Multiple Inheritance:

- C++ allows a class to inherit from multiple base classes. This can lead to complications, so it should be used with caution.

7. Virtual Functions:

- Virtual functions allow dynamic binding and late binding, enabling polymorphic behavior. They are particularly useful when working with base class pointers to derived class objects.

8. Forms of inheritence

1. Single Inheritance:
   - In single inheritance, a derived class inherits from only one base class.
   - This is the simplest form of inheritance and is commonly used to model an "is-a" relationship.

2. Multilevel Inheritance:
   - In multilevel inheritance, a derived class inherits from another derived class, which itself inherits from a base class.
   - This creates a hierarchical structure with multiple levels of inheritance.

3. Hierarchical Inheritance:
   - In hierarchical inheritance, multiple derived classes inherit from the same base class.
   - This creates a branching structure with multiple classes sharing common properties and behavior.

4. Multiple Inheritance:
   - In multiple inheritance, a derived class inherits from more than one base class.
   - This can lead to complications and the "diamond problem," where ambiguity arises due to two base classes having a common ancestor.

5. Hybrid (Virtual) Inheritance:
   - Hybrid inheritance is a way to deal with the ambiguity problem in multiple inheritance.
   - It uses virtual inheritance to ensure that there's only one instance of the common base class.

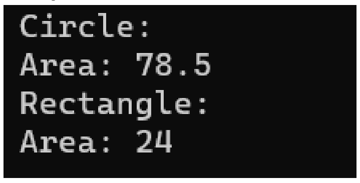
[A] Write a C++ program to study implementation of Hierarchical form of Inheritance

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

class Shape {
public:
   float area;
   void getArea() {
      cout << "Area: " << area << endl;
   }
};
```

```
class Circle : public Shape {
public:
   float radius;
   Circle(float r) {
      radius = r;
      area = 3.14 * radius * radius;
   }
};

class Rectangle : public Shape {
public:
   float length, width;
```

```
  Rectangle(float l, float w) {
    length = l;
    width = w;
    area = length * width;
  }
};

int main() {
  Circle circle(5);
  Rectangle rectangle(4, 6);

  cout << "Circle:" << endl;
  circle.getArea();

  cout << "Rectangle:" << endl;
  rectangle.getArea();

  return 0;
}
```

Output:

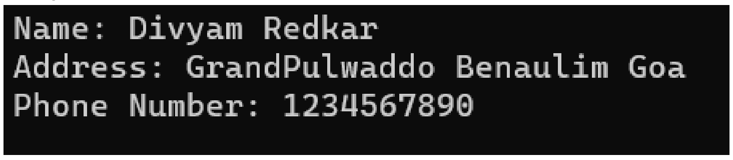


```
Circle:
Area: 78.5
Rectangle:
Area: 24
```

[B] Write a C++ program to study implementation of Hybrid form of Inheritance(Virtual Base Class)

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

class Person {
public:
  string name;
  void setName(string &n) {
    name = n;
  }
};

class Address : virtual public Person {
public:
  string address;
  void setAddress(string &addr) {
    address = addr;
  }
};

class Contact : virtual public Person {
public:
  string phoneNumber;
  void setPhoneNumber(const string &phone) {
    phoneNumber = phone;
  }
};

class ContactInfo : public Address, public Contact {
public:
  void display() {
    cout << "Name: " << name << endl;
    cout << "Address: " << address << endl;
    cout << "Phone Number: " << phoneNumber <<
endl;
  }
};

int main() {
  ContactInfo contactInfo;
  contactInfo.setName("Divyam Redkar");
  contactInfo.setAddress("GrandPulwaddo Benaulim
Goa");
  contactInfo.setPhoneNumber("1234567890");
  contactInfo.display();
  return 0;
}
```

Output:



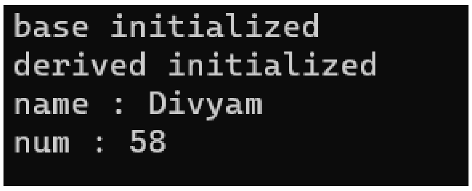
```
Name: Divyam Redkar
Address: GrandPulwaddo Benaulim Goa
Phone Number: 1234567890
```

[C] Write a C++ program to study concept of constructors in derived classes.

```
#include<iostream>
#include<string>

using namespace std;

class base
{
  int num;
public:
  base(int n){
    num = n;
    cout<<"base initialized"<<endl;
  }
  void show_num(){
    cout<<"num : "<<num<<endl;
  }

};

class derived: public base
```

```
{
   string name;
public:
   derived(string _name, int n):base(n){
      name = _name;
      cout<<"derived initialized"<<endl;
   }
   void show_name(){
      cout<<"name : "<<name<<endl;
   }
};

int main()
{
   derived A("Divyam", 58);
   A.show_name();
   A.show_num();
}
```

Output:



```
base initialized
derived initialized
name : Divyam
num : 58
```

Conclusion: The fundamentals of Inheritance were understood and implemented in the c++ programs above.

Nitesh Naik

(Subject Faculty)