Experiment 10

AIM:

Consider the following class definition

```
class Father {
    protected: int age;
    public: Father(int x) {
        age = x;
    }
    virtual void iam() {
        cout << "I am the father, my age is " << age << endl;
    }
};
```

Derive 2 classes son and daughter from the above class. For each define suitable constructors for these classes. Write a program to create instances of all the 3 classes and call iam() for them. Use pointer to base class to store store objects of derived class and call iam() through it.

Theory:

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

A virtual function is a function in a base class that is declared using the keyword virtual. Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function.

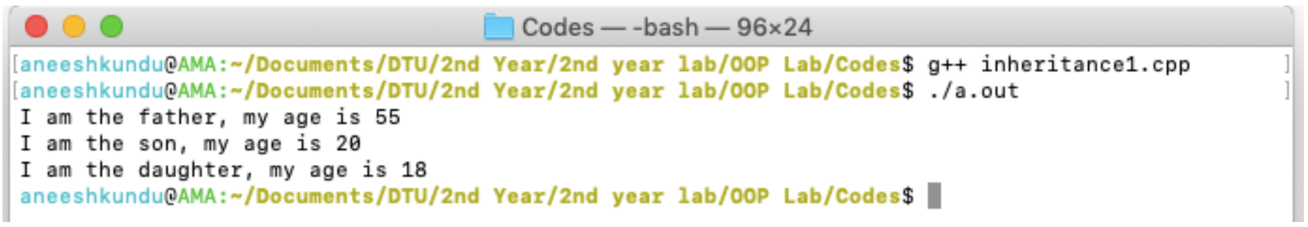
In inheritance, the properties of existing classes are extended to the new classes. The new classes that can be created from the existing base class are called as derived classes. The inheritance provides the hierarchical organization of classes. It also provides the hierarchical relationship between two objects and indicates the shared properties between them. All derived classes inherit properties from the common base class. Pointers can be declared to the point base or derived class. Pointers to objects of the base class are type compatible with pointers to objects of the derived class. A base class pointer can point to objects of both the base and derived class. In other words, a pointer to the object of the base class can point to the object of the derived class; whereas a pointer to the object of the derived class cannot point to the object of the base class.

Code:

```
# include < iostream >
using namespace std;
class Father {
protected:
    int age;
public:
    Father(int x) {
        age = x;
    }
          virtual void iam() {
              cout << "I am the father, my age is " << age << endl;
          }
      };
      class Son: public Father {
```

```cpp
    public:
        Son(int x): Father(x) {}
        void iam() {
            cout << "I am the son, my age is " << age << endl;
        }
    };
    class Daughter: public Father {
    public:
        Daughter(int x): Father(x) {}
        void iam() {
            cout << "I am the daughter, my age is " << age << endl;
        }
    };
    int main() {
        Father * ptr;
        ptr = new Father(55);
        ptr - > iam();
        delete ptr;
        ptr = new Son(20);
        ptr - > iam();
        delete ptr;
        ptr = new Daughter(18);
        ptr - > iam();
        return 0;
    }
```

Output:



```
aneeshkundu@AMA:~/Documents/DTU/2nd Year/2nd year lab/OOP Lab/Codes$ g++ inheritance1.cpp
aneeshkundu@AMA:~/Documents/DTU/2nd Year/2nd year lab/OOP Lab/Codes$ ./a.out
I am the father, my age is 55
I am the son, my age is 20
I am the daughter, my age is 18
aneeshkundu@AMA:~/Documents/DTU/2nd Year/2nd year lab/OOP Lab/Codes$
```

Discussion:
As it can be seen base class pointer can point to derived class objects, and only those methods which are present in the base class can be accessed. To facilitate late binding virtual functions are used.