Program – 3.3

Aim: To write C++ programs that show how different types of inheritance work:

1. **Single Inheritance** – One class inherits from one base class.
2. **Multiple Inheritance** – One class inherits from two or more base classes.
3. **Multilevel Inheritance** – A class inherits from a class that is already derived.
4. **Hierarchical Inheritance** – Multiple classes inherit from the same base class.
5. **Hybrid Inheritance** – A mix of two or more types of inheritance

Program:

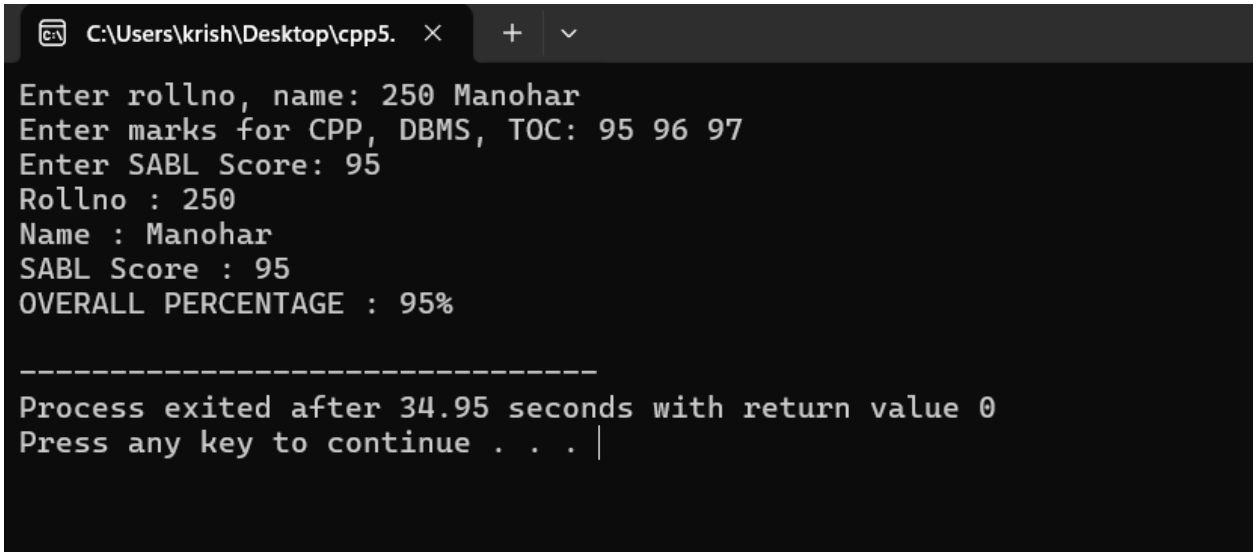
1) Single Inheritance:

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

// Base class 'person'
class person {
protected:
   string name;  // Protected member: accessible in derived class
   char gender;
   int age;

   // Protected member function to input personal details
   void get() {
      cout << "Enter name, age, gender: ";
      cin >> name >> age >> gender;
   }
};

// Derived class 'student' inherits from 'person'
```

```
class student : public person {
private:
    int pinno;     // Private data members specific to student
    float per;

public:
    // Member function to initialize student-specific details
    void getst() {
        pinno = 434;
        per = 95;

        // Call base class's protected get() function to input personal details
        get();
    }

    // Member function to display all details
    void display() {
        cout << "Name       : " << name << endl;
        cout << "Age        : " << age << endl;
        cout << "Gender     : " << gender << endl;
        cout << "Pin no     : " << pinno << endl;
        cout << "Percentage : " << per << endl;
    }
};

int main() {
    student s;      // Create object of derived class
    s.getst();      // Input student and personal details
    s.display();    // Display all details
```

```
    return 0;
}
```

OUTPUT:

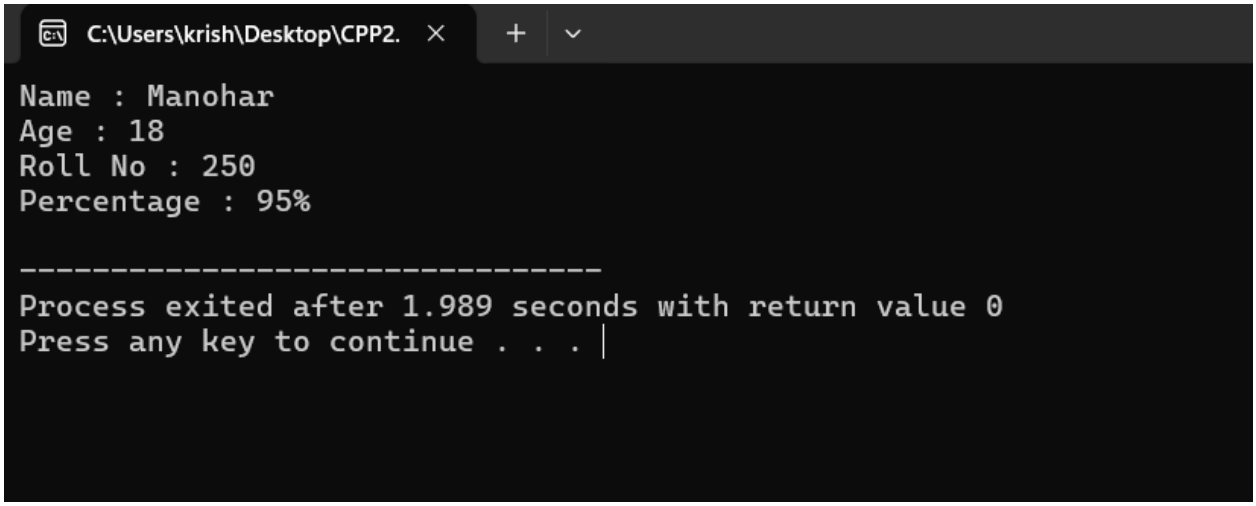


2) Multiple Inheritance:

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

// Base class 1: Personal details
class Person {
public:
    string name;
    int age;

    void get(string n, int a) {
        name = n;
        age = a;
    }
};

// Base class 2: Academic details
class Student {
```

```
public:
    int rollNo;
    float percentage;

    void getst(int r, float p) {
        rollNo = r;
        percentage = p;
    }
};

// Derived class: Inherits from both Person and Student
class Result : public Person, public Student {
public:
    void display() {
        cout << "Name        : " << name << endl;
        cout << "Age         : " << age << endl;
        cout << "Roll No     : " << rollNo << endl;
        cout << "Percentage  : " << percentage << "%" << endl;
    }
};
int main() {
    Result r;
    // Set data using base class functions
    r.get("Manoj", 18);
    r.getst(434, 95);
    r.display();              // Display combined result
    return 0;
}
```

OUTPUT:



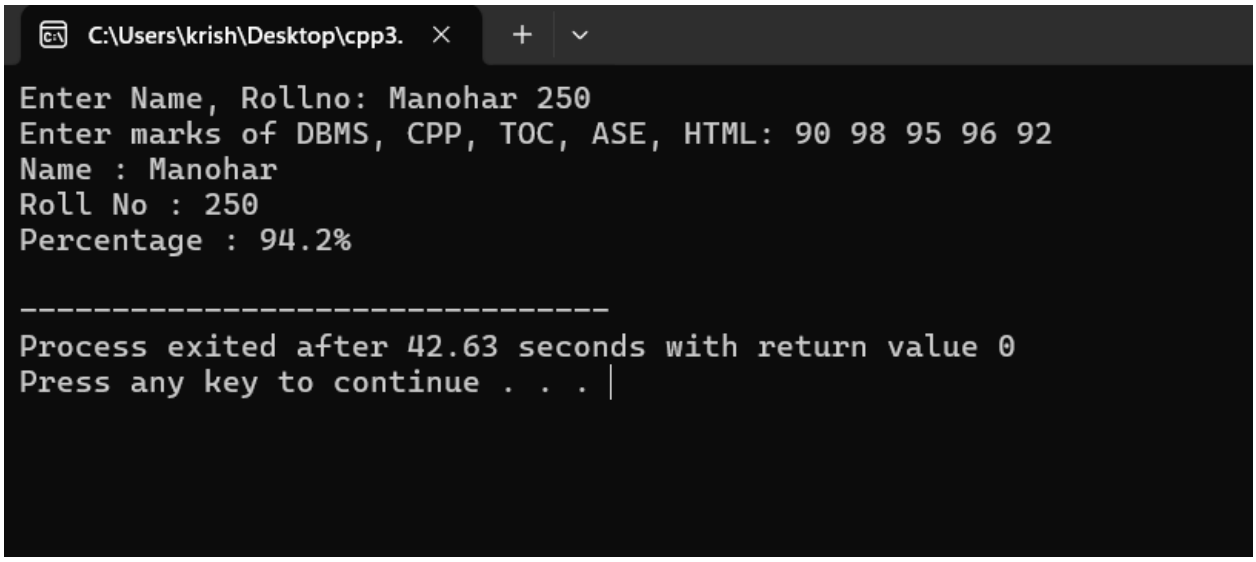
3) Multi-level Inheritance:

```
#include<iostream>
using namespace std;
class student                    // Base class: student
{
protected:
   string name;
   int rollno;

   // Function to input student details
   void getst() {
      cout << "Enter Name, Rollno: ";
      cin >> name >> rollno;
   }

   // Function to display student details
   void showst() {
      cout << "Name     : " << name << endl;
      cout << "Roll No  : " << rollno << endl;
   }
```

```
};

// Derived class: marks inherits from student
class marks : public student {
protected:
    int DBMS, CPP, TOC, ASE, HTML;

    // Function to input marks
    void getm() {
        getst();                // Call base class function to get student details
        cout << "Enter marks of DBMS, CPP, TOC, ASE, HTML: ";
        cin >> DBMS >> CPP >> TOC >> ASE >> HTML;
    }
};

// Derived class: percentage inherits from marks (multilevel)
class percentage : public marks {
public:
    float per;
    // Function to calculate and display percentage
    void showp() {
        getm();  // Call marks class function to get marks and student details
        per = (float)(DBMS + CPP + TOC + ASE + HTML) / 5;  // Calculate average
        showst();  // Display student details
        cout << "Percentage : " << per << "%" << endl;
    }
};

int main() {
```

```
    percentage p;              // Create object of most derived class
    p.showp();                 // Call function to input data and display result
    return 0;
}
```

OUTPUT:



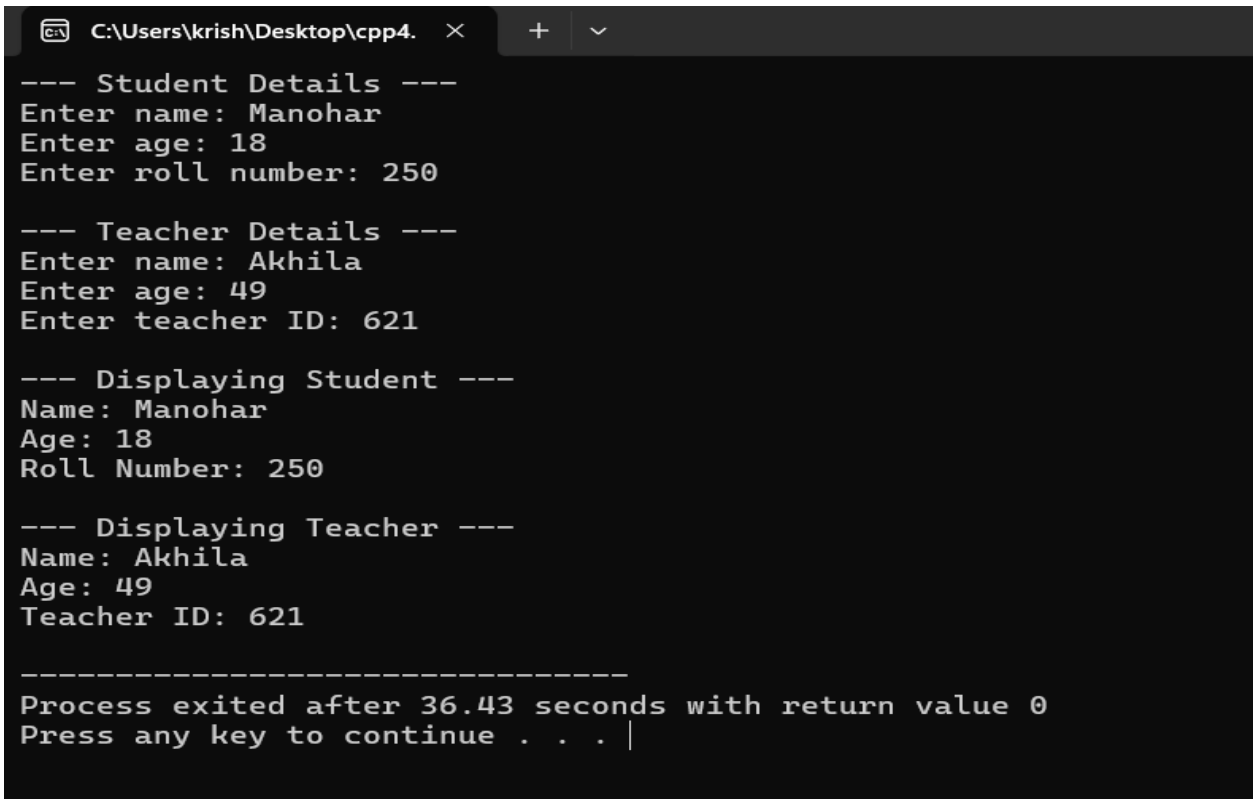
4) Hierarchical Inheritance:

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

class Person {
public:
    string name;
    int age;

    void input() {
        cout << "Enter name: ";
        cin >> name;
        cout << "Enter age: ";
        cin >> age;
```

```
    }

    void display() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};

class Student : public Person {
public:
    int roll;

    void inputStudent() {
        input();
        cout << "Enter roll number: ";
        cin >> roll;
    }

    void displayStudent() {
        display();
        cout << "Roll Number: " << roll << endl;
    }
};

class Teacher : public Person {
public:
    int id;

    void inputTeacher() {
```

```
        input();
        cout << "Enter teacher ID: ";
        cin >> id;
    }

    void displayTeacher() {
        display();
        cout << "Teacher ID: " << id << endl;
    }
};

int main() {
    Student s;
    Teacher t;

    cout << "--- Student Details ---" << endl;
    s.inputStudent();

    cout << "\n--- Teacher Details ---" << endl;
    t.inputTeacher();

    cout << "\n--- Displaying Student ---" << endl;
    s.displayStudent();

    cout << "\n--- Displaying Teacher ---" << endl;
    t.displayTeacher();

    return 0;
```

OUTPUT:



5) Hybrid Inheritance:

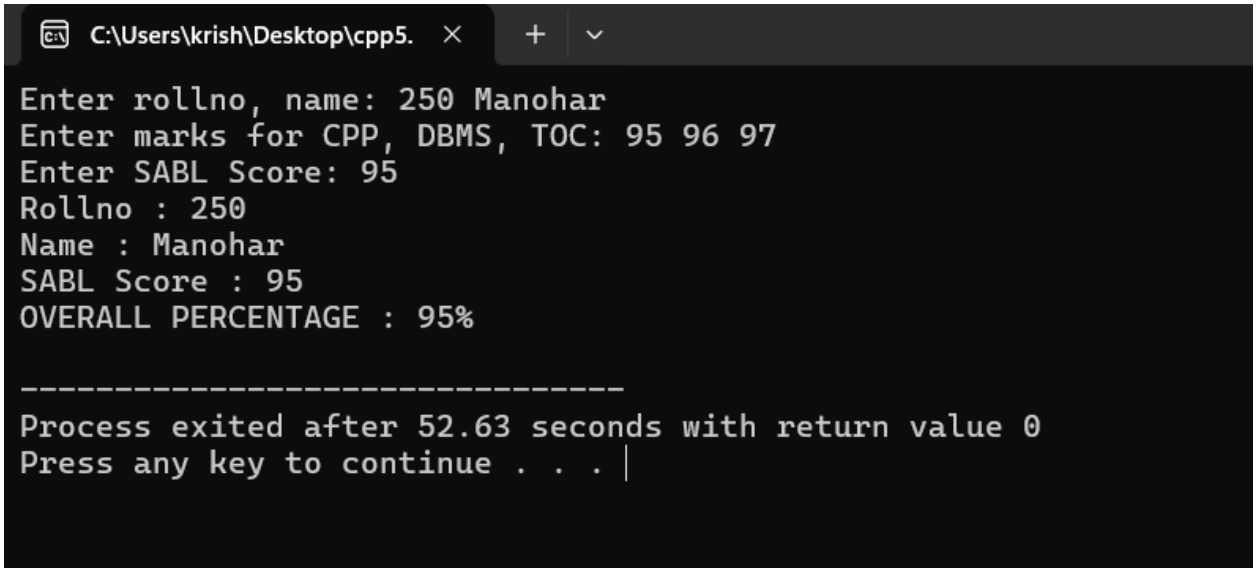
```
#include<iostream>
using namespace std;

// Base class: student
class student {
protected:
    int rollno;
    string name;

public:
    // Function to input student details
    void getst() {
        cout << "Enter rollno, name: ";
        cin >> rollno >> name;
```

```
    }

    // Function to display student details
    void showst() {
        cout << "Rollno : " << rollno << endl;
        cout << "Name   : " << name << endl;
    }
};

// Derived class: marks inherits from student (Single Inheritance)
class marks : public student {
protected:
    int CPP, DBMS, TOC;

    // Function to input marks
    void getm() {
        getst();  // Call base class function
        cout << "Enter marks for CPP, DBMS, TOC: ";
        cin >> CPP >> DBMS >> TOC;
    }
};

// Independent class: SABL (not related to student)
class SABL {
protected:
    int sablscore;

public:
    // Function to input SABL score
```

```
    void getsabl() {
        cout << "Enter SABL Score: ";
        cin >> sablscore;
    }

    // Function to display SABL score
    void showsabl() {
        cout << "SABL Score : " << sablscore << endl;
    }
};

// Derived class: percentage inherits from both marks and SABL (Hybrid Inheritance)
class percentage : public marks, public SABL {
public:
    // Function to gather all inputs
    void get() {
        getm();       // From marks → student
        getsabl();    // From SABL
    }

    // Function to calculate and display overall percentage
    void calculateper() {
        int per = (CPP + DBMS + TOC + sablscore) / 4;
        showst();     // From student
        showsabl();   // From SABL
        cout << "OVERALL PERCENTAGE : " << per << "%" << endl;
    }
};
```

```
int main() {
    percentage p;     // Create object of most derived class
    p.get();          // Input all data
    p.calculateper(); // Display result
    return 0;
}
```

OUTPUT:



Program – 3.4

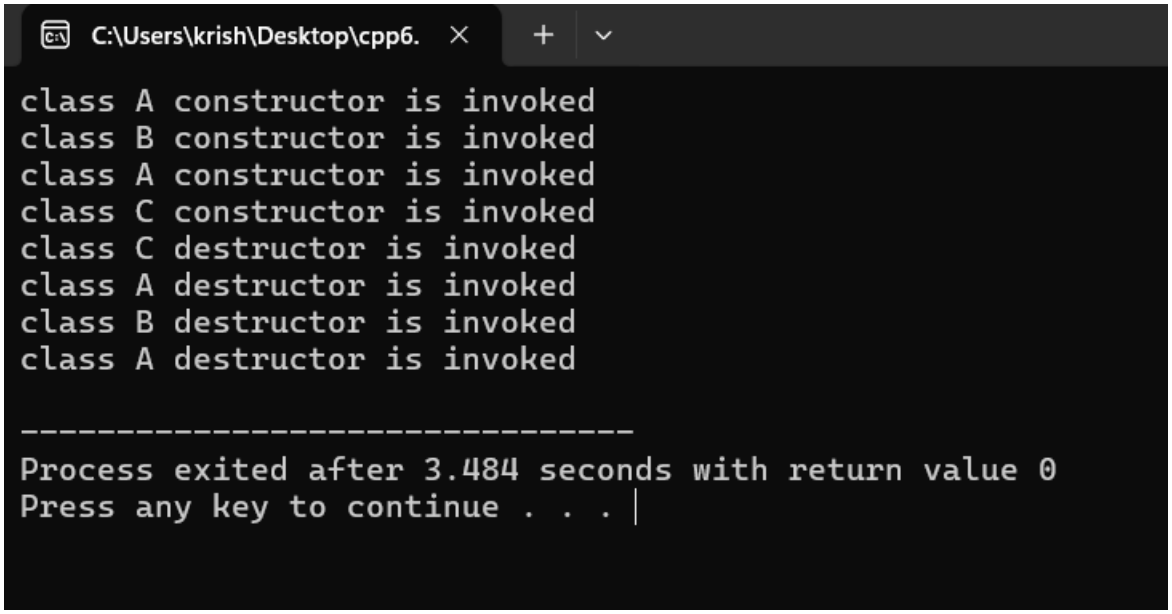
Aim: To demonstrate the order of execution of constructors and destructors in hierarchical inheritance using C++

Program:

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

// Base class A
class A {
public:
   // Constructor of class A
   A() {
      cout << "class A constructor is invoked" << endl;
   }

   // Destructor of class A
   ~A() {
      cout << "class A destructor is invoked" << endl;
   }
};

// Derived class B inherits from A
class B : public A {
public:
   // Constructor of class B
   B() {
      cout << "class B constructor is invoked" << endl;
   }
```

```
    // Destructor of class B
    ~B() {
        cout << "class B destructor is invoked" << endl;
    }
};

// Derived class C inherits from A
class C : public A {
public:
    // Constructor of class C
    C() {
        cout << "class C constructor is invoked" << endl;
    }

    // Destructor of class C
    ~C() {
        cout << "class C destructor is invoked" << endl;
    }
};

int main() {
    B b;  // Object of class B created
    C c;  // Object of class C created

    // When objects go out of scope, destructors are automatically called
    return 0;
}
```

OUTPUT:


C:\Users\krish\Desktop\cpp6.
class A constructor is invoked
class B constructor is invoked
class A constructor is invoked
class C constructor is invoked
class C destructor is invoked
class A destructor is invoked
class B destructor is invoked
class A destructor is invoked

Process exited after 3.484 seconds with return value 0
Press any key to continue . . .

Program – 3.5

Aim: To understand and demonstrate key object-oriented programming concepts in C++—specifically the use of:

- **Object as a class member** (composition),
- **Pointer to a class** (dynamic access to members),
- **this pointer** (reference to the calling object),
- **Virtual base class** (resolving ambiguity in multiple inheritance),

Program:

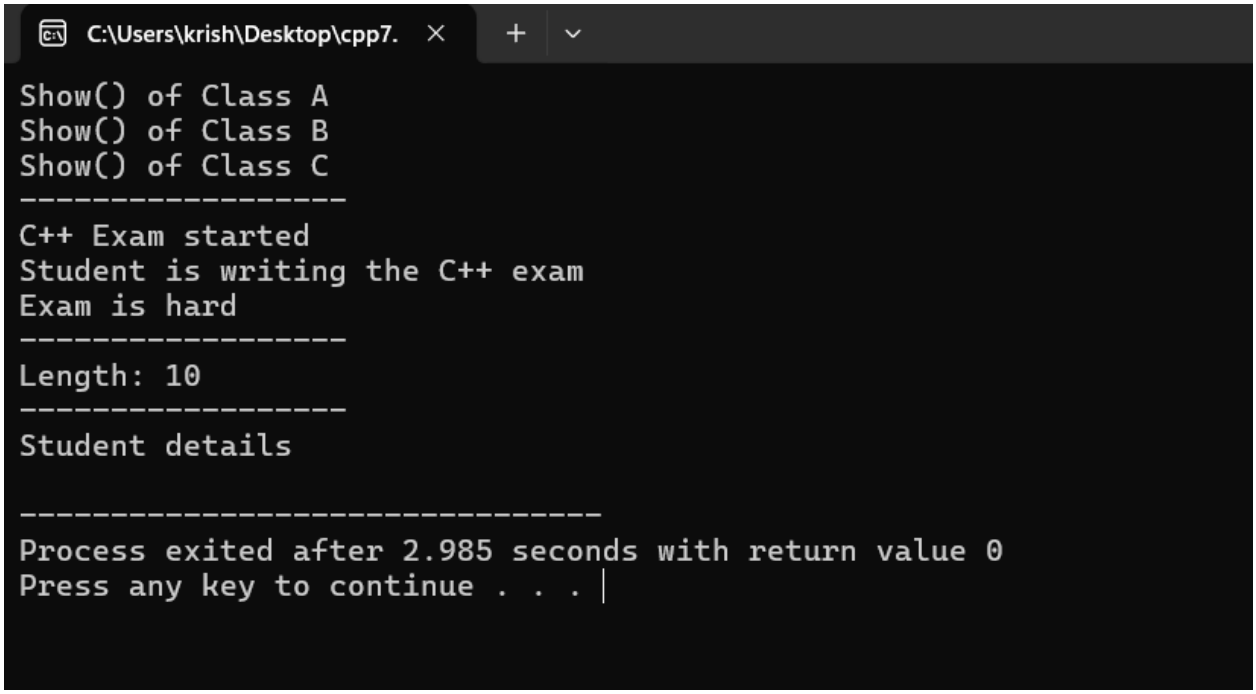
```
#include <iostream>
using namespace std;

class A {
public:
   void show() {
      cout << "Show() of Class A\n";
   }
};

class B : virtual public A {
public:
   void showB() {
      cout << "Show() of Class B\n";
   }
};

class C : virtual public A {
```

```
public:
    void showC() {
        cout << "Show() of Class C\n";
    }
};

class D : public B, public C {
    // inherits A, B, C
};

// Object as class member (StudentExam has a CppExam)
class CppExam {
public:
    void begin() {
        cout << "C++ Exam started\n";
    }
};

class StudentExam {
    CppExam exam;
public:
    void attend() {
        exam.begin();
        cout << "Student is writing the C++ exam\n";
        cout << "Exam is hard\n";
    }
};
```

```
// Class using 'this' pointer
class Box {
    int length;
public:
    Box(int length) {
        this->length = length;
    }
    void display() {
        cout << "Length: " << this->length << "\n";
    }
};

// Pointer to class
class Student {
public:
    void show() {
        cout << "Student details\n";
    }
};

int main() {
    D obj;
    obj.show();
    obj.showB();
    obj.showC();
```

```
    cout << "------------\n";

    StudentExam se;
    se.attend();

    cout << "------------\n";

    Box b(10);
    b.display();

    cout << "------------\n";

    Student s;
    Student* ptr = &s;
    ptr->show();

    return 0;
}
```

OUTPUT:


C:\Users\krish\Desktop\cpp7.
Show() of Class A
Show() of Class B
Show() of Class C
C++ Exam started
Student is writing the C++ exam
Exam is hard
Length: 10
Student details
Process exited after 2.985 seconds with return value 0
Press any key to continue . . .

Program – 3.6

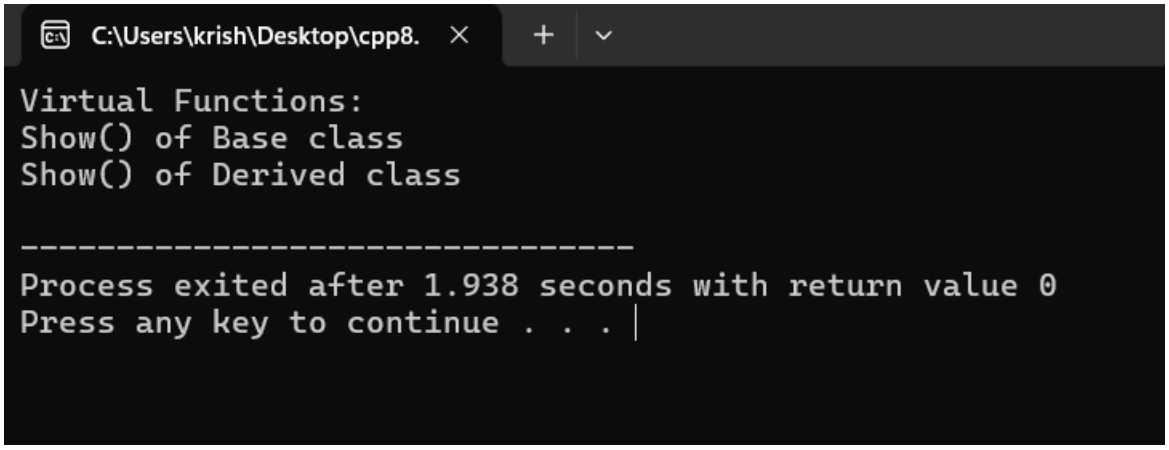
Aim: To demonstrate the concept of **virtual functions** in C++, enabling **runtime polymorphism** through base class pointers calling overridden functions in derived classes.

Program:

```
#include<iostream>
using namespace std;
class Base {
        public:
                virtual void show()
                {
                        cout<<"Virtual Functions:"<<endl;
                        cout<<"Show() of Base class"<<endl;
                }
};
class Derived : public Base
{
        public:
                void show()
                {
                        cout<<"Show() of Derived class"<<endl;
                }
};
int main()
{
        Base *bptr,b;
```

```
        Derived d;
        bptr = &b;
        bptr -> show();
        bptr = &d;
        bptr -> show();
}
```

OUTPUT:



Program – 3.7

Aim: To implement a pure virtual function in an abstract class and use it to calculate the area of different shapes such as square, circle, and triangle by overriding the function in derived classes.

Program:

```
#include<iostream>

using namespace std;


// Abstract base class 'shapes' with a pure virtual function 'area'

class shapes {

   public:

      virtual void area() = 0;      // Pure virtual function to be overridden by derived classes

};


// Derived class 'Rectangle' inherits from 'shapes'

class Rectangle : public shapes {

   public:

      int length, breadth;

      Rectangle() {

         length = 20;

         breadth = 30;

      }

      void area() {

         cout << "Pure Virtual Functions of Different shapes:" << endl;

         cout << "Area of Rectangle: " << length * breadth << endl;

      }
```

```
};
// Derived class 'Triangle' inherits from 'shapes'

class Triangle : public shapes {
    public:
        int base, height;
        Triangle() {
            base = 5;
            height = 8;
        }
        void area() {
            cout << "Area of Triangle: " << 0.5 * base * height << endl;
        }
};

// Derived class 'Circle' inherits from 'shapes'
class Circle : public shapes {
    public:
        int radius;
        float pi;
        Circle() {
            radius = 5;
            pi = 3.14; // Use float for accurate decimal representation
        }
        void area() {
            cout << "Area of Circle: " << pi * radius * radius << endl;
        }
};
```

```
// Derived class 'Square' inherits from 'shapes'
class Square : public shapes {
    public:
        int side;
        Square() {
            side = 6;
        }
        void area() {
            cout << "Area of Square: " << side * side << endl;
        }
};

// Main function demonstrating runtime polymorphism
int main() {
    shapes *s; // Pointer to base class

    Rectangle r;
    s = &r;
    s->area(); // Calls Rectangle's area()

    Triangle t;
    s = &t;
    s->area(); // Calls Triangle's area()

    Circle c;
    s = &c;
    s->area(); // Calls Circle's area()

    Square x;
```

```
    s = &x;
    s->area(); // Calls Square's area()

    return 0;
}
```

OUTPUT: