Lab Number:	6
Student Name:	Sakshi Vadiraj Kaveri
Roll No :	33

Title:

1. To perform Multiple Inheritance in C++. Create a student class representing student roll number, name and branch and an exam class (derived class of student) representing the scores of the student in various subjects (maths, physics and chemistry) and sports class representing the score in sports. The sports and exam class isinherited by a result class which adds the exam marks and sports score to generate the final result.

2. To perform Hierarchical Inheritance in C++. Create an Employee class with attributes EmpID and EmpSalary. Also create necessary methods/constructors to accept these values from the user. Create classes permenantEmployee and TemporaryEmployee which will be derived classes of Employee. Mention hike attribute in these derived classes and calculate the total salary using generate_salary() method for respective types of employees. Objects of the derived classes should be created and salaries for the permanent and temporary employees should be calculated and displayed on the screen.

Learning Objective:

- Students will be able to perform multiple inheritance using C++.

Learning Outcome:

- Understanding the inheritance concept and reusability of the code.

Course Outcome:

ECL304.2	Comprehend building blocks of OOPs language, inheritance, package and interfaces

Theory:

- Explain in details about inheritance, its types, syntaxes and block diagrams.

Inheritance is a mechanism by which the attributes or methods of a certain class can be inherited by another class or in other words one class can acquire data from other class. The class whose variables and methods are acquired or inherited is called base/parent/super class and the class acquiring the data or inheriting the data is known as derived/child/sub class.

In C++ there are 5 types of inheritance, they are:-

1) Single inheritance

2) Multi-level inheritance

3) Multiple inheritance

4) Hierarchical inheritance

5) Hybrid inheritance

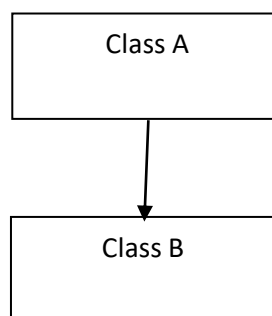a) <u>SINGLE INHERITANCE:</u>

Syntax:

```
//Base Class

class A

{

 public void fooA()

 {

 }

};


//Derived Class

class B :public A

{

 public void fooB()

 {

 }

};
```

Here a child class inherits a parent class.

Block diagram:

```
+-------------------+
|      Class A      |
|                   |
+-------------------+
          |
          v
+-------------------+
|      Class B      |
|                   |
+-------------------+
```

b) Multi-level inheritance:

Syntax:

//Base Class

class A

{

 public void fu1()

 {

 }

};

//Derived Class
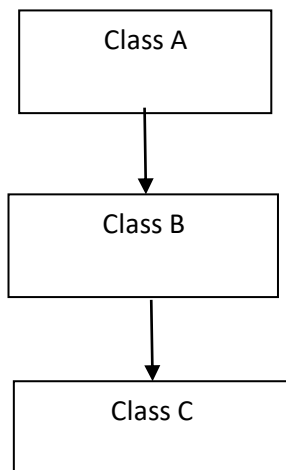
class B :public A

{

 public void fu2()

 {

 }

};

//Derived Class

class C :public B

{

 public void fu3()

 {

 }

};

Block diagram:

```
┌─────────────────┐
│                 │
│    Class A      │
│                 │
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│                 │
│    Class B      │
│                 │
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│                 │
│    Class C      │
│                 │
└─────────────────┘
```

In this inheritance, a derived class is created from another derived class.

Class C inherits properties from class B and as class B has inherited properties from Class A, class C has also access to the data from Class A.

c) Multiple inheritance:

Syntax:

//Base Class

class A

{

 public void fun1()

 {

 }

};

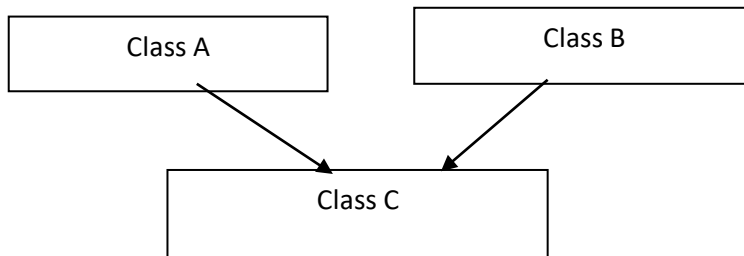//Base Class

class B

{

 public void fun2()

 {

 }

};

//Derived Class

```
class C :public A, public B
{
 public void fun3()
 {
 }
};
```

Block Diagram:

In this inheritance, a derived class is created from more than one base class. In the given example, class c inherits the properties and behaviour of class B and class A at same level. So, here A and Class B both are the parent classes for Class C.

d) Hierarchical inheritance:

Syntax:

```
//Base Class
class A
{
 public void fun1()
 {
 }
};


//Derived Class
class B : public A
{
```

```
 public void fun2()

 {

 }

};


//Derived Class

class C :public A

{

 public void fun3()

 {

 }

};


//Derived Class

class D :public B

{

 public void fooD()

 {

 }

};


//Derived Class

class E :public B

{

 public void fun4()

 {

 }

};
```
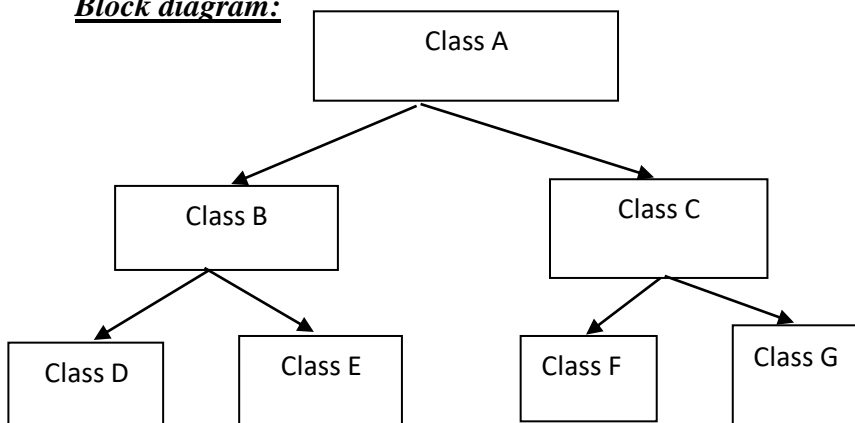
```
//Derived Class

class F :public C

{

 public void fun5()

 {

 }

};


//Derived Class

class G :public C

{

 public void fun6()

 {

 }

};
```

Block diagram:

In this inheritance, more than one derived classes are created from a single base class and futher child classes act as parent classes for more than one child classes.

In the given example, class A has two childs class B and class C. Further, class B and class C both are having two childs - class D and E; class F and G respectively.
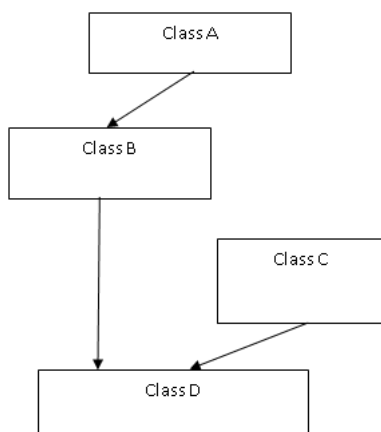
e) Hybrid inheritance:

Syntax:

class A

{

 ....

};

class B : public A

{

 ....

} ;

class C

{

 ....

};

 class D : public B, public C

{

 ....

};

Block diagram:

This is combination of more than one inheritance. Hence, it may be a combination of Multilevel and Multiple inheritance or Hierarchical and Multilevel inheritance or Hierarchical and Multipath inheritance or Hierarchical, Multilevel and Multiple inheritance.

Faculty: Ms. Deepali Kayande

Q1. /* To perform Multiple Inheritance in C++.

Create a student class representing student roll number, name and branch and an exam class (derived class of student)

 representing the scores of the student in various subjects (maths, physics and chemistry) and sports class representing the score in sports.

The sports and exam class is inherited by a result class which adds the exam marks and sports score to generate the final result. */

ALGORITHM:

1) Create class student and declare string name, int roll_no, string branch variables. Use a default constructor, student(), and in it take input from user.

2) Derive a class exam from student and declare variables of marks of subjects mathematics, physics and chemistry.

3) Create class sports and declare and take input from user for Sports_marks.

4) Create class result and calculate total by total=marks of all subjects + sports marks and display the output.

5) Create object of derived class result ob1 and the constructors of the base classes are called automatically.

Program:

```cpp
#include<iostream>

using namespace std;

class student{

        public:
        string name;
        int roll_no;
        string branch;


                student()
                {
                        cout<<"Enter the name of the student: ";
                        cin>>name;
```

```cpp
                cout<<"Enter the roll-number of the student: ";

                cin>>roll_no;

                cout<<"Enter the branch of the student: ";

                cin>>branch;

        }
};
class exam: public student{


        public:

        int maths_marks;

        int phy_marks;

        int chem_marks;


                exam()
                {
                cout<<"Enter the marks obtained by the student in Mathematics out of 100: ";

                cin>>maths_marks;

                cout<<"Enter the marks obtained by the student in Physics out of 100: ";

                cin>>phy_marks;

                cout<<"Enter the marks obtained by the student in Chemistry out of 100: ";

                cin>>chem_marks;

                }
};
class sports{


        public:
```

```cpp
        int sports_marks;


                sports()
                {
                        cout<<"Enter the marks obtained by the student in sports: ";
                        cin>>sports_marks;

                }
};
class result: public exam, public sports{


        public:
        int total;


                void display()
                {
                        total = maths_marks + phy_marks + chem_marks + sports_marks;
                        cout<<"\n\nThe name of the student is "<<name;
                        cout<<"\n\nThe roll-no:. of the student is "<<roll_no;
                        cout<<"\n\nThe branch of the student is "<<branch;
                        cout<<"\n\nThe marks obtained by the student in Mathematics are: "<<maths_marks;
                        cout<<"\n\nThe marks obtained by the student in Physics are: "<<phy_marks;
                        cout<<"\n\nThe marks obtained by the student in Chemistry are: "<<chem_marks;
                        cout<<"\n\nThe marks obtained by the student in Sports are: "<<sports_marks;
                        cout<<"\n\nThe total marks obtained by the student are: "<<total;

                }
```

```
};


int main()

{

        result ob1;

        ob1.display();

        return 0;

}
```
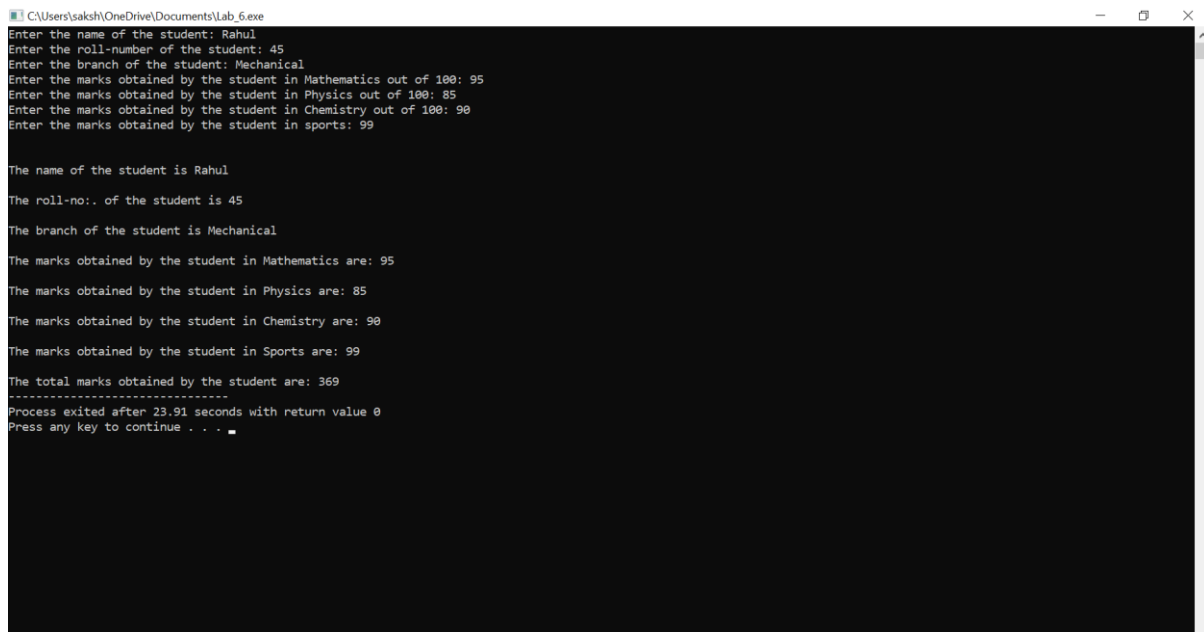
Input given:

Name of the student- Rahul, roll-no: 45, branch- Mechanical, Maths marks- 95, Physics marks-85, chemistry marks-90, sports marks- 99

Output screenshot:

Q2) /* To perform Hierarchical Inheritance in C++.

 Create an Employee class with attributes EmpID and EmpSalary.Also create necessary methods/constructors to accept these values from the user.

Create classes permenantEmployee and TemporaryEmployee which will be derived classes of Employee.

Mention hike attribute in these derived classes and calculate the total salary using generate_salary() method for respective types of employees.

Objects of the derived classes should be created and salaries for the permanent and temporary employees should be calculated and

displayed on the screen. */

Algorithm:

1) Create a class employee and declare int EmpID, int EmpSalary.

2) Derive PermanentEmployee and TemporaryEmployee classes from class Employee declare float hike=0.1*EmpSalary for permanent employee and float hike= 0.05*EmpSalary for temporrary employee.

3) Create objects of the derived classes and display the output.

Program:

```cpp
#include<iostream>

using namespace std;

class Employee{

        public:

        int EmpID;

        int EmpSalary;

        Employee()

        {

                cout<<"Enter the ID of the Employee- ";

                cin>>EmpID;

                cout<<"Enter the current salary of the Employee- ";

                cin>>EmpSalary;

        }

};

class PermanentEmployee: public Employee{

        float hike = (0.1*EmpSalary);

        public:

                PermanentEmployee(){
```

```cpp
                cout<<"\n\nHi this is section of Permanent Employee "<<endl;
        }

                void generate_salary()

                {

                        int total_sal= EmpSalary + hike;

                        cout<<"The total Salary of the Employee is
"<<total_sal<<endl<<endl;

                }
                };
class TemporaryEmployee: public Employee{

        float hike=(0.05*EmpSalary);

        public:

                TemporaryEmployee(){

                        cout<<"\n\nHi this is section of Temporary Employee "<<endl;}

                        void generate_salary()

                        {

                                int total_sal= EmpSalary + hike;

                                cout<<"The total Salary of the Employee is
"<<total_sal<<endl;

                }
                };


int main()

{

        PermanentEmployee ob1;

        ob1.generate_salary();

        TemporaryEmployee ob2;

        ob2.generate_salary();

        return 0;
```
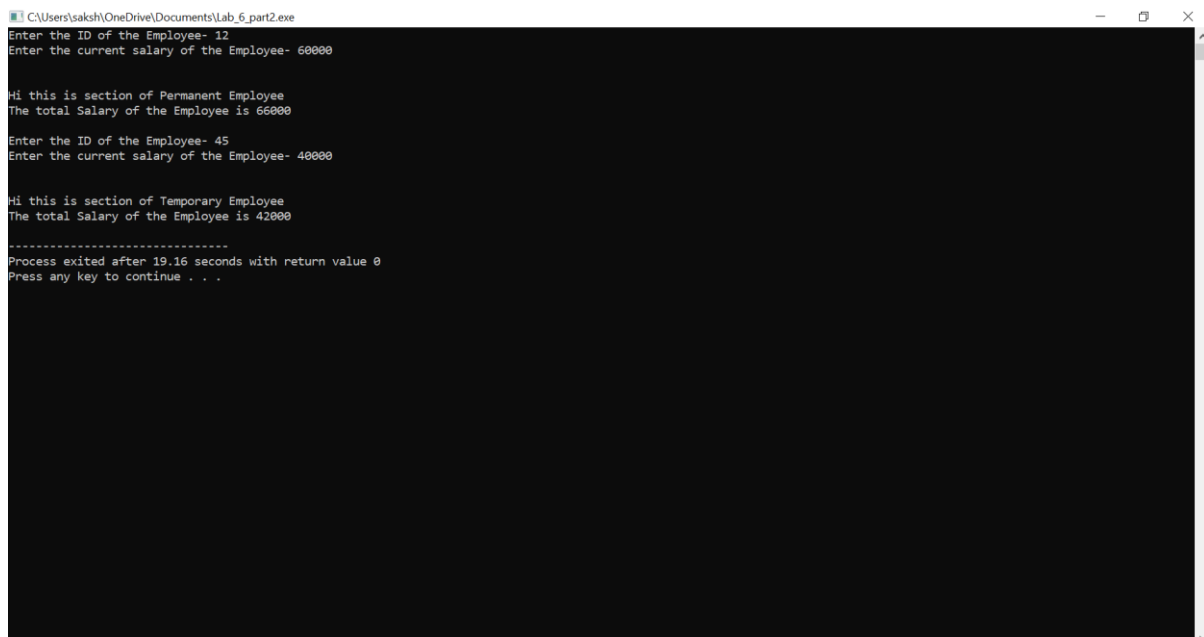
}

Input given:

 employee id for permanent employee- 12, employee salary= 60,000

employee id for temporary employee- 45, employee salary=40,000

Output screenshot:

```
C:\Users\saksh\OneDrive\Documents\Lab_6_part2.exe                                                    —  □  ×
Enter the ID of the Employee- 12
Enter the current salary of the Employee- 60000


Hi this is section of Permanent Employee
The total Salary of the Employee is 66000

Enter the ID of the Employee- 45
Enter the current salary of the Employee- 40000


Hi this is section of Temporary Employee
The total Salary of the Employee is 42000

--------------------------------
Process exited after 19.16 seconds with return value 0
Press any key to continue . . .
```