|  |  |  |
| --- | --- | --- |
| **Rubrics for Object Oriented Programming Lab** | | |
| **Lab #:** | **07** | |
| **Lab Title:** | **Overriding in Inheritance** | |
| **Submitted by:** | | |
| **Name** | | **Registration #** |
| **AMMAR**  **MUHAMMAD KALEEM ULLAH** | | **FA19-BCE-001**  **FA19-BCE-007** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics name & number** | | | **Marks** | | |
| **In-Lab** | | **Post-Lab** |
| **Engineering Knowledge** | ***R2: Use of Engineering Knowledge and follow Experiment Procedures:***  *Ability to follow experimental procedures, control variables, and record procedural steps on lab report.* | |  | | |
| **Problem Analysis** | | ***R5: Data/Evidence Measurements:***  *Ability to record raw data / evidence.* | |  | |
| **Design** | | ***R8: Best Coding Standards:***  *Ability to follow the coding standards and programming practices.* | |  | |
| **Modern Tools Usage** | | ***R9: Understand Tools:*** *Ability to describe and explain the principles behind and applicability of engineering tools.* | |  | |
| **Individual and Teamwork** | | ***R12: Individual Work Contributions:*** *Ability to carry out individual responsibilities.* | |  | |
| ***R13: Management of Team Work:***  *Ability to appreciate, understand and work with multidisciplinary team members.* | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics #** | R2 | R5 | R8 | R9 | R12 | R13 |
| **In –Lab** |  |  |  |  |  |  |
| **Post- Lab** |  |  |  |  |  |  |

**Lab#07**

**Overriding in Inheritance**

1. **Objectives:**

Objective of this lab to get familiar with

* 1. Concept of overriding in inheritance
  2. Importance of overriding

1. **Introduction:**
   1. **Overriding:**

Function overriding is a feature that allows us to have a same function in child class which is already present in the parent class. A child class inherits the data members and member functions of parent class, but when you want to override functionality in the child class then you can use function overriding.

You have learned that a derived class can inherit both **public and protected members** (both variable and functions) from a base class. However, the derived class can also redefine the inherited member function. If the derived class defines a member function with has the same signature (number and type of parameters) as the base class, **then the derived class is overriding the base class's member function**.

* 1. **Overloading v/s** **Overriding:**

**Overloading** occurs when two or more methods in one class have the same method name but different parameters. **Overriding** means having two methods with the same method name and parameters (i.e., method signature). One of the methods is **in the** parent class and the other is **in the** child class.

* 1. **Accessing the overridden function:**

To access the overridden function of the base class from the derived class, scope resolution operator:: is used. For example, If you want to access getData() function of the base class, you can use the following statement in the derived class.



1. **In-Lab Tasks:**
   1. **Task#01:** Write a program that declares two classes. The parent class is called Simple that has two data members num1 and num2to store two numbers. It also has four member functions.

* The add() function adds two numbers and displays the result.
* The sub() function subtracts two numbers and displays the result.
* The mul() function multiplies two numbers and displays the result.
* The div() function divides two numbers and displays the result.

The child class is called Complex that overrides all four functions. Each function in the child class checks the value of data members. It calls the corresponding member function in the parent class if the values are greater than 0. Otherwise, it displays error message.

* **Code:**

/\* Write a program that declares two classes. The parent class is called Simple that has two data members num1 and num2 to store two numbers.

It also has four member functions.

" The add() function adds two numbers and displays the result.

" The sub() function subtracts two numbers and displays the result.

" The mul() function multiplies two numbers and displays the result.

" The div() function divides two numbers and displays the result.

The child class is called Complex that overrides all four functions. Each function in the child class checks the value of data members.

It calls the corresponding member function in the parent class if the values are greater than 0. Otherwise it displays error message.

\*/

#include<iostream>

using namespace std;

class Simple

{

protected:

int num1,num2;

int addition,subtraction,multiplication,division;

public:

void getData()

{

cout<<"\nEnter the Number 1 : ";cin>>num1;

cout<<"\nEnter the Number 2 : ";cin>>num2;

}

void add()

{

addition=num1+num2;

cout<<"The Addition i.e. Num1 + Num2 = "<<addition;

}

void sub()

{

subtraction=num1-num2;

cout<<"\nThe Subtraction i.e. Num1-Num2 = "<<subtraction;

}

void mul()

{

multiplication=num1\*num2;

cout<<"\nThe Multiplication i.e. Num1\*Num2 = "<<multiplication;

}

void div()

{

division=num1/num2;

cout<<"\nThe Division i.e. Num1/Num2 ="<<division;

}

};

class Complex:public Simple

{

public:

void add()

{

Simple::add();

}

void sub()

{

Simple::sub();

}

void mul()

{

if(num1==0||num2==0)

{

cout<<"\nerror !!! ";

}

else

{

Simple::mul();

}

}

void div()

{

if(num1==0||num2==0)

{

cout<<"\nerror !!! ";

}

else

{

Simple::div();

}

}

};

int main()

{

Complex c;

char n,choice;

cout<<"\nFor Addition Enter '+'";

cout<<"\nFor Subtraction Enter '-'";

cout<<"\nFor Multiplication Enter '\*'";

cout<<"\nFor Division Enter '/'";

do

{

c.getData();

cout<<"\nEnter the Operation You Want to Try : ";

cin>>n;

switch (n)

{

case '+':

{

c.add();

break;

}

case '-':

{

c.sub();

break;

}

case '\*':

{

c.mul();

break;

}

case '/':

{

c.div();

break;

}

}

cout<<"\nDo you wanna try this again ? ";

cout<<"\nEnter 'y' for yes and 'n' for no >> ";

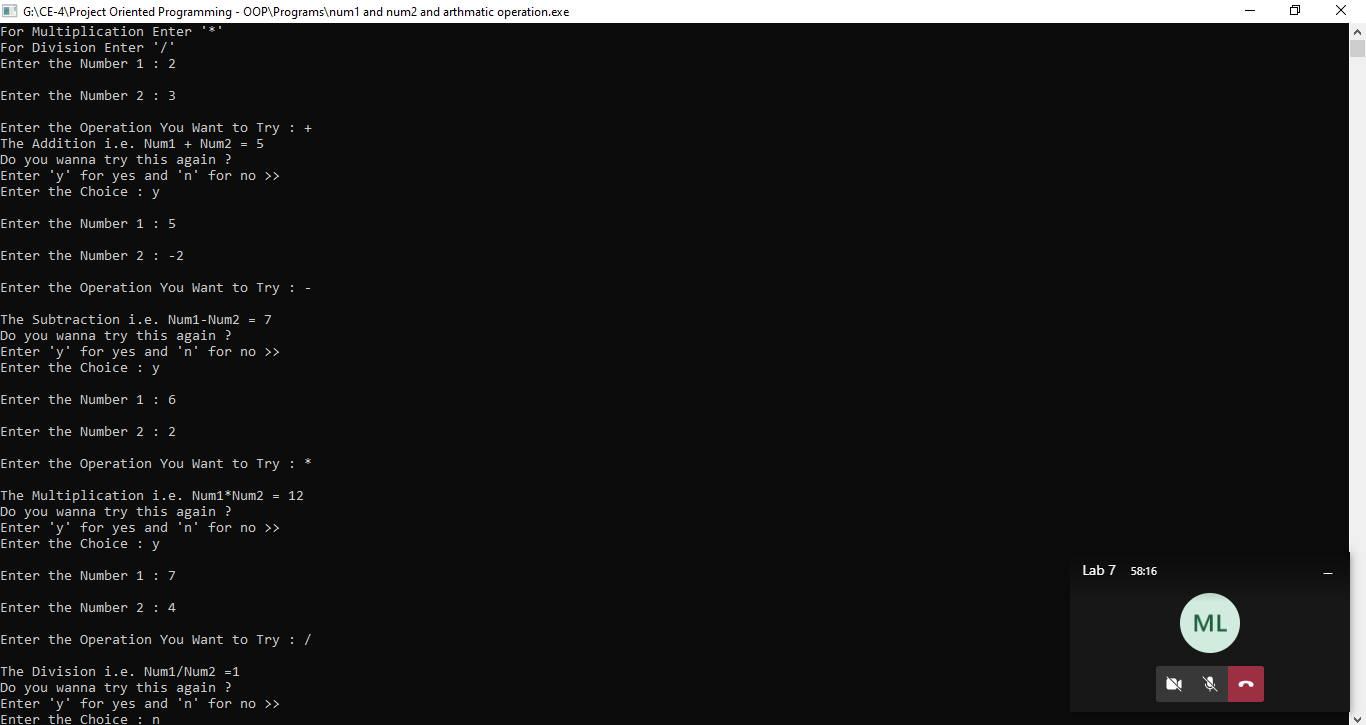
cout<<"\nEnter the Choice : ";cin>>choice;

}while(choice=='y');

return 0;

}

* **Output:**

****

* 1. **Task#02:** Suppose we are developing a program that a car dealership can use to manage its inventory of used cars. The dealership’s inventory includes three types of automobiles: cars, pickup trucks, and sport-utility vehicles (SUVs). Regardless of the type, the dealership keeps the following data about each automobile:

• Make

• Year model

• Mileage

• Price

Each type of vehicle that is kept in inventory has these general characteristics, plus its own specialized characteristics. For cars, the dealership keeps the following additional data:

• Number of doors (2 or 4)

For pickup trucks, the dealership keeps the following additional data

• Drive type (two-wheel drive or four-wheel drive)

And, for SUVs, the dealership keeps the following additional data:

• Passenger capacity

Write a program which has a single base class to keep the record of general information regarding all automobiles further derive three classes for each type of automobile each having its specific characteristic i.e., Number of doors, Drive type and Passenger capacity.

Define appropriate accessor and mutator functions in the base class to get data from user and display the results, override these functions in respective derived classes. In main function, initialize any two of objects from user and one through argument constructor.

Your Output should look like this:

**We have the following car in inventory:**

**2007 BMW with mileage of 50000 miles having 4 doors.**

**Price: $15000.00**

**We have the following truck in inventory:**

**2006 Toyota with mileage of 40000 miles and 4WD drive type.**

**Price: $12000.00**

**We have the following SUV in inventory:**

**2005 Volvo with mileage of 30000 miles and 5 passenger capacity.**

**Price: $18000.00**

* **Code:**

#include<iostream>

#include<string>

using namespace std;

class baseData

{

protected:

string companyName;

int yearModal,price,mileAge;

public:

daseData()

{

companyName=" ";

yearModal=0;

mileAge=0;

price=0;

}

void getData()

{

cin.ignore();

cout<<"\nEnter the Company Name : ";

getline(cin,companyName,'\n');

cout<<"\nEnter The Model Year : ";cin>>yearModal;

cout<<"\nEnter the Price of the Vehicle : ";cin>>price;

cout<<"\nEnter the mileAge";cin>>mileAge;

}

void putData()const

{

cout<<endl<<yearModal<<" "<<companyName<<" with mileage of "<<mileAge<<" miles ";

}

};

class car:public baseData

{

private:

int door;

public:

void getData()

{

cout<<"\nEnter the Car Data : \n";

baseData::getData();

cout<<"Number of doors (2 or 4) >> ";cin>>door;

}

void putData()

{

cout<<"\nWe have the following car in inventory:";

baseData::putData();

cout<<"having "<<door<<"doors.";

cout<<endl<<"Price: $"<<price;

}

};

class pickupTruck:public baseData

{

private:

int drive;

public:

void getData()

{

cout<<"\nEnter the Pickup Truck Data : ";

baseData::getData();

cout<<"Drive type (two-wheel drive or four-wheel drive) >>";

cin>>drive;

}

void putData()

{

cout<<"\nWe have the following truck in inventory:";

cout<<endl;

baseData::putData();

cout<<"and "<<drive<<"WD drive type.";

cout<<endl<<"Price: $"<<price;

}

};

class SUVs:public baseData

{

private:

int pcapacity;

public:

void getData()

{

cout<<"\nEnter the sport-utility vehicles (SUVs) ";

baseData::getData();

cout<<"\nPassenger Capacity >> ";cin>>pcapacity;

}

void putData()

{

cout<<"\nWe have the following SUV in inventory:";

baseData::putData();

cout<<"and "<<pcapacity<<" passenger capacity.";

cout<<endl<<"Price: $"<<price;

}

};

int main()

{

car c;

pickupTruck pt;

SUVs s;

c.getData();

pt.getData();

s.getData();

system("CLS");

c.putData();

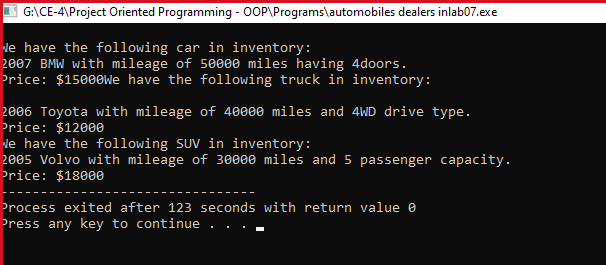
pt.putData();

s.putData();

return 0;

}

* **Output:**

****

1. **Post-Lab Tasks:**
   1. **Task# 01:** An electricity board charges the following rates to domestic users to discourage large consumption of energy.

* For the first 100 units − 50 P per unit
* Beyond 100 units − 60 P per unit

If the total cost is more than Rs.250.00 then an additional surcharge of 15% is added on the difference. Define a class Electricity in which the function Bill computes the cost. Define a derived class More Electricity and override Bill to add the surcharge.

* **Code:**

#include<iostream>

using namespace std;

class Electricity

{

protected:

int cost,units;

public:

Electricity()

{

cost=0;

units=0;

}

void getData()

{

cout<<"\nEnter the Unit Consumed : ";cin>>units;

}

void bill()

{

if(units>100)

{

cost = 50 + (units-100)\*60/100;

}

else

{

cost = 50\*units/100; //if less than 100

}

}

void display()const

{

cout<<"Your Bill = "<<cost; //printing bills

}

~Electricity()

{

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

}

};

class moreElectricity:public Electricity

{

public:

moreElectricity()

{

Electricity::getData();

}

void bill()

{

Electricity::bill();

if(cost>250)

{

cost = cost + (15./100)\*(cost-250);

}

}

void display()

{

cout<<"Your Bill = "<<cost; //printing bills

}

};

int main()

{

moreElectricity b;

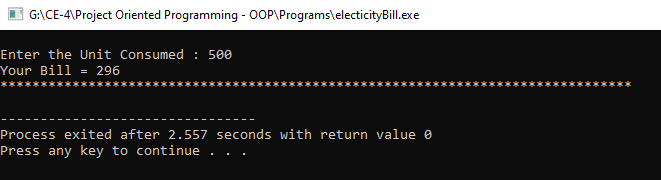
b.bill();

b.display();

return 0;

}

* **Output:**

****

* 1. **Task# 02:** Create a class 2D having the x and y coordinates of an object. Derive a class 3D form from 2D as a base class with an additional z coordinate, override x and y coordinates of 2D class using setdata() function. Get the coordinates for two specific points and compute the distance between them using formula d = ((x2 - x1)2 + (y2 - y1)2 + (z2 - z1)2)1/2
* **Code:**

#include<iostream>

#include<cmath>

using namespace std;

class twoD

{

protected:

float x1,y1,x2,y2;

public:

twoD()

{

x1=0;

y2=0;

x2=0;

y2=0;

}

void setData()

{

cout<<"\nEnter the First Point Coordinates >> ";

cout<<"\nEnter the x Coordinates : ";cin>>x1;

cout<<"\nEnter the y Coordinates : ";cin>>y1;

cout<<"\nEnter the 2nd Point Coordinates >> ";

cout<<"\nEnter the x Coordinates : ";cin>>x2;

cout<<"\nEnter the y Coordinates : ";cin>>y2;

}

~twoD()

{

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

}

};

class threeD:public twoD

{

private:

float z1,z2,d;

public:

threeD()

{

z1=z2=0;

}

void setData()

{

twoD::setData();

cout<<"\nEnter the first points Z Coordinates >> ";

cout<<"\nEnter the z Coordinates : ";cin>>z1;

cout<<"\nEnter the 2nd points Z Coordinates >> ";

cout<<"\nEnter the z Coordinates : ";cin>>z2;

}

void compute()

{

//d = ((x2 - x1)2 + (y2 - y1)2 + (z2 - z1)2)1/2;

float d1=(x2 - x1)\*(x2 - x1) + (y2 - y1)\*(y2 - y1) + (z2 - z1)\*(z2 - z1);

d=sqrt (d1);

cout<<"\nDistance : "<<d;

}

};

int main()

{

threeD d;

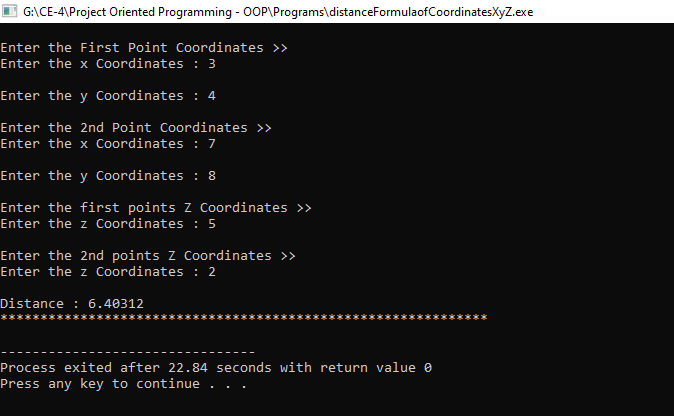
d.setData();

d.compute();

return 0;

}

* **Output:**

****

1. **Conclusion:**

After completing this lab, we are able to:

* Perform the overriding of member functions
* Learn how to access the overridden function in the base class from the derived class.
* Practice different problems to clear and use practically the concept of overriding in inheritance.