|  |  |  |
| --- | --- | --- |
| **Rubrics for Object Oriented Programming Lab** | | |
| **Lab #:** | **08** | |
| **Lab Title:** | **Class Hierarchiers (Single And Multi Level)** | |
| **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 08**

**Class Hierarchies(Multi-Level & Multiple Inheritance)**

1. **Objectives**

To familiarize the students with class hierarchies, multiple inheritance and multilevel inheritance .

1. **Outcome**

After this lab the students should be able to differentiate between public and private

inheritance. They should learn the role of constructor and destructors in derived classes. They should be able to implement multilevel inheritance and multiple inheritance.

1. **Introduction** 
   1. **Public and private inheritance**

The public keyword in the inheritance syntax means that publicly accessible members inherited from the base class stay publicly accessible in the derived class. But for private keyword in the inheritance syntax means that accessible members inherited from the base become private members of derived class.

* 1. **Constructors and Destructors in Derived Classes**

Derived classes do not inherit constructors or destructors from their base classes, but they do call the constructor and destructor of base classes. When an object of a derived class is created, the base class’s constructor is executed first, followed by the derived class’s constructor. When an object of a derived class is destroyed, its destructor is called first, then that of the base class.

* 1. **Multiple inheritance and Multi-level inheritance**

Classes can be derived from other classes that are already derived from other classes. This creates multi-level Inheritance

**Syntax:**

**class A**

**{ public: int x;**

**};**

**class B : public A**

**{ public:**

**int y;**

**};**

**class C : public B { };**

A Class can be derived from more than one base class. This is called *Multiple inheritance.*

**Syntax:**

**class A**

**{ }; class B**

**{ };**

**class C: public A, public B**

**{ };**

**}**

1. **Lab Tasks** 
   1. Create a class Person having *name, age* and *gender* as its data members. Create another class Department which has *DepartmentName* and *ProgramName* as it data members. Derive a class Student from class Person and class Department which has student *ID*,*grade* and *number of courses* as its member variables.
      1. Write set and get functions to enter and display the data members.
      2. Write main function to implement these classes. Enter the student data to show multiple inheritance.

**Code:**

#include <iostream>

using namespace std;

class Person

{

protected:

string Name;

int Age;

char Gen;

public:

void GetData();

void Display();

};

void Person::GetData()

{

cout<<"\n\t Please Enter Your Name : ";getline(cin,Name);

cout<<"\n\t Please Enter Your Age : ";cin>>Age;

cout<<"\n\t Please Enter Your Gender : ";cin>>Gen;

}

void Person::Display()

{

cout<<"\n\t Name : "<<Name;

cout<<"\n\t Age : "<<Age;

cout<<"\n\t Gender :"<<Gen;

}

class Department

{

protected:

string DepName;

string ProgName;

public:

void GetData();

void Display();

};

void Department::GetData()

{

cin.ignore();

cout<<"\n\t Please Enter Your Department Name : ";getline(cin,DepName,'\n');

cout<<"\n\t Please Enter Your Program Name : ";getline(cin,ProgName);

}

void Department::Display()

{

cout<<"\n\t Department Name : "<<DepName;

cout<<"\n\t Program Name : "<<ProgName;

}

class Student:public Person, public Department

{

private:

int ID;

char Grade;

int Courses;

public:

void GetData();

void Display();

};

void Student::GetData()

{

Person::GetData();

Department::GetData();

cout<<"\n\t Enter You ID :";cin>>ID;

cout<<"\n\t Enter Your Grade :";cin>>Grade;

cout<<"\n\t Enter Number Of Courses : ";cin>>Courses;

}

void Student::Display()

{

Person::Display();

Department::Display();

cout<<"\n\t ID : "<<ID;

cout<<"\n\t Grade : "<<Grade;

cout<<"\n\t Total Courses : "<<Courses;

}

int main()

{

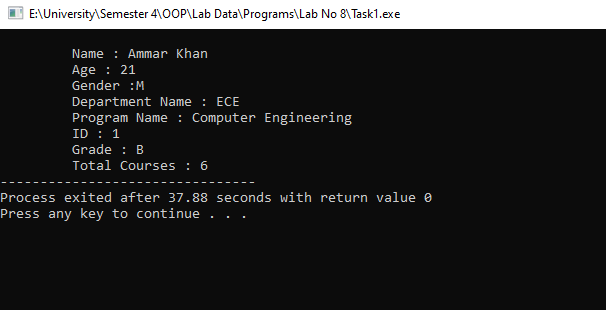
Student Alpha;

Alpha.GetData();

system("CLS");

Alpha.Display();

}



* 1. Design a class named Employee. The class should keep the following information in

•Employee name

•Employee number

•Hire date

Write one or more constructors and the appropriate accessor and mutator functions for the class. Next, write a class named ProductionWorker that is derived from the Employee class. The ProductionWorker class should have member variables to hold the following information:

•Shift (an integer)

•Hourly pay rate (a double )

The workday is divided into two shifts: day and night. The shift variable will hold an integer value representing the shift that the employee works. The day shift is shift 1, and the night shift is shift 2. For night shift hourly rate will be doubled, write down a function called salary to calculate the total salary of worker. Write one or more constructors and the appropriate accessor and mutator functions for the class. Demonstrate the classes by writing a program that uses a ProductionWorker object, ask the user for how many workers he wants to store the data and then display the recorded data.

**Code:**

#include<iostream>

#include<string>

#include<limits>

using namespace std;

class Employee

{

protected:

string Name;

int Num;

string Date; //Input Format(D/M/Y)

public:

Employee() ; // Constructor

void GetData(); //Input

};

Employee::Employee()

{

Name="\*\*\*";Num=0;Date="\*\*\*";

}

void Employee::GetData()

{

cin.ignore( numeric\_limits <streamsize> ::max(), '\n' );

cout<<"\n\n\t Please Enter The Employee Name : ";getline(cin,Name);

cout<<"\n\t Please Enter The Employee Number : ";cin>>Num;

cin.ignore( numeric\_limits <streamsize> ::max(), '\n' );

cout<<"\n\t Please Enter The Joing Date Of Employee (D/M/Y) : ";getline(cin,Date);

}

class Production\_Worker:public Employee

{

private:

int Shift;

double Pay\_Rates;

int Hours;

double Salary;

public:

Production\_Worker(); //Constructor

void Get\_Data(); // Input

int Calculate\_Salary(); // Salary Calculation

void Display() ; //Output

};

Production\_Worker::Production\_Worker()

{

Shift=0;Pay\_Rates=0;Salary=0;Hours=0;

}

void Production\_Worker::Get\_Data()

{

Employee::GetData();

cout<<"\n\t Please Enter The Shift Number 1(Day) or 2(Night) : ";cin>>Shift;

cout<<"\n\t Please Enter The Pay\_Rate of Emplyee Mr."<<Name<<" : ";cin>>Pay\_Rates;

cout<<"\n\t Please Enter The Hours Employee Worked For : ";cin>>Hours;

}

int Production\_Worker::Calculate\_Salary()

{

if(Shift==1)

{

Salary=Pay\_Rates\*Hours;

}

else

{

Salary=(Pay\_Rates + Pay\_Rates/2) \* Hours;

}

return(Salary);

}

void Production\_Worker::Display()

{

cout<<"\n\t The Salary Of Mr."<<Name<<" Is : "<<Calculate\_Salary();

}

int main()

{

cout<<"\n\t \*\*\*\*\*\*\*\*\*\*\* Please Press Enter To Start The Program \*\*\*\*\*\*\*\*\*\*\* ";

Production\_Worker Alpha[5]; // Objects

for(int i=0 ;i<5;i++) // Inputer

{

Alpha[i].Get\_Data();

system("CLS");

}

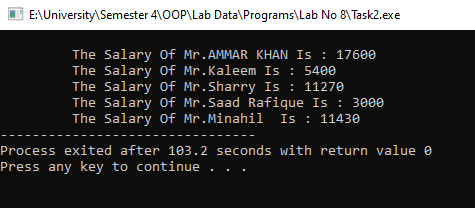
for(int i=0 ;i<5;i++) // Output

{

Alpha[i].Display();

}

}



* 1. **Consider the following classes**

Create a class Date having day, month & year as its data members**.** Create another class called Time with its data members as hours, minutes & seconds. Write down the following functions for both classes:

1. void display(); // to displays the data
2. get() function // to accesses the data members
3. void set(); // to sets the values of data members

Define a class DateandTime from above two classes which displays both date and time.

1. Define an instance object of class DateTime called Watch.
2. Write a main () function that would initialize the values through the constructor functions, and then allows them to be reset through the set () functions. Be sure and display the results following the constructor before you use the set functions.
3. Through the use of the display () function, the time and date are to be displayed. Note that the display () functions in all three classes need to be defined, as well as the constructor and all the access functions.

**Code:**

#include <iostream>

using namespace std;

class Date

{

protected:

int Day;

int Month;

int Year;

public:

Date();

void Display();

void Set();

//void Get();

};

Date::Date()

{Day=0;Month=0;Year=0;}

void Date::Set()

{

cout<<"\n\t Please Enter Day : ";cin>>Day;

cout<<"\n\t Please Enter Month : ";cin>>Month;

cout<<"\n\t Please Enter Year : ";cin>>Year;

}

void Date::Display()

{

cout<<"\n\t Entered Date : "<<Day<<" / "<<Month<<" / "<<Year<<endl;

}

class Time

{

protected:

int H;

int M;

int S;

public:

Time();

void Display();

void Set();

//void Get();

};

Time::Time()

{H=0;M=0;S=0;}

void Time::Set()

{

cout<<"\n\t Please Enter Hours : ";cin>>H;

cout<<"\n\t Please Enter Minutes : ";cin>>M;

cout<<"\n\t Please Enter Seconds : ";cin>>S;

}

void Time::Display()

{

cout<<"\n\t Entered Time : "<<H<<" : "<<M<<" : "<<S<<endl;

}

class Date\_Time: public Date, public Time

{

public:

void Display();

void Set();

};

void Date\_Time::Display()

{

cout<<"\n\t Here I'm In Date\_Time Class : "<<endl;

Date::Display();

Time::Display();

}

void Date\_Time::Set()

{

Date::Set();

Time::Set();

}

int main()

{

Date\_Time Watch;

cout<<"\n\t Initial Values : \n ";

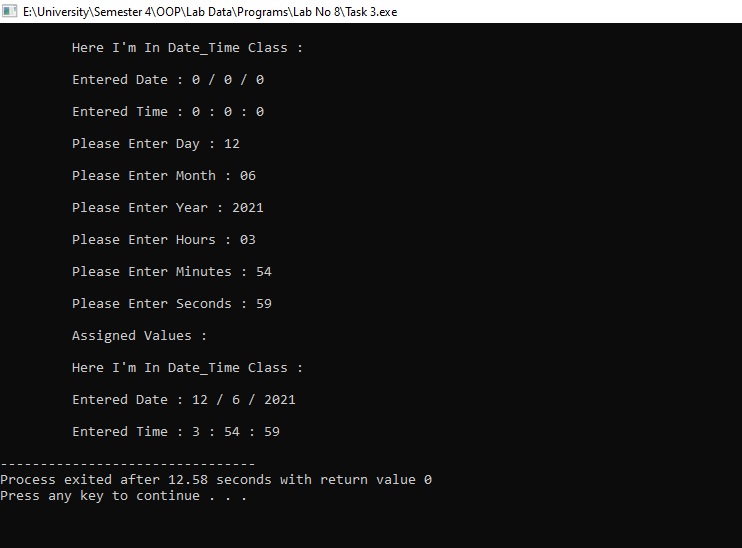
Watch.Display();

Watch.Set();

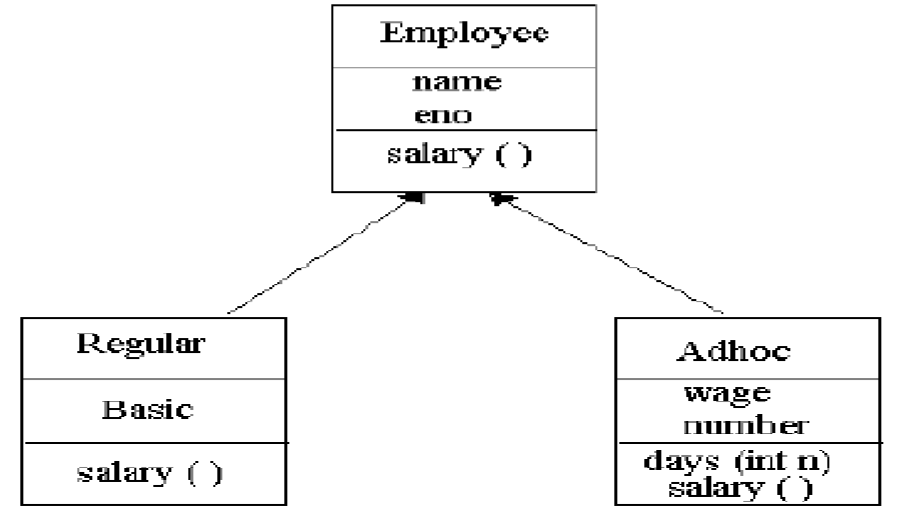
cout<<"\n\t Assigned Values : \n ";

Watch.Display();

}



1. **Home Tasks** 
   1. An organization has two types of employees: regular and adhoc. Regular employees get a salary which is basic + DA + HRA where DA is 10% of basic and HRA is 30% of basic.Adhoc employees are daily wagers who get a salary which is equal to Number \* Wage.
      1. Define the classes shown in the following class hierarchy diagram:



* + 1. Define the constructors. When a regular employee is created, basic must be a parameter.

When adhoc employee is created wage must be a parameter.

* + 1. Define the destructors.
    2. Define the member functions for each class. The member function days ( ) updates number of the Adhoc employee.
    3. Write a test program to test the classes.

**Code:**

#include <iostream>

#include <string>

using namespace std;

class Employee

{

public:

Employee(string name, int id\_);

protected:

string name;

int id;

};

Employee::Employee(string name\_, int id\_) : name(name\_), id(id\_)

{}

class Regular : public Employee

{

public:

Regular(string name\_, int id\_, double salary\_);

void Display();

private:

double salary;

double DA = 0.1;

double HRA = 0.3;

};

Regular::Regular(string name\_, int id\_, double salary\_): Employee(name\_, id\_), salary(salary\_)

{

name=name\_;

id=id\_;

salary=salary\_;

}

void Regular::Display()

{

cout << "\n\t Name is: " << name << endl;

cout << "\n\t Employee ID is: " << id << endl;

cout << "\n\t Employee Salary Is: $" << (salary + salary\*DA + salary\*HRA) << endl;

}

class Adhoc : public Employee

{

public:

Adhoc(string name\_, int id\_, double wages\_,int days\_);

int days;

void Display();

private:

double wages;

};

Adhoc::Adhoc(string name\_, int id\_, double wages\_,int days\_) : Employee(name\_, id\_), wages(wages\_)

{

name =name\_;

id=id\_;

wages=wages\_;

days=days\_;

}

void Adhoc::Display()

{

cout << "\n\t Name is: " << name << endl;

cout << "\n\t Employee ID is: " << id << endl;

cout << "\n\t Employee Salary Is: $" << wages\*days << endl;

}

int main()

{

// Regular Employee

string Name1;

int Id1,Salary1;

cin.ignore();

cout<<"\n\t Plaese Enter Name : ";getline(cin,Name1,'\n');

cout<<"\n\t Plaese Enter "<<Name1<<" Id Number : ";cin>>Id1;

cout<<"\n\t Plaese Enter "<<Name1<<" Basic Salary : ";cin>>Salary1;

Regular Reg(Name1, Id1,Salary1);

system("CLS");

// Daily Wages Emplyee

string Name2;

int Id2,Salary2,Days2;

cin.ignore();

cout<<"\n\t Plaese Enter Name : ";getline(cin,Name2,'\n');

cout<<"\n\t Plaese Enter "<<Name2<<" Id Number : ";cin>>Id2;

cout<<"\n\t Plaese Enter "<<Name2<<" Basic Salary : ";cin>>Salary2;

cout << "\n\t Enter How Many days " << Name2 << " Worked : ";cin>>Days2;

Adhoc Wag(Name2, Id2, Salary2,Days2);

system("CLS");

// Display

cout<<"\n\t Regular Emplpyee : "<<endl;

Reg.Display();

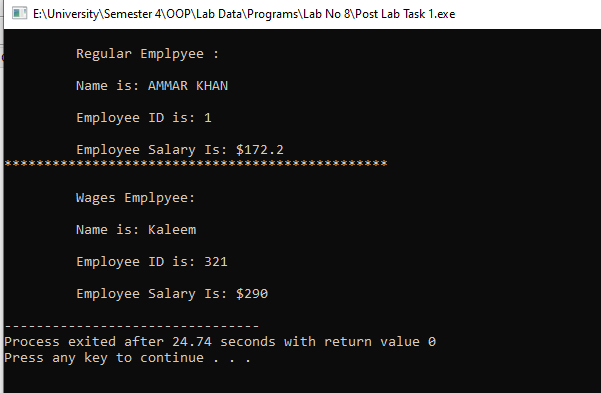
cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout<<"\n\t Wages Emplpyee: "<<endl;

Wag.Display();

return 0;

}



* 1. Write a class LocalPhone that contains an attribute *phone* to store a local telephone number. The class contains member functions to input and display phone number. Write a child class NatPhone for national phone numbers that inherits LocPhone class. It additionally contains an attribute to store *city code*. It also contains member functions to input and show the city code. Write another class IntPhone for international phone numbers that inherit NatPhone class. It additionally contains an attribute to store *country code*. It also contains member functions to input and show the country code. Test these classes from main() by creating objects of derived classes and testing functions in a way that clear concept of multi-level Inheritance.

**Code:**

#include <iostream>

using namespace std;

class LocalPhone //Class 1

{

protected:

string NumL;

public:

LocalPhone();

void Input();

void Display();

};

LocalPhone::LocalPhone()

{

NumL="0";

}

void LocalPhone::Input()

{

cout<<"\n\t Please Enter Phone Number : ";getline(cin,NumL);

}

void LocalPhone::Display()

{

cout<<"\n\t Phone Number : "<<NumL<<endl;

}

class NatPhone : public LocalPhone //Class 2

{

protected:

string City\_Code;

public:

NatPhone();

void Input();

void Display();

};

NatPhone::NatPhone()

{

City\_Code="0";

}

void NatPhone::Input()

{

cout<<"\n\t Please Enter City Code : ";getline(cin,City\_Code);

LocalPhone::Input();// Input Phone Number

}

void NatPhone::Display()

{

cout<<"\n\t City Code : "<<City\_Code;

LocalPhone::Display();// Phone Number Display

}

class IntPhone : public NatPhone

{

private:

string Con\_Code;

public:

IntPhone();

void Input();

void Display();

};

IntPhone::IntPhone()

{

Con\_Code="0";

}

void IntPhone::Input()

{

cout<<"\n\t Please Enter Country Code : ";getline(cin,Con\_Code);

NatPhone::Input();// Input City Code + Phone Number

}

void IntPhone::Display()

{

cout<<"\n\t Country Code : "<<Con\_Code;

NatPhone::Display();// City Code + Phone Number Display

}

// Driver Code

int main()

{

// Local Phone Number

LocalPhone Alpha1;

cout<<"\n\t Local Phone Detail ";

Alpha1.Input();

// National Phone Number

NatPhone Alpha2;

cout<<"\n\t National Phone Detail ";

Alpha2.Input();

//International Phone Number

IntPhone Alpha3;

cout<<"\n\t International Phone Detail ";

Alpha3.Input();

system("CLS");

// Output Zone

cout<<"\n\t Displaying Local Phone Details \n : ";

Alpha1.Display();

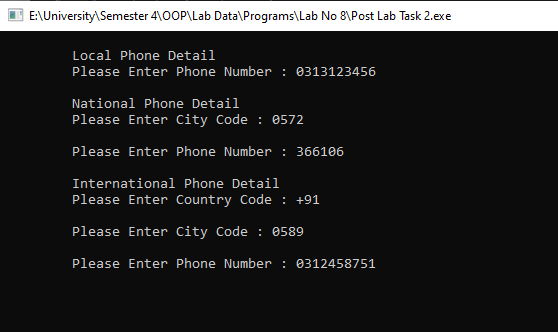
cout<<"\n\t Displaying National Phone Details \n : ";

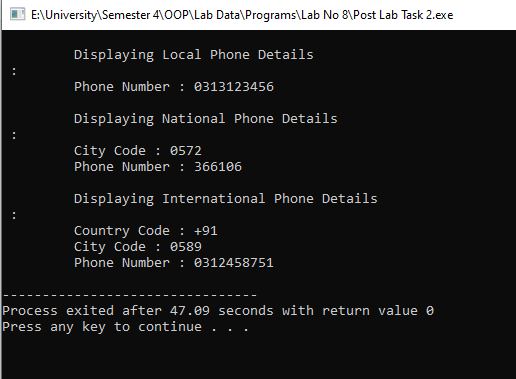
Alpha2.Display();

cout<<"\n\t Displaying International Phone Details \n : ";

Alpha3.Display();

}





Conclusions:

* Inheritance promotes reusability.
* Reusability enhanced reliability.
* As the existing code is reused, it leads to less development and maintenance costs.
* Inheritance makes the sub classes follow a standard interface.
* Inheritance helps to reduce code redundancy and supports code extensibility.