|  |  |  |
| --- | --- | --- |
| **Rubrics for Object Oriented Programming Lab** | | |
| **Lab #:** | **11** | |
| **Lab Title:** | **Polymorphism** | |
| **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#11**

**Polymorphism**

1. **Objectives:**

The objectives of the lab are to get familiar the students with:

* Virtual functions
* Polymorphism
* Late/dynamic binding
* Abstract class and pure virtual functions

1. **Introduction:**
   1. **Polymorphism**

The word polymorphism is a combination of two words poly and morphism. ‘Poly’ means many and ‘morphism’ means form. In object-oriented programming, polymorphism is the ability of objects of different types to respond to functions of the same name. The user does not have to know the exact type of the object in advance. Polymorphism enables to “program in general” rather than “program in specific”. There are derived classes that inherit from the generic base class.

* 1. **Virtual Functions**

A function that appears to exist in some part of a program but does not exist really is called virtual function. Virtual functions are used to implement polymorphism. They enable user to execute completely different functions by same function call. Virtual function allows implementing a different functionality in the derived class.

* With virtual functions, the type of object being pointed to determines which virtual function to execute
* Technically, polymorphism is to declare a pointer of base class and point it towards an object of any derived class, and then using the pointer, calling the functions of the derived class
* To be able to do this, you need to have virtual functions in the base class and override them in derived classes
  1. **Why Virtual Functions are used?**

Suppose you have a number of objects of different classes but you want to put them all in an array and perform a particular operation on them using the same function call.

For example, suppose a graphics program includes several different shapes: a triangle, a ball, a square, and so on.

Each of these classes has a member function **draw()** that causes the object to

be drawn on the screen. Now suppose you plan to make a picture by grouping a number of these elements together and you want to draw the picture in a convenient way. One approach is to create an array that holds pointers to all the different objects in the picture. **The array might be defined like this:**

shape\* ptrarr[100]; // array of 100 pointers to shapes

If you insert pointers to all the shapes into this array, you can then draw an entire picture using a simple loop:

for(int j=0; j<N; j++)

ptrarr[j]->draw();

This is an amazing capability: Completely different functions are executed by the same function call. If the pointer in ptrarr points to a ball, the function that draws a ball is called; if it points to a triangle, the triangle-drawing function is called. This is called **polymorphism,** which means different forms.

* 1. **Early Binding and Late Binding**
* The early binding occurs when everything required to call a function is known at compile time.
* The late binding occurs when some information to call a function is decided at execution time.
  1. **Abstract Class**

An abstract class is a class that can only be a base class for other classes. You cannot create an instance of an abstract class. Instead, it contains operations that are common to all of its derived classes.

* A class becomes abstract, if it contains **even one** “pure virtual function” Pure virtual function:
* Doesn’t and can’t have a function body i.e. its just a function prototype/function declaration
* Is appended by “= 0”
* MUST and must always be overridden in derived classes
* Note: simple virtual function doesn’t necessarily need overriding in derived classes
* Abstract classes are almost always only used as base classes

1. **In-Lab Tasks:**
   1. **Task#01:**

Imagine a publishing company that markets both book and audiocassette versions of its works. As in that exercise, create a class called **publication** that stores the title (a string) and price (type float) of a publication. From this class derive two classes: **book**, which adds a page count (type int); and **tape**, which adds a playing time in minutes (type float). Each of the three classes should have a getdata() function to get its data from the user at the keyboard, and a putdata() function to display the data.

Write a main() program that creates an array of pointers to publication. In a loop, ask the user for data about a particular book or tape, and use new to create an object of type book or tape to hold the data. Put the pointer to the object in the array. When the user has finished entering the data for all books and tapes, display the resulting data for all the books and tapes entered, using a for loop and a single statement such as

pubarr[j]->putdata();

to display the data from each object in the array.

* **Code:**

#include<iostream>

#include<string>

using namespace std;

class Publication

{

private:

string title;

float price;

public:

void getName()

{

cout<<"Enter Title: "; cin>>title;

cout<<"Enter Price: $"; cin>>price;

}

void putName()

{

cout<<"\nTitle: "<<title;

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

}

virtual void getData() = 0;

};

class Book : public Publication

{

private:

int pages;

public:

void getData()

{

Publication::getName();

cout<<"Enter Pages: "; cin>>pages;

}

void putData()

{

Publication::putName();

cout<<", Pages: "<<pages<<endl;

}

};

class Tape : public Publication

{

private:

float minutes;

public:

void getData()

{

Publication::getName();

cout<<"Enter Minutes: "; cin>>minutes;

}

void putData()

{

Publication::putName();

cout<<", Minutes: "<<minutes<<endl;

}

};

int main()

{

Publication\* ptrPub[100];

int n = 0;

char choice;

do

{

cout<<"Book or Tape? (b/t): "; cin>>choice;

if(choice == 'b')

{ ptrPub[n] = new Book; ptrPub[n]->getData(); }

else

{ ptrPub[n] = new Tape; ptrPub[n]->getData(); }

n++;

cout<<"Enter another? (y/n): "; cin>>choice;

} while(choice == 'y');

for(int i=0; i<n; i++)

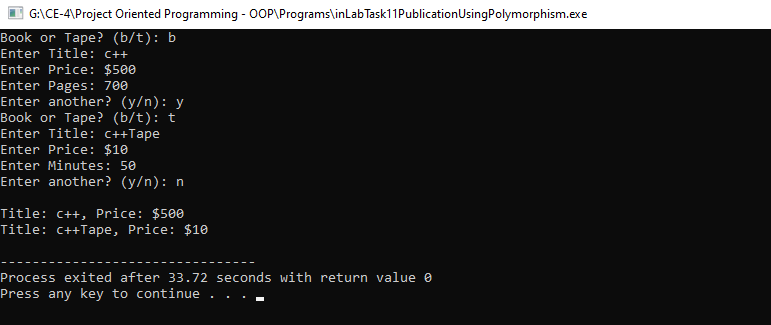
ptrPub[i]->putName();

cout<<endl;

return 0;

}

* **Output:**

****

1. **Post-Lab Tasks:**
   1. **Task#02:**

Develop a simple payroll application. There are three kinds of employees in the system: salaried employee, hourly employee, and commissioned employee. The system takes as input an array containing employee objects, calculates salary polymorphically, and generates report.

Make Employee an abstract class. Declare salary() and display() as pure virtual functions in it. Derive salaried employee (monthly), hourly employee (per hour basis), and commissioned employee (bonus on completing each target) from base class Employee. The display() function should show employee no, employee name, and salary of all employees.

* **Code:**

#include<iostream>

#include<string>

using namespace std;

class Employee

{

protected:

long emp\_num;

string name;

long sal;

public:

virtual void salary () = 0 ;

virtual void display() = 0 ;

};

class salariedemployee : public Employee

{

private:

long emp\_num;

string name;

long sal;

public:

void salary()

{

cout << "Enter employee number : ";

cin >> emp\_num;

cout << "Enter employee name : ";

cin >> name;

cout << "Enter employee salary : ";

cin >> sal;

}

void display()

{

cout << "\n\n";

cout << "==== DETAILED REPORT =====";

cout << "\n\n";

cout << "Employee Number is : " << emp\_num << endl;

cout << "Employee Name is : " << name << endl;

cout << "Employee Salary is : " << sal << endl;

}

};

class hourlyemployee : public Employee

{

protected:

double wage;

double hours;

double sal;

public:

void salary()

{

cout<<"\nEnter the wage : ";

cin>>wage;

cout<<"\nEnter the Hours : ";

cin>>hours;

sal = hours \* wage;

}

void display()

{

cout << "\n Salary is : " << sal;

}

};

class commisonedemployee : public Employee

{

protected:

double grosssales;

double sal;

double comissionrate;

public:

void salary()

{

cout<<"\nEnter the comission rate : ";cin>>comissionrate;

grosssales = sal + comissionrate;

}

void display()

{

cout << "Gross sales : " << grosssales;

}

};

int main ()

{

salariedemployee Sa;

hourlyemployee h;

commisonedemployee c;

Employee \*ptr;

ptr = &Sa;

ptr -> salary();

ptr -> display();

ptr = &h;

ptr -> salary();

ptr -> display();

ptr = &c;

ptr -> salary();

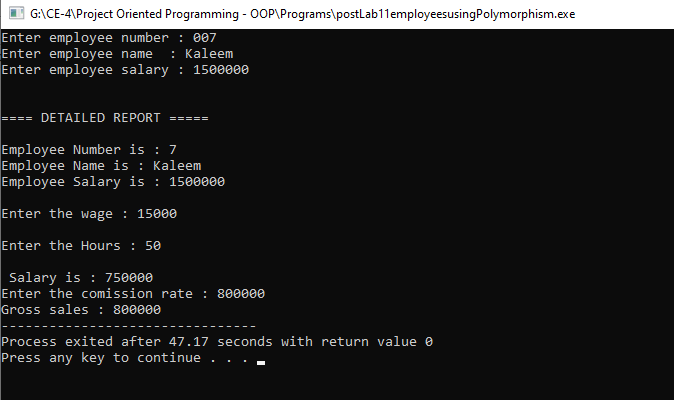
ptr -> display();

//system("pause");

return 0;

}

* **Output:**

****

1. **Conclusion:**

After completing this lab, we are able to do:

* + - Program in general rather than program in specific.
    - Cast derived class objects as objects of the base class.
    - Invoke parent class functions on derived class objects through polymorphism
    - Use virtual functions
    - Create abstract classes with the help of pure virtual functions