|  |
| --- |
| **Computer Engineering Department - ITU** |
| **CE101L: Object Oriented Programming Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 18/05/2022** |
| **Teaching Assistant: Aqsa Khalid** | **Semester: Spring 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 10A. Problem Based Learning through Open Ended Questions**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 100)** | **Scaled to 10** | **Total**  **(out of 10)** |
| Muhammad Abubakar Saif | BSCE21017 |  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Objective**

The objective of this lab is to observe the basic knowledge of programming classes in C++.

## **Equipment and Component**

|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **Conduct of Lab**

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

## **Theory and Background**

**Open-ended problem** is a problem that has several or many correct answers, and several ways to the correct answer(s). The Open-Ended Approach provides students with "experience in finding something new in the process"(Shimada 1997). It is basically facilitating the development of creative problem-solving skills.

Diagram

Description automatically generated

Figure 1: \*What is Open Ended Problem Solving??

**Lab Task**

**Task A: [Marks: 20]**

Write a program to print the sum, difference and product of two complex numbers by creating a class named 'Complex' with separate methods for each operation whose real and imaginary parts are entered by user

|  |
| --- |
| **(Complex.h)**  #ifndef INC\_2022\_SPRING\_TEMPLATE\_CE\_OOP\_ASSIGNMENT\_7\_BSCE21017\_COMPLEX\_H #define INC\_2022\_SPRING\_TEMPLATE\_CE\_OOP\_ASSIGNMENT\_7\_BSCE21017\_COMPLEX\_H #include **<iostream> using namespace** std;  **class** Complex { **private**:  **double** real,imaginary; **public**:  Complex();  Complex(**double** actual, **double** supposed);  **void** setComplex(**int** choice, **double** r = 0, **double** imagine = 0);  **double** getComplex(**int** choice);  **void** printComplexNo();  Complex addTwoComplexNo(Complex second);  Complex subtractTwoComplexNo(Complex second);  Complex multiplyTwoComplexNo(Complex second);  Complex divideTwoComplexNo(Complex second);  Complex getComplexConjugate();  Complex **operator**+ (Complex number);  Complex **operator**- (Complex number);  Complex **operator**\* (Complex number);  Complex **operator**/ (Complex number);  **void operator**= (Complex number);  **friend** ostream &**operator**<<(ostream &output,Complex &test){  test.printComplexNo();  **return** output;  }  **friend** istream &**operator**>>(istream &input,Complex &test){  **double** real1, imaginary1; *// cout<<"Enter Complex Number \n";* cout<<**"Enter Real Part: "**;  cin>>real1;  cout<<**"Enter Imaginary Part: "**;  cin>>imaginary1;  test.setComplex(1 , real1);  test.setComplex(2 ,real1 , imaginary1);  **return** input;  } };   #endif *//INC\_2022\_SPRING\_TEMPLATE\_CE\_OOP\_ASSIGNMENT\_7\_BSCE21017\_COMPLEX\_H*  **(Complex.cpp)**  #include **"Complex.h"** #include **<iostream>  using namespace** std;  **void** Complex::setComplex(**int** choice, **double** r, **double** imagine) { *//set the values of real and imaginary of Complex class* **switch** (choice) { *//switch the cases based upon choice of user* **case** 1:  real = r; *//saves the value of "r" in "real"* **break**; *//breaks the case* **case** 2:  imaginary = imagine; *//saves the value of "imagine" in "imaginary"* **break**; *//breaks the case* }  **return**; }  **double** Complex::getComplex(**int** choice) { *//get the values of real and imaginary of Complex class* **double** value; *//declares double-type variable* **switch** (choice) { *//switch the cases based upon choice of user* **case** 1:  value = real; *//saves the value of "real" in "value"* **break**;*//breaks the case* **case** 2:  value = imaginary; *//saves the value of "imaginary" in "value"* **break**; *//breaks the case* }  **return** value; *//returns the value to invoker function* }  Complex::Complex(**double** actual, **double** supposed) { *//parameterized constructor definition* real = actual; *//saves the value of "actual" in "real"* imaginary = supposed; *//saves the value of "supposed" in "imaginary"* }  **void** Complex::printComplexNo() { *//prints the Complex Number* cout */\*<< "Complex Number: " \*/*<< real << **" + "**;  **if** (imaginary < 0) { *//if imaginary is negative then...* cout << **"("** << imaginary << **"i)"**;  } **else** cout << imaginary << **"i"**;  **return**; *//returns value to invoker function* }  Complex Complex::addTwoComplexNo(Complex second) { *//add two complex numbers* Complex result; *//declare instance of Complex class* result.real = real + second.real;  result.imaginary = imaginary + second.imaginary;  **return** result; *//returns value to invoker function* }  Complex Complex::subtractTwoComplexNo(Complex second) { *//subtract two complex numbers* Complex result; *//declare instance of Complex class* result.real = real - second.real;  result.imaginary = imaginary - second.imaginary;  **return** result;*//returns value to invoker function* }  Complex Complex::multiplyTwoComplexNo(Complex second) { *//multiply two complex numbers* Complex product; *//declare instance of Complex class* product.real = (real \* second.real) - (imaginary \* second.imaginary);  product.imaginary = (real \* second.imaginary) + (imaginary \* second.real);  **return** product; *//returns value to invoker function* }  Complex Complex::divideTwoComplexNo(Complex second) { *//divide two complex numbers* Complex quotient; *//declare instance of Complex class* quotient.real = ((real \* second.real) + (imaginary \* second.imaginary)) /  ((second.real \* second.real) + (second.imaginary \* second.imaginary));  quotient.imaginary = ((imaginary \* second.real) - (real \* second.imaginary)) /  ((second.real \* second.real) + (second.imaginary \* second.imaginary));  **return** quotient; *//returns the value to invoker function* }  Complex::Complex() { *//default constructor* real = 0; *//sets the value to ZERO* imaginary = 0; *//sets the value to ZERO* }  Complex Complex::getComplexConjugate() { *//find the conjugate of complex numbers* Complex conjugate; *//declare instance of Complex class* conjugate.real = real;  conjugate.imaginary = (-1 \* imaginary); *//binds "-1" with imaginary part* **return** conjugate; *//returns value to invoker function* }  Complex Complex::**operator**+(Complex number) { *//addition operator overloading* Complex result; *//declare instance of Complex class* result.real = real + number.real;  result.imaginary = imaginary + number.imaginary;  **return** result; *//returns value to invoker function* }  Complex Complex::**operator**-(Complex number) { *//subtraction operator overloading* Complex result; *//declare instance of Complex class* result.real = real - number.real;  result.imaginary = imaginary - number.imaginary;  **return** result; *//returns value to invoker function* }  Complex Complex::**operator**\*(Complex number) { *//multiplication operator overloading* Complex product; *//declare instance of Complex class* product.real = (real \* number.real) - (imaginary \* number.imaginary);  product.imaginary = (real \* number.imaginary) + (imaginary \* number.real);  **return** product; *//returns value to invoker function* }  Complex Complex::**operator**/(Complex number) { *//division operator overloading* Complex quotient; *//declare instance of Complex class* quotient.real = ((real \* number.real) + (imaginary \* number.imaginary)) /  ((number.real \* number.real) + (number.imaginary \* number.imaginary));  quotient.imaginary = ((imaginary \* number.real) - (real \* number.imaginary)) /  ((number.real \* number.real) + (number.imaginary \* number.imaginary));  **return** quotient; *//returns value to invoker function* }  **void** Complex::**operator**=(Complex number) { *//assignment operator overloading // Complex copy;* real = number.real;  imaginary = number.imaginary; *// return copy;* }  **(MAIN FUNCTION)**  **int** main {  string rep; *//declares variable of string-type*  **int** opt; *//declares variable of int-type*  Complex test; *//declares the instance of Complex class* update: *//"GOTO" label* system(**"clear"**); *//clear the screen* cout << **"Enter First Complex Number: \n"**;  cin >> test; *//takes input from user* Complex t1, result; *//takes instance* cout << **"Enter Second Complex Number: \n"**;  cin >> t1; *//takes input from user* again: *//"GOTO" label* system(**"clear"**); *//clear the screen* cout << **"1. Print Complex Number \n"**;  cout << **"2. Add Two Complex Number \n"**;  cout << **"3. Subtract Two Complex Number \n"**;  cout << **"4. Multiply Two Complex Number \n"**;  cout << **"5. Divide Two Complex Number \n"**; *// cout << "6. Get Conjugate of Complex Number \n";* cout << **"6. Print Resultant Complex Number \n"**;  cout << **"7. Get Real Number \n"**;  cout << **"8. Get Imaginary Number \n"**;  cout << **"Enter Your Choice: "**;  cin >> opt; *//takes input from the user* **switch** (opt) { *//switch the cases based upon user choice* **case** 1: {  cout << **"1. Print First Complex Number \n"**;  cout << **"2. Print Second Complex Number \n"**;  cout << **"3. Print Resultant Complex Number \n"**;  cout << **"4. Print All of the Above \n"**;  cout << **"Enter Your Choice: "**;  cin >> opt; *//takes input from the user* **switch** (opt) { *//switch the cases based upon user choice* **case** 1:  test.printComplexNo(); *//calls the member function to print Complex Number* cout << endl; *//makes a new line on console screen* **break**;  **case** 2:  t1.printComplexNo(); *//calls the member function to print Complex Number* cout << endl; *//makes a new line on console screen* **break**;  **case** 3:  result.printComplexNo(); *//calls the member function to print Complex Number* cout << endl; *//makes a new line on console screen* **break**;  **case** 4:  cout << **"First Complex Number: "** << test << endl;  cout << **"Second Complex Number: "** << t1 << endl;  cout << **"Resultant Complex Number: "** << result << endl;  **break**;  }  **break**; *//breaks the case* }  **case** 2: {  result = test.addTwoComplexNo(t1); *//calls the member function to add two Complex Number* cout << **" ("** << test << **") + ("** << t1 << **") = "** << result << endl;  **break**;  }  **case** 3: {  result = test.subtractTwoComplexNo(t1); *//calls the member function to subtract two Complex Number* cout << **" ("** << test << **") - ("** << t1 << **") = "** << result << endl;  **break**;  }  **case** 4: {  result = test.multiplyTwoComplexNo(t1); *//calls the member function to multiply two Complex Number* cout << **" ("** << test << **") x ("** << t1 << **") = "** << result << endl;  **break**;  }  **case** 5: {  result = test.divideTwoComplexNo(t1); *//calls the member function to divide two Complex Number* cout << **" ("** << test << **") / ("** << t1 << **") = "** << result << endl;  **break**;  }  **case** 6: {  cout << result; *//calls the cout operator overload to print Complex Number* **break**;  }  **case** 7:  cout<<**"Real Number of First Complex Number: "**<<test.getComplex(1) <<endl; *//calls the member function to print complex part of Complex Number* cout<<**"Real Number of Second Complex Number: "**<<t1.getComplex(1) <<endl; *//calls the member function to print complex part of Complex Number* cout<<**"Real Number of Resultant Complex Number: "**<<result.getComplex(1) <<endl; *//calls the member function to print complex part of Complex Number* **break**;  **case** 8:  cout<<**"Imaginary Number of First Complex Number: "**<<test.getComplex(2) <<endl; *//calls the member function to print imag. num of Complex Number* cout<<**"Imaginary Number of Second Complex Number: "**<<t1.getComplex(2) <<endl; *//calls the member function to print imag. num of Complex Number* cout<<**"Imaginary Number of Resultant Complex Number: "**<<result.getComplex(2) <<endl; *//calls the member function to print imag. num of Complex Number* **break**;  }  cout << endl << **"Do you want to use the Complex Number Calculator again? \n"**;  cout << **"If yes, then tell whether you want to reuse complex numbers or enter new complex numbers : \n"**;  cout << **"Enter E to exit the calculator, R to reuse previous values, N to enter new complex numbers \n"**;  cin >> rep; *//takes input from user* **if** (rep == **"E" or** rep == **"e"**) {  **return 0**; *//breaks the case* } **else if** (rep == **"R" or** rep == **"r"**) {  **goto** again; *//redirects to "again" label* } **else if** (rep == **"N" or** rep == **"n"**) {  **goto** update; *//redirects to "update" label* } } |

**Output:**

**Taking Input:**

**Text, letter

Description automatically generated**

**Displaying Complex Numbers:**

**Text

Description automatically generated**

**Addition:**

Text, letter

Description automatically generated

**Subtraction:**

**Text, letter

Description automatically generated**

**Multiplication:**

**Text, letter

Description automatically generated**

**Division:**

**Text, letter

Description automatically generated**

**Task B: [Marks: 20]**

Write a program in which you have to create a class that has variables to store Car data like; CarModel, CarName, CarPrice and CarOwner. The program should include functions to assign user defined values to the above-mentioned variable and a display function to show the values. Write a main that calls these functions. Now write another runner class that declares three Car objects and displays the data of all three.

|  |
| --- |
| **(Car.h)**  #ifndef INC\_2022\_SPRING\_CE\_OOP\_WEEK10\_LABTASK\_A\_BSCE21017\_CAR\_H #define INC\_2022\_SPRING\_CE\_OOP\_WEEK10\_LABTASK\_A\_BSCE21017\_CAR\_H  #include **<iostream>** #include **<string>  using namespace** std;  **class** Car { **private**:  string carModel, carName, carOwner;  **int** carPrice; **public**:  **void** setter(**int** choice, string test, **int** value = 0) {  **switch** (choice) {  **case** 1: *// cout<<"Car Name: "<<carName<<endl;* carName = test;  **break**;  **case** 2: *// cout<<"Car Owner: "<<carOwner<<endl;* carOwner = test;  **break**;  **case** 3:  carModel = test; *// cout<<"Car Model: "<<carModel<<endl;* **break**;  **case** 4:  carPrice = value; *// cout<<"Car Price: "<<carPrice<<endl;* **break**;  }  **return**;  }  **void** display(){  cout<<**"Car Name: "**<<carName<<endl;  cout<<**"Car Owner: "**<<carOwner<<endl;  cout<<**"Car Model: "**<<carModel<<endl;  cout<<**"Car Price: "**<<carPrice<<endl;  **return**;  } };  **class** Runner{ **public**:  Car a;  Car b;  Car c;  ~Runner(){  cout<<**"\nDisplay: \n"**;  cout<<**"First Object: \n"**;  a.display();  cout<<**"Second Object: \n"**;  b.display();  cout<<**"Third Object: \n"**;  c.display();  } };   #endif *//INC\_2022\_SPRING\_CE\_OOP\_WEEK10\_LABTASK\_A\_BSCE21017\_CAR\_H*  **(MAIN FUNCTION)**  **int** main {  Runner r1;  cout<<**"Enter Credentials of first object of Car: \n"**;  cout<<**"Enter Car Owner Name: "**;  cin.ignore(); *//ignores the previous cin* getline(cin,test); *//takes a string line from the user* r1.a.setter(2,test); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Car Name: "**; *// cin.ignore();* getline(cin,test); *//takes a string line from the user* r1.a.setter(1,test); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Car Model: "**; *// cin.ignore();* getline(cin,test); *//takes a string line from the user* r1.a.setter(3,test); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Car Price: "**; *// cin.ignore();* cin>>price; *//takes input from user* r1.a.setter(4,test,price); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Credentials of second object of Car: \n"**;  cout<<**"Enter Car Owner Name: "**;  cin.ignore(); *//ignores the previous cin* getline(cin,test); *//takes a string line from the user* r1.b.setter(2,test); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Car Name: "**; *// cin.ignore();* getline(cin,test); *//takes a string line from the user* r1.b.setter(1,test); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Car Model: "**; *// cin.ignore();* getline(cin,test); *//takes a string line from the user* r1.b.setter(3,test); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Car Price: "**; *// cin.ignore();* cin>>price; *//takes input from user* r1.b.setter(4,test,price); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Credentials of third object of Car: \n"**;  cout<<**"Enter Car Owner Name: "**;  cin.ignore(); *//ignores the previous cin* getline(cin,test); *//takes a string line from the user* r1.c.setter(2,test); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Car Name: "**; *// cin.ignore();* getline(cin,test); *//takes a string line from the user* r1.c.setter(1,test); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Car Model: "**; *// cin.ignore();* getline(cin,test); *//takes a string line from the user* r1.c.setter(3,test); *//calls setter function to set the value in the member variable of class* cout<<**"Enter Car Price: "**; *// cin.ignore();* cin>>price; *//takes input from user* r1.c.setter(4,test,price); *//calls setter function to set the value in the member variable of class* **return 0**; } |

**Output:**

Text

Description automatically generated with medium confidence

#### **Assessment Rubric for Lab**

**Method for assessment:**

Lab reports and instructor observation during lab sessions. Outcome assessed:

a. Ability to conduct experiments, as well as to analyze and interpret data (P) b. Ability to function on multi-disciplinary teams (A)

c. Ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (P)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Performance metric** | **Task** | **CLO** | **Description** | **Max marks** | **Exceeds expectation** | **Meets expectation** | **Does not meet expectation** | **Obtained marks** |
| 1. Realization of experiment (a) | 1 | 1 | Functionality | 40 | Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed (35-40) | Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed (20-34) | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed (0-19) |  |
| 2. Teamwork (b) | 1 | 3 | Group Performance | 5 | Actively engages and cooperates with other group member(s) in effective manner (4-5) | Cooperates with other group member(s) in a reasonable manner but conduct can be improved (2-3) | Distracts or discourages other group members from conducting the experiment (0-1) |  |
| 3. Conducting experiment (a, c) | 1 | 1 | On Spot Changes | 10 | Able to make changes (8-10) | Partially able to make changes (5-7) | Unable to make changes (0-4) |  |
| 1 | 1 | Viva | 10 | Answered all questions (8-10) | Few incorrect answers (5-7) | Unable to answer all questions (0-4) |  |
| 4. Laboratory safety and disciplinary rules (a) | 1 | 3 | Code commenting | 5 | Comments are added and does help the reader to understand the code (4-5) | Comments are added and does not help the reader to understand the code (2-3) | Comments are not added (0-1) |  |
| 5. Data collection (c) | 1 | 3 | Code Structure | 5 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap (4-5) | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables (2-3) | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy (0-1) |  |
| 6. Data analysis (a, c) | 1 | 4 | Algorithm | 20 | Solution is efficient, easy to understand, and maintain (15-20) | A logical solution that is easy to follow but it is not the most efficient (6-14) | A difficult and inefficient solution (0-5) |  |
| 7. Computer use (c) | 1 | 2 | Documentation & GitHub Submissions | 5 | Timely (4-5) | Late (2-3) | Not done (0-1) |  |
|  | Max Marks (total): | | | 100 | Obtained Marks (total): | | |  |

Lab Engineer Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_