University of Calgary
Department of Electrical and Computer Engineering
Principles of Software Design (ENSF 480)
Lab 2

Written by: M. Moussavi, PhD, PEng

Important Notes:

1. In this lab you can work with a partner (groups of three or more **are NOT allowed**). Working with a partner usually should give you the opportunity to discuss some of details of the topics and learn from each other. Also, it will give you the opportunity to practice one of the popular methods of program development called, **pair-programming.** This method which is normally associated with "Agile Software Development" technique, two programmers work together normally on the same workstation (you may consider a zoom session for this purpose). While one partner, the driver, writes the code, the other partner, acts as observer, looks over his/her shoulder making sure the syntax and solution logic is correct. Partners should switch roles frequently in a way that both of them have equivalent opportunity to practice both roles.

   If you decided to work with a partner, please submit only one lab report with both names. Submitting two lab reports with the same content will be considered as copies and plagiarism.

2. When submitting your source code (.cpp, or .h), make sure the following information appears at the top of your file:
   a. File Name
   b. Assignment and exercise number
   c. Lab section
   d. Your name
   e. Submission Date:

   Here is an example:

```
/*
 *  File Name: lab1exe_F.c
 *  Assignment: Lab 1 Exercise F
 *  Lab Section: for example, B01
 *  Completed by: Your Name (or your team name for group exercises)
 *  Submission Date: for example, Sept 18, 2022
 */
```

Due Dates:

Lab section 1 (B01): Monday, Sept 25, before 2:00 PM
Lab section 2 (B02): Wed, Sept 27, before 2:00 PM
Lab section 3 (B03): Friday, Sept 29, before 2:00 PM

Note: The due date was extended to Monday Oct 2, 2023, before 2:00 PM.

Marking scheme:

The total mark for the exercises in this lab is: **36 marks**

- Exercise A (16 marks)
- Exercise B (20 marks)

Objectives:

1. Understand the concepts such as `friend` and overloading operators in C++.
2. Understand the concept of hierarchy, another pillar of OOP, which is known as, inheritance.

Note:

Your lab reports must have a cover page with the following information. This is just for helping your TAs to easily record your marks on the D2L system.

Name (s): **If working with a partner both students' name must be written**
Course Name: Principles of Software Design
Lab Section: B01 or B02
Course Code: ENSF 480
Assignment Number: For example, Lab-1
Submission Date and Time: DD/MM/YYYY

Exercise A - Overloading Operators in C++ (16 marks)

In this exercise you are going to use some of the files that you have used in lab1, exercise B, and you will overload several operators, for classes Mystring, and DictionaryList. For your convenient the copy of those files is again posted on the D2L, under the Lab 2 section.
Note: Once you downloaded the files from D2L, make sure to use your code from lab 1, exercise B, to complete the missing definitions of member functions: `find, copy, and destroy.`

What to Do:

Open the file `exBmain.cpp` and uncomment the line that calls function `test_operator_overloading`, and its prototype at the top of the file. Then, change the conditional compilation right above the implementation of `test_operator_overloading` from `#if 0` to `#if 1` and try to compile and run the program. Now you will see a few errors. These errors are due to the fact that this function is trying to make some operations on Mystring or DictionaryList objects using common C++ relational operators such as, >, <, <=, !=, or other type of operators such as << or [] that are not by default defined by C++ compiler for objects of Mystring or DictionaryList. Your job in this assignment is to find out which operator is required to be overloaded and write the necessary code in files `mystring_B.h, mystring_B.cpp, dictionalyList.h` and `dictionaryList.cpp`.

Since there are several operators to be overloaded, again the best strategy is to work incrementally. Means first comment out most of the lines in function test_overloading, except the two lines that uses the operator >= and << . Then write the necessary code in the given files. If they work fine, then uncomment the next few lines to implement and test the next operator -- until all required operators are properly defined and tested.

Instructions to Submit Exercises A:

Submit the following files electronically on the D2L:

- As part of your lab report (PDF file), submit the copy of all .cpp and .h files, and the output of your program.
- Submit a zipped file that contains your actual source code (all .cpp and .h files)

Exercise B - Inheritance in C++ (20 marks)

The concepts of aggregation, composition, and inheritance in C++ and Java are similar, but they are completely different in terms of implementation and syntax. Also, another major difference between the two languages is that C++ unlike Java supports multiple inheritance.

What to Do: Inheritance in C++:

In this exercise, first you will write the definitions of following classes: Point, Shape, Rectangle, Square, GraphicsWorld, as explained below.

Class Point:

This class is supposed to represent a point in a Cartesian plane. It should have three data members: **x**, and **y** coordinates and an **id** number that its value will be assigned automatically. The first object's **id** number should be 1001, second one should be 1002, and so on. Class Point should also have at least the following member functions:

- `display` - that displays x, and y coordinates of the point in the following format:
  ```
  X-coordinate: ######.##
  Y-coordinate: ######.##
  ```
- A constructor that initializes its data members.
  Note: You are not supposed to define a default constructor in this class. Automatic calls to the default constructor will hide some of the important aspects of this assignment (marks will be deducted if you define a default constructor for this class).

- Access functions, getters and setters, as needed.

- Function `counter()` that returns the number of objects of class Point at any time.

- Two distance functions that return the distance between two points. One of the two must be a static function.

You should create two files for this class: called `point.h and point.cpp`

Class Shape:

This class is the base class or the ancestor of several classes, including class Rectangle, Square, Circle, etc. It should support basic operations and structures that are common among the children of this class. This class should have an object of class `Point` called `origin`, and a `char` pointer called `shapeName` that points to a dynamically allocated memory space, allocated by the class constructor. This class should also have several functions and a constructor as follows:

- A constructor that initializes its data members.
- **No default constructor**
- A destructor that de-allocates the memory space allocated dynamically for `shapeName`
- `getOrigin` – that returns a reference to point origin. The reference should not allow the x and y values of point to be modified through this reference.
- `getName` – that returns the name of the shape.
- `display` – that prints on the screen the shape's name, x and y coordinates of point origin, in the following format:
  ```
  Shape Name:
  X-coordinate:
  Y-coordinate:
  ```
- two `distance` functions
  ```
  double distance (Shape& other);
  static double distance (Shape& the_shape, Shape& other);
  ```

- move: that changes the position of the shape, the current x and y coordinates, to x+dx, and y+dx. The function's interface (prototype) should be;
```
void move (double dx, double dy);
```

You should create two files for this class: called shape.h and shape.cpp

Class Square:

This class is supposed to be derived from class Shape, and should have one data member, called side_a, and in addition to its constructor should have a constructor and several member functions as follows:
- One constructor that initializes its data members with its arguments supplied by the user.
- **No default constructor**. Marks will be deducted if a default constructor is defined for this class
- area – that returns the area of a square
- perimeter: that returns the perimeter of a square
- get and set - as needed.
- display – that displays the name, x and y coordinates of the origin, side_a, area, and perimeter of a square object in the following format:
```
Square Name:
X-coordinate:
Y-coordinate:
Side a:
Area:
Perimeter;
```

- More Functions, if needed.

You should create two files for this class: called square.h and square.cpp

Class Rectangle:

Class Rectangle that is derived from class Square needs to have one more side called side_b. This class, in addition to its constructor (no default constructor), should support the following functions:
- area – that calculates and returns the area of a rectangle
- perimeter – that calculates and returns the perimeter of a rectangle
- get and set – that retrieves or changes the values of its private data members.
- display – that displays the name, and x and y coordinate origin of the shape, side_a, side_b, area, and perimeter of a rectangle object, in the following format:
```
Rectangle Name:
X-coordinate:
Y-coordinate:
Side a:
Side b:
Area:
Perimeter;
```
- More Functions, if needed.

You should create two files for this class: called rectangle.h and rectangle.cpp

Class GraphicsWorld:
This class should have only a function called *run*, which declares instances of the above-mentioned classes as its local variable. This function should first display a short message about the author(s) of the program and then start testing all of

the functions of the classes in this system. A sample code segment of function *run* is given below. You are recommended to use conditional compilation directives to test your code (one class at a time).

```
void GraphicsWorld::run(){
#if 0                // Change 0 to 1 to test Point
      Point m (6, 8);
      Point n (6,8);
      n.setx(9);
      cout << "\nExpected to dispaly the distance between m and n is: 3";
      cout << "\nThe distance between m and n is: " <<      m.distance(n);
      cout << "\nExpected second version of the distance function also print: 3";
      cout << "\nThe distance between m and n is again: "
           <<  Point::distance(m, n);
#endif              // end of block to test Point

#if 0                // Change 0 to 1 to test Square
    cout << "\n\nTesting Functions in class Square:" <<endl;
    Square s(5, 7, 12, "SQUARE - S");
    s.display();
#endif              // end of block to test Square

#if 0                // Change 0 to 1 to test Rectangle
      cout << "\nTesting Functions in class Rectangle:";
      Rectangle a(5, 7, 12, 15, "RECTANGLE A");
      a.display();
      Rectangle b(16 , 7, 8, 9, "RECTANGLE B");
      b.display();
      double d = a.distance(b);
      cout <<"\nDistance between square a, and b is: " << d << endl;
      Rectangle rec1 = a;
      rec1.display();
      cout << "\nTesting assignment operator in class Rectangle:" <<endl;
      Rectangle rec2 (3, 4, 11, 7, "RECTANGLE rec2");
      rec2.display();
      rec2 = a;
      a.set_side_b(200);
      a.set_side_a(100);
      cout << "\nExpected to display the following values for objec rec2: " << endl;
      cout << "Rectangle Name: RECTANGLE A\n" << "X-coordinate: 5\n"  << "Y-coordinate: 7\n"
               << "Side a: 12\n" << "Side b: 15\n" << "Area: 180\n" << "Perimeter: 54\n" ;
      cout << "\nIf it doesn't there is a problem with your assignment operator.\n" << endl;
      rec2.display();

      cout << "\nTesting copy constructor in class Rectangle:" <<endl;
      Rectangle rec3 (a);
      rec3.display();
      a.set_side_b(300);
      a.set_side_a(400);
      cout << "\nExpected to display the following values for objec rec2: " << endl;
      cout << "Rectangle Name: RECTANGLE A\n" << "X-coordinate: 5\n"  << "Y-coordinate: 7\n"
      << "Side a: 100\n" << "Side b: 200\n" << "Area: 20000\n" << "Perimeter: 600\n" ;
      cout << "\nIf it doesn't there is a problem with your assignment operator.\n" << endl;
      rec3.display();
#endif              // end of block to test Rectangle
#if 0                // Change 0 to 1 to test using array of pointer and polymorphism
      cout << "\nTesting array of pointers and polymorphism:" <<endl;
      Shape* sh[4];
      sh[0] = &s;
      sh[1] = &b;
      sh [2] = &rec1;
      sh [3] = &rec3;
      sh [0]->display();
      sh [1]->display();
      sh [2]->display();
      sh [3]->display();

#endif              // end of block to test array of pointer and polymorphism
```

You should create two files for this class: called `graphicsWorld.h and graphicsWorld.cpp`

What to Submit for Exercise B: As part of your lab report (PDF file) copy and paste your source codes (.cpp and .h files) and the program output, showing your program works. Also submit a zipped file that contains your source files: .cpp and .h for all your classes.