University of Calgary
Department of Electrical and Computer Engineering
ENSF 614 Lab 5 – Winter, 2023

Notes:

- You can work with a partner, which means only groups of two are allowed (groups of three or more are not allowed).
- Some exercises in our lab assignments will not be marked. **Please do not skip them**, as unmarked exercises are as important as others.

Objective:

The purpose of this lab is to practice some of the basic object-oriented design concepts such as aggregation, composition, polymorphism, inheritance, multiple inheritance in C++, and application of pointer to pointer in languages such as C and C++.

Marking Scheme: (30 marks total)

- Exercise A: 20 marks
- Exercise B: 10 marks
- Exercise C: Not marked.

Due Date:

Sunday, March 5, 2023, before 11:59 PM

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

The concepts of aggregation, composition, and inheritance in C++ and Java in terms of concepts 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: Single Inheritance in C++:

In this exercise, first, you will write the definitions of the following classes: Point, Shape, Rectangle, Square, and 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; the second one should be 1002, and so on. Class Point should also have at least the following member functions:

- `display` - that displays the 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: `point.h and point.cpp`

Class Shape:
This class is the base class or the ancestor of several classes, including the 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: `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: `square.h and square.cpp`

Class Rectangle:

Class Rectangle 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: `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 the functions of the classes in this system. A sample code segment of the function *run* is given on the next page.

You are recommended to use conditional compilation directives to test your code (one class at a time).

<u>Sample Code for the run () function:</u>

```
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 display 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
```

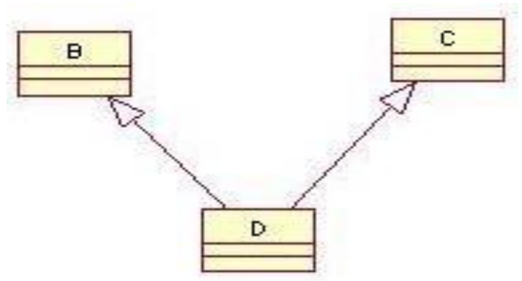
You should create two files for this class: called `graphicsWorld.h and graphicsWorld.cpp`

What to Submit for Exercise A:

As part of your lab report (PDF file), submit your source codes (.cpp and .h files) and the program output showing your code in exercise A.

Exercise B - Multiple-Inheritance (10 marks)

This exercise is the continuation of exercise A. You must complete exercise A and then start it.
C++ allows a class to have more than one parent:



This is called multiple inheritance. For example, if we have two classes called B and C, and class D is derived from both classes (B and C), we say class D has two parents (multiple inheritance). This is a powerful feature of C++ language that other languages, such as Java, do not support. For further details, please refer to your class notes.

What to Do

In this exercise, you should add two new classes to your program:

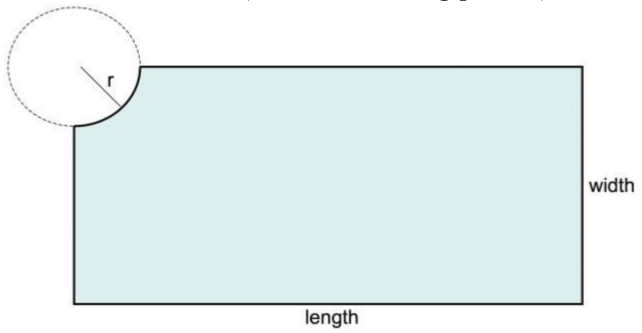
Class Circle:

This class is supposed to be derived from class Shape, and should have a data member radius. In addition to its constructor, it should support similar functions as the class `Rectangle`, such as `area`, `perimeter`, `get`, `set`, and `display`.

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

Class CurveCut:

CurveCut represents a shape that needs properties of class Rectangle and class Circle. In fact, it is a rectangle whose left top corner can have an arch-form cut (see the following picture).



This class must be derived from class Rectangle and class Circle. The origins (x and y coordinates) of both shapes are the same. This class, in addition to a constructor, should support the following functions:

- `area` – that calculates the highlighted area of the above figure.
- `perimeter` – that calculates and returns the perimeter of highlighted areas.

- `display` – that displays the name, x, and y coordinates of the origin of the shape, width, and length (as shown in the picture above), and radius of the cut, in the following format:

```
 CurveCut Name:
X-coordinate:
Y-coordinate:
Width:
Length:
Radius of the cut:
```

Note: The radius of the circle must always be less than or equal to the smaller of the width and length. Otherwise, the program should display an error message and terminate.

You should also add some code to the function `run` in class `GraphicsWorld` to:

- test the member functions of the class CurveCut.
- use an array of pointers to Shape objects, where each pointer points to a **different object** in the shapes hierarchy. Then test the functions of each class again.

A sample code segment that you can use to test your program is given in the following box (you can add more codes to this function if you want to show additional features of your program).

```
void GraphicsWorld::run(){
    /****************************ASSUME THIS CODE SEGMENT FOR EXERCISE A IS HERE
************************/
#if 0
       cout << "\nTesting Functions in class Circle:" <<endl;
       Circle c (3, 5, 9, "CIRCLE C");
       c.display();
       cout << "the area of " << c.getName() <<" is: "<< c.area() << endl;
       cout << "the perimeter of " << c.getName() << " is: "<< c.perimeter() << endl;
       d = a.distance(c);
       cout << "\nThe distance between rectangle a and circle c is: " <<d;
       CurveCut rc (6, 5, 10, 12, 9, "CurveCut rc");
       rc.display();
       cout << "the area of " << rc.getName() <<" is: "<< rc.area();
       cout << "the perimeter of " << rc.getName() << " is: "<< rc.perimeter();
       d = rc.distance(c);
       cout << "\nThe distance between rc and c is: " <<d;


       // Using an array of Shape pointers:
       Shape* sh[4];
       sh[0] = &s;
       sh[1] = &a;
       sh [2] = &c;
       sh [3] = &rc;
       sh [0]->display();
       cout << "\nthe area of "<< sh[0]->getName() << "is: "<< sh[0] ->area();
       cout << "\nthe perimeter of " << sh[0]->getName () << " is: "<< sh[0]->perimeter();
       sh [1]->display();
       cout << "\nthe area of "<< sh[1]->getName() << "is: "<< sh[1] ->area();
       cout << "\nthe perimeter of " << sh[0]->getName () << " is: "<< sh[1]->perimeter();
       sh [2]->display();
       cout << "\nthe area of "<< sh[2]->getName() << "is: "<< sh[2] ->area();
       cout << "\nthe circumference of " << sh[2]->getName ()<< " is: "<< sh[2]->perimeter();
       sh [3]->display();
       cout << "\nthe area of "<< sh[3]->getName() << "is: "<< sh[3] ->area();
       cout << "\nthe perimeter of " << sh[3]->getName () << " is: "<< sh[3]->perimeter();
       cout << "\nTesting copy constructor in class CurveCut:" <<endl;
       CurveCut cc = rc;
       cc.display();
       cout << "\nTesting assignment operator in class CurveCut:" <<endl;
       CurveCut cc2(2, 5, 100,  12, 9,  "CurveCut cc2");
       cc2.display();
       cc2 = cc;
       cc2.display();
#endif
}       // END OF FUNCTION run
```

What to Submit for Exercise B:

1. As part of your lab report (PDF file) submit a copy of your complete source codes (.cpp and .h files), and the program output showing your code in exercise B works.
2. A zipped file with source file: All `.cpp` and `.h` files

Exercise C: Array of Pointers and Command Line Arguments

Read This First:

Up to this point, in our C++ programs, we have always used main functions without any arguments. In fact, C++ allows us to write programs whose main functions receive arguments. Here is the heading of such a function:

```
int main(int argc, char **argv){ ... }
```

Of course, we never call a main function from anywhere in our program, and we never pass arguments to it. A main function can only receive its arguments from the command line. In other words, when a user runs the program from the command line, he/she can pass one or more arguments to the main. Good examples of this type of program are many Linux and Unix commands such as `cp, mv, gcc, g++`. For example, the following cp command receives the name of two files to make file f.dat a copy of file `f.txt`:

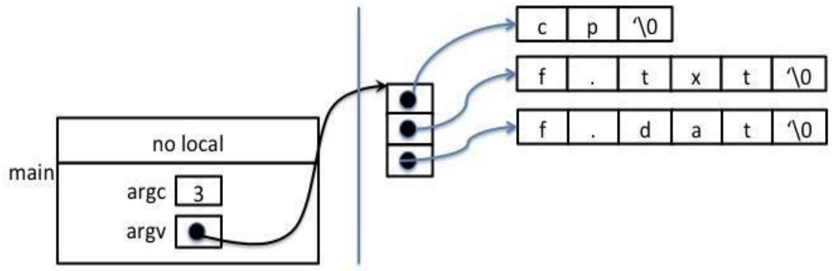
```
            cp f.txt f.dat
```

The first token in the above cp command is the program's executable file name, followed by two file names. This information will be accessible to the `main`. How does it work? Here is the answer:

To access the command-line arguments, a C++ main function can have arguments as follows:

```
#include <iostream> using
std::cerr;
int main(int argc, char **argv) {
 if (argc != 3) {
        cerr << "Usage: incorrect number of arguments on the command line...\n”";
exit (1);
    }
    // MORE CODE AS NEEDED.
return 0;
}
```

Where `argc` is the number of tokens on the command line. In our cp command example, the value of `argc` should be always 3 (one for the program name and two for the file names). If the value of `argc` is not `3`, the program terminates after giving an appropriate message. In other words, this argument is mainly used for error-checking to make sure the user has entered the right number of tokens on the command line. The delimiter to count the number of tokens on the command- line is one or more spaces. The second argument is a pointer-to-pointer, which points to an array of pointers. Each pointer in this array points to one of the string tokens on the command line. The following figure shows how `argv[0], argv[1]`, and `argv[2]` point to the tokens on the command line:



The following code shows how you can simply access and display the command line strings in a C++ program:

```
cout << "The program name is: " << argv[0] << endl;
cout << "The first string on the command line is: " << argv[1] << endl;
cout << "The second string on the command line is: " << argv[2] << endl;
```

And here is the output of this code segment for our cp command example:

```
The program name is: cp
The first string on the command line is: f.txt
The second string on the command line is: f.dat
```

The exact location of the memory allocated for command-line arguments depends on the underlying OS and the compiler. Still, for most C++ systems, it's a specific area on the stack that is not used for the activation records.

Read This Second:
Since this part of exercise-A deals with the command line entries by the user, you are **strongly** recommended to implement, compile, and run the program on a Cygwin terminal or Mac terminal.

You may be interested in knowing whether it is possible to pass command-line arguments to your program from inside an IDE environment such as Visual C++ or XCode. The answer is yes! It is possible.

Most commonly used IDEs d IDEs provide some way of entering the command line arguments into C++ project. For example, in the newer versions of the XCode for the Mac computers, you can click on the 'Edit Scheme' under the 'Product' menu option and enter the command line argument strings on the 'arguments' tab. A similar way is also available in Visual Studio: Right-click on the project, then select 'Properties' -> Debugging, and enter your arguments into the arguments box. If you choose to use this option, you should seek more details by consulting the help options available on your target IDE or searching on the Internet.

Our experience shows that using the command line argument from inside some of the IDEs is not always straightforward, and figuring out how exactly it works, can sometimes be time-consuming.

What to Do:
Download the file `lab5ExC.cpp` from D2L. This is a simple program that uses the insertion-sort algorithm to sort an array of integer numbers. Now, you should take the following steps:

1. Read the given program carefully to understand what it does.
2. If you are using Cygwin, or Mac terminal, use the following command to create an executable file called `sort`: `g++ -Wall lab5ExC.cpp -o sort`
3. Run the program using the following command: `./sort`
   It should print the list of several integer numbers, followed by the same list, sorted in ascending order.
4. Now change the value of the local variable `sort_order` from 1 to 2, recompile the program, and run it again. Now it should sort the array in descending order.
5. Your task from this point is to modify the main function to have access to the command line arguments. We want to be able to run the program with the option of sorting the numbers either in ascending or descending or to run the program with the option of sorting the numbers der, based on the user's input on the command line.
   Considering that the program's executable name is `sort`, users must be able to run the program from the command line in one of the following formats:
   ```
   ./sort  -a
   ```
   Or:
   ```
   ./sort  -d
   ```

   In the first command, option `-a` (no spaces between the dash and a) stands for the ascending, and the second command with option `-d` stands for the descending order.

   Depending on the user's selection of one of these options, the program must sort an array accordingly. To be more precise, the main function of this program must check the following conditions:
   - If argc > 2, the program should give the following message and terminate:
     Usage: Too many arguments on the command line

   - If argc == 1, (meaning that the user did not enter any of the two options), the program should use ascending sort as its default order.

- If argv[1], is NOT one of the following: –a, -A, -d, or –D, the program should give the following message and terminate:
  Usage: Invalid entry for the command line option.

Then, the program should:

- Set the value of `sort_order` to 1, if `argv[1]` points to the C-string: `“–a”` or `“-A”`.
- Set the value of `sort_order` to 2, if `argv[1]` points to the C-string: `“–d”` or `“-D”`.

Note: as you should recall from the material discussed earlier in the course, you cannot compare C-strings simply using relational operators. Therefore, you need to call the C++ library function strcmp. You may refresh your memory regarding this function by reading your course notes, any of your textbooks, or online sources such as cplusplus.com at: http://www.cplusplus.com/reference/cstring/strcmp.

What to Submit:

You do not need to submit anything for this exercise.