Lab 10
Abstract Base Classes (ABC), Exception Handling

Objectives

1. Get more familiar with inheritance and ABCs
2. Learn about Exception Handling mechanisms in C++

Prelab Assignments

- If not done, finish the reading assignment 9, posted on Blackboard.
- Make sure you have working code from lab 7.

Part 1

In the first part of this final lab you will define an ABC for Conic Sections, similar to the one described in lecture 15. You will then use that base class to create a Circle class and an Ellipse class, through inheritance. More specific details and requirements are the following:

- Your classes should have appropriate constructors to initialize the various data members.
- Both Circle and Ellipse classes should have: a Move method, to shift the center of the conic; an Area method, to calculate the surface area of the conic; a Resize method, to modify the radius or semi-major/minor axis values, respectively; and a Display method, to print all the relevant information of the conic. **Which methods should be virtual?**
- The Circle class should have an additional method to determine if a given point (x, y) is inside, outside or on the circle.
- The Ellipse class should have two additional methods: one to rotate the ellipse, and another to calculate the position of the vertices.

Experiments:

- Create different conic objects; test all your methods.
- Try creating objects of the ABC directly. What happens? Explain.
- Create pointers to the ABC class and use them to point to the derived class objects that you created before. Use those pointers to call the different methods (dynamic binding). Compare your results to those obtained using the original objects.

Part 2

In this part you will modify your lab 7 code (which modified lab 6) to include exception handling through "throws", "catch blocks" and "try blocks". You are to define where in your program you will have the try blocks, and where you will have the throws and catch blocks (handlers). Furthermore, you will define *what* to throw. You will need to modify your lab 7 functions / methods a bit to include the throws, catch and try blocks.

The exceptions and minimal requirements and cases are described below (you are encouraged to experiment and report additional ones). You may create additional, simple functions for testing purposes. You may also remove / comment out or simplify some of your lab 7 code that may not be needed for the experiments described next.

Exceptions:

- Invalid inputs (e.g. wrong or missing text file, scaling the signals by 0 will not be allowed).
- Trying an operation (offset, scale, normalize, etc.) on an empty signal vector (no data to process).
- Trying to add Signal objects with different lengths. In lab 7, you were required to create a non-member addition operator that would take two Signal objects. Their corresponding data vectors should be the same length. Otherwise, an exception should be thrown.

Requirements:

- You should have at least 3 different matching throws and catch blocks, including one object type. For example: catch(int x) ←→ throw(0); catch(myClass obj) ←→ throw (obj1). The exception class that you create can be very simple. Check the Lecture 19 examples.
- You should have a "catch-all" handler.
- You should have one "re-throw". You will need two try blocks to test a "re-throw" properly.
- At least one of your functions / methods should have an exception specification to indicate what kind(s) of exception(s) it throws.

Experiments:

- Test all your exceptions.
- Try throwing an exception that does not have a matching catch block. What happens? Next, comment out the "catch-all" handler. What happens then when the exception is thrown? Explain.

- Test the "re-throw". Then, comment it out. Explain what you observe.
- When testing your function(s) / method(s) for which you included an exception specification, try throwing a different type than the one specified. What happens? What happens if you remove the exception specification from your function definition?

Push your code to your github repository.

Deliverables:

1. Lab Report:
    1) The report should follow the lab report format posted on Blackboard.
    2) Results: show results of part 1 and 2. Include answers to the questions stated in the guide. Include screenshots when appropriate.
    3) Discussion: Make sure to discuss your results and to describe any problems and interesting things that you may have encountered while coding the lab, especially those related to the new concepts.
    4) Screenshot of your Github repository showing your commit.
    5) Append your source code with proper comments and indentation to the end of you report.

2. Demonstration:

    Show a few test cases to the TA.

Grading:

Demo:	20	
Report:	40	
Code:	40	**(Well organized, properly commented and indented)**