HW8
Out: Nov 29, Wednesday -- DUE: December 8, Friday 4:00pm
EC327 Introduction to Software Engineering – Fall 2023

Total: 100 points

- *Comment your code and use a clean, easily understandable coding convention. A few coding style guides for your reference:*

http://geosoft.no/development/cppstyle.html
https://google.github.io/styleguide/cppguide.html

Submission:

1. Failure to follow guidelines (filenames, program I/O, and others) WILL result in a significant loss of points on the assignment as we use autograders.
2. Completed assignments **must** be submitted on GradeScope.
3. Submit the required files for each question: Q1.cpp Q2.cpp Q2.h Q3.cpp Q4.cpp
   Please make sure your code compiles and runs as intended on the **engineering grid**. Code that does not compile will NOT be graded and will receive a 0.

Late Homework Submissions: Late submission policy does **not** apply to HW8. Submissions received after the deadline will not be graded.

Please write C++ code for solving the following problems. Each question is 25 points.

Notes about this assignment and the final exam:

- The questions are similar in style and length to final exam questions. It is recommended that you time yourself when coding; each question should ideally take around 30 minutes. Note that fourth question may take more time as you will be exploring how to use a map in STL.

- This year's final exam will differ in organization and content, but there will be three coding questions (that you will pick among tentatively four questions). The exam may also contain a few short questions on concepts, similar in style to the questions in quizzes.

Q1. – Files to submit: Q1.cpp. Write all your program in the provided Q1.cpp file.

a. (10 pts) Write a C++ function that reverses the order of characters in a string. The function has the following prototype and returns the reversed string:

```
string reverseString(const string &s1);
//The reverseString function should throw an exception if the input string
//is more than 15 characters.
```

b. (7.5 pts) Write a C++ program that reads an arbitrary number of words from a file called *words.txt* (one word per line), reverses the order of letters in each word using the reverseString function, and prints the resulting list of words to a file named *reversewords.txt*.

c. (7.5 pts) In your program, implement exception handling (using `try` and `catch`) to print error messages for input words that are longer than 15 characters. The program should be able to continue execution after handling the exceptions and stop only when there are no more lines to read in the file. If an input line is empty, your program should skip the empty line and continue processing other lines.

Example test case:

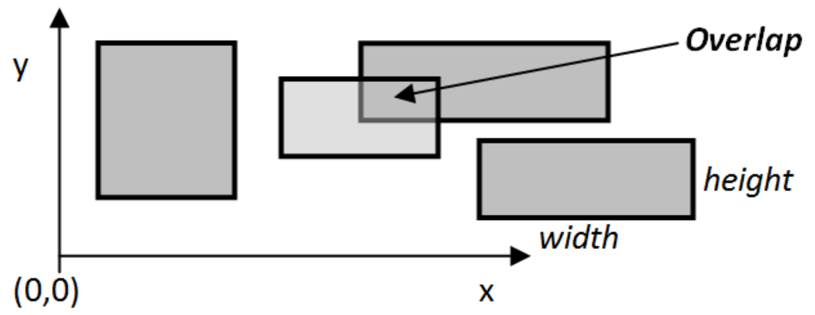
words.txt	*reversewords.txt*	*Console output*
This is your final exam haveawonderfulwinterbreak ec327	sihT si ruoy lanif maxe 723ce	Error: line 7 exceeds 15 characters, skipping

Q2. - Files to submit: Q2.cpp and Q2.h. Your class declaration is in Q2.h and all other code should go in Q2.cpp.

Consider a **Rectangle** class that stores the following information:

Height – height of the rectangle (integer)
Width – width of the rectangle (integer)
Coordinates – Position of the *lower left corner* of a rectangle in a plane (two integers).

One side of each rectangle is always parallel to the x axis of the plane and all rectangles lie in the positive xy plane. Figure below shows a visual abstraction of a set of Rectangle objects.



The definition of the Rectangle class is provided in Q2.h.

a. (10 pts) Implement the appropriate accessor functions (getHeight and getWidth).

b. (10 pts) Implement two additional member functions defined below:

```
int getOverlapArea(const Rectangle & r) const;
//Checks whether the Rectangle object overlaps with another Rectangle
object (Rectangle &r) on the plane. Function returns 0 if there is no
overlap, and returns the area of the overlap otherwise.

string getCoordinates() const;
//returns the coordinates of the lower left corner of the Rectangle in a
string format. For example, if x=1 and y=2, it will return "1,2".
```

c. (5 pts) Basic test code is provided to you in Q2.cpp. Write additional test code to ask the user to create two rectangle objects (by providing height, width, and coordinates) and print out the overlap area. Your test code should follow the sample run in the given Q2.cpp file.

Q3. - Files to submit: Q3.cpp (implement all classes and functionality in a single file)

You are provided with a *Dessert* class in Q3.cpp. It should compile and run. Make sure that it does compile BEFORE proceeding.

1. Add a destructor to *Dessert*. Include the cout statement provided. (5 points)
2. Add a class called *Tiramisu*. This class should be derived from *Dessert*. All it needs is a destructor (nothing else). The destructor needs to include the cout statement provided. (10 points)
3. Add the following function `void eat(Dessert *ptr)` as a non-member function to your program. Use dynamic casting to determine if the argument provided to the function can be eaten. Only Tiramisu can be eaten. Use the cout statements provided to indicate the result. (10 points)

When you are done, un-comment the lines in main. DO NOT add any more lines to main and DO NOT add any other cout statements.

Hint: consider the result of a successful dynamic cast vs. an unsuccessful cast.

Q4. - Files to submit: Q4.cpp. You are provided a skeleton implementation of your submission in Q4.cpp.

This question requires you to use data structures that are implemented by the Standard Template Library (STL).

Write a program to store the book catalog of a library. A skeleton for the program is provided in `Q4.cpp`. The book database is implemented as a map, where the key is the ISBN number for the book (as an integer) and the value is the title of the book (as a string). The user is asked to provide the details for a new book (including both the ISBN and the title) until they press "-1" when asked if they want to insert a new item. Implement the following functionalities:

a) (10 pts) Create a map (with integers as key and strings as values) to store the items provided by the user.

b) (5 pts) Every time a new item is provided, first check that a book with the same ISBN does not already exist in the map. If it already exists, print out the message below:

```
A book with that ISBN is already in the library!
```

If the item does not exist, add it to the list.

c) (10 pts) After the user selects that they don't want to add any more items, write the book titles to a file named library_list.txt, one item per line. Include only the titles and not the ISBN codes.