ECE250: Lab Project 1

Due Date: Friday, January 27, 2017 – 11:00PM

1. Project Description

The goal of this project is to design and implement a dynamic range stack as an array. A range stack data structure maintains the range of values stored in a stack as well as the entries themselves. Thus, it is possible to call *top()* to return the top of the stack, but it is also possible to call *maximum()*, which returns the maximum element in the stack, and *minimum()*, which returns the minimum element in the stack. All operations must occur in $O(1)$ time. In this project, to implement this data structure, we use three arrays; the *stack array* which is a simple stack for storing the items of the stack. *min_array and max_array* are also used to keep track of the minimum and maximum values of the stack. The skeleton code for the class *Dynamic_range_stack.h* and the utility classes *Exception.h* and *ece250.h* are given on the course website. Make sure to include comments as appropriate in your program as per course guidelines. Also, it is not allowed to use functions of other libraries except those that are included in the skeleton files.

2. How to Test Your Program

We use drivers and tester classes for automated marking, and provide them for you to use while you build your solution. We also provide you with a basic test case, in file *test.in* which can serve as a sample for you to create more comprehensive test cases.

3. How to Submit Your Program

Once you have completed your solution, and tested it comprehensively, you need to build a compressed file, in tar.gz format, which should contain the followings:

- Dynamic_range_stack.h

Build your tar file using the UNIX tar command as given below:

```
tar -cvzf xxxxxxxx_pn.tar.gz Dynamic_range_stack.h
```

where ***xxxxxxxx*** is your UW user id (ie. jsmith), and ***n*** is the project number which is 1 for this project. All characters in the file name must be lower case. Submit your tar.gz file using LEARN, in the drop box corresponding to this project

4. Class specification

The class implements a dynamic range stack (in fact, three stacks) which has the specified behaviors. For run-time requirements, the number of elements in the stacks is n. Notice that the expected running time of each member function is specified in parentheses at the end of the function description. It is important that your implementation follows this requirement strictly (submissions that do not satisfy these requirements will not be given full marks).

Member Variables
The class has eight members:

- An integer storing the initial size of the array*: initial_size.*
- An integer storing the current size of the arrays*: array_size.*
- An integer storing the number of elements in the stack array*: entry_count.*
- An integer storing the number of elements in the maximum array: *max_count*
- An integer storing the number of elements in the minimum array: *min_count*
- Three arrays of size *array_size: stack_array, maximum_array,* and *minimum_array.*

Constructor
Dynamic_range_stack(int n = 10)- The constructor takes as an argument the initial size of the arrays and allocates memory for the three arrays. If the argument is either 0 or a negative integer, set the initial size of the array to 1. The default initial capacity of the array is 10. Other member variables are assigned as appropriate.

Destructor
~Dynamic_range_stack() – The destructor deletes the memory allocated for the arrays.

Accessors
This class has five accessors:

- *int top() const* – Return the object at the top of the stack (*stack_array*). It may throw an underflow exception ($O(1)$).
- *int maximum() const* – Return the maximum object currently in the stack. It may throw an underflow exception ($O(1)$)).
- *int minimum() const* – Return the minimum object currently in the stack. It may throw an underflow exception ($O(1)$)).
- *int size() const* – Returns the number of elements currently stored in the stack ($O(1)$)).
- *int capacity() const* – Return the current capacity of the arrays in ($O(1)$)).
- *bool empty() const* – Determines whether the stack is empty or not ($O(1)$).

Mutators
This class has three mutators:

- *void push(int const &)* – If the stack array is full, create three new arrays which are double in size, copy over all the entries, and delete the old arrays. Push the argument onto the top of the *stack_array*, and update the other stacks if needed. Assume that push function is not called for an item that is already in the stack (no duplicate) (in average $O(1)$).
- *int pop()* – Pop the top element off of the stack(s) by removing it from the stack array and update other stacks if needed. This may throw an underflow exception (in average $O(1)$).
- *void clear()* – Empties the stacks by resetting the member variables. If the current array size does not equal the initial size, delete the arrays and create three new arrays equal to the initial size ($O(1)$).