ECE250: Lab Project 1

Due Date: Sunday, October 8, 2017 – 11:00PM

1. Project Description

The goal of this project is to design and implement a dynamic stack data structure. A stack is a data structure that stores elements in an ordered list and allows insertions and deletions at one end only (the top of stack) in **O**(1) time. A dynamic stack is able to increase its size if necessary in **O**(**n**).

Create a class called *Dynamic_stack* according to the requirements given in Section 4 of this document. The skeleton code for the class *Dynamic_stack* 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.

2. How to Test Your Program

You will need to implement the methods of the stack class in the *Dynamic_stack.h* file. Make sure that your test cases cover most functionality for the *Dynamic_stack* class as well as boundary cases.

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* that can serve as a sample for you to create more comprehensive test cases. You can find the testing files on the course website. To compile and run your code with an input test file:

```
g++ Dynamic_stack_driver.cpp -o Dynamic_stack_driver
./Dynamic_stack_driver < test.in
```

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_stack.h

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

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

where ***xxxxxxxx*** is your UW user id (e.g., 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 *Dynamic_stack* class implements a dynamic stack data structure using an array. In order to simplify the implementation of the data structure in the first project, we will assume that this stack can only store elements of type *int*. For the rest of the projects, we will use C++ Class Templates to create data structures that accept a variety of data types.

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 *Dynamic_stack* class has four member variables:

- *int* array* – A pointer to be used as an array
- *int count* – Number of elements in the stack
- *int array_size* – The size of the array
- *int initial_size* – The initial size of the array

Constructor

Dynamic_stack (int n=10) - The constructor takes as argument the initial size of the array and allocates memory for it. The default capacity (or number of entries) is 10. If the argument *n* is less than one, use an array size of 1. Other class members should be initialized with appropriate values.

Destructor

~Dynamic_stack() – The destructor deletes the memory allocated for the array.

Accessors

This class has four accessors:

- *int top() const* – Return the integer at the top of the stack in **O**(1). It may throw an underflow exception.
- *int size() const* – Return the number of elements currently stored in the stack in **O**(1).
- *bool empty() const* – Return true if stack is empty false otherwise in **O**(1).
- *int capacity() const* – Return the capacity of stack in **O**(1).

Mutators

This class has three mutators:

- *void push(int const &)* – Insert a new item at the head of the stack in **O**(1). If the array is full, create a new array with size *2n* (where *n* is current size of the array) and copy the elements from the current array to the new array in **O**(n).
- *int pop()* – Remove the element at the top of the stack in **O**(1). This may throw an underflow exception.
- *void clear()* – Remove all elements from the stack in **O**(1). If the current array size does not equal the initial size, delete the array and create a new array equal to the initial size.