Lab 12: Error Handling with Exceptions  
In this lab you will:

* Write exception classes
* Catch and handle exceptions

You are given an IntStack class that has plenty of error checking. However, whenever an error occurs, the program exits. This is bad because you don't want IntStack to handle the problem — you want to handle the problem in main().

The general rule is that only main() should decide if the program should exit. For example, if the stack is empty when we try to pop a value, main may know a way to handle it and can avoid exiting.

Step 1: Get the files  
The only file you need for this lab is [exceptions.cpp](http://www.csee.umbc.edu/courses/undergraduate/202/fall15_marron/labs/lab12/exceptions.cpp). This file contains the main() and the IntStack class. If you've ssh’ed into the GL servers, you can copy the file to your working directory with the command:

cp /afs/umbc.edu/users/c/m/cmarron/pub/cmsc202/exceptions.cpp .

The IntStack class works as follows. When constructed, it allocates a vector of a specified size. This size will not change and limits the number of items that can go in the stack. IntStack also has two functions that are usually included with stacks:push() and pop(). push() adds an item to the stack; this can cause an error if the stack doesn't have any more room. pop() removes the most recently added item; this can cause an error if the stack is empty. The stack keeps track of the top with an unsigned integer called cur\_index. This number stores the index that will be pushed onto next. For example, if cur\_index is 0, the stack is empty and the next pushed item will be placed in index 0 of the vector.

The main() function includes three tests that specifically trigger each error case.

# Step 2: Handle the constructor's error case

An error occurs when a negative size is passed to the constructor. At the top ofexceptions.cpp, write a InvalidSize class to represent this error. This exception should take a string in its constructor and have a GetMessage() accessor that returns the string. Modify the construtor so that it throws this exception with an error message passed to the exception's constructor rather than printing an error message and callingexit(1). Write the code in main() to catch the exception and use cerr to print out the message returned by GetMessage().

Step 3: Handle push's error case  
An error occurs if push() is called when the stack is full. At the top ofexceptions.cpp, write a StackFull class to represent this error. This exception should take a string and an int in its constructor; the string is an error message and the int is the argument to push() that was *not* able to be pushed onto the stack. The class should have a GetMessage() accessor that returns the string and a GetValue()accessor that returns the value. Modify push() so that it throws this exception with an error message and value passed to the exception's constructor *rather than* printing an error message and calling exit(1). Write the code in main() to catch the exception and use cerr to print out the message returned by GetMessage() and the value returned by GetValue().

Step 4: Handle pop's error case  
An error occurs if pop() is called when the stack is empty. At the top ofexceptions.cpp, write a StackEmpty class to represent this error. This exception should take a string in its constructor and have a GetMessage() accessor that returns the string. Throw this exception with an error message passed to the exception's constructor *rather than* printing an error message and calling exit(1). Write the code in main() to catch the exception and use cerr to print out the message returned byGetMessage().

Step 5: Compile and Run

Compile the program. Run it. If you did everything correctly, the program should run and print something like the following:

linux1[8]% ./exc

Cannot create a negative size stack

Push to full stack, value = 5

Pop from empty stack

Completed!