Once again, you will complete this lab individually. As usual, you may discuss the lab with classmates, subject to the empty hands policy.

Assignment: Write your own `Stack` class, to behave as, well, a stack.

This `Stack` is going to hold only integers. I have provided, on Blackboard, the file `Stack.h`, which provides all of the required declarations. You *must* use this header file as is and *must* #-include `Stack.h` in your `Stack.cpp` code file that contains all of the method definitions (implementations).

For completeness, I include `Stack.h` below:

```
#ifndef __STACK_H__
#define __STACK_H__

#include <sstream>
#include <stdexcept> // for runtime_error
#include <iostream>
#include <string>

using namespace std;

class Stack {

private:

  static const size_t INIT_CAP{10};

  size_t size;
  size_t cap;
  int* values;  // array of ints that will hold the values pushed onto the stack

  void resize();

public:
  Stack();
  Stack(const Stack& other);
  Stack& operator=(const Stack& other);
  Stack(Stack&& other)noexcept;
  Stack& operator=(Stack&& other)noexcept;
  ~Stack();

  void push(int element);
  int pop();
  bool isEmpty();
  void print();

  // These are really for testing.  You wouldn't have them
  // in the API for a real Stack class
  size_t getSize(){return size;}
  size_t getCap(){return cap;}
  int* getValues(){return values;}

};

#endif
```

Notes of interest:

- This code, of course, must be in two files: the header file containing the declarations, and the .cpp file that contains the method definitions.
- For handling attempts to `pop()` off an empty stack, `throw` (but do *not* `catch`) an out-of-range exception defined in `<exception>`. For example, you should include code similar to that below:

```
stringstream ss;  // ss acts like cout -- you can "write to a string"
ss << "Attempt to pop() off an empty stack!";
throw runtime_error(ss.str().c_str()); // needs a const char*, not string
```

As with lab 2, you will need to write a tester to test your code. Your tester *will* require a `try-catch` block.

- I will *not* provide a tester file for this lab. Instead, you will supply a tester in a file called `StackTest.cpp`. A portion of your grade will depend on the number and quality of the tests that you implement. Specifically, by providing the header file for you, I have implicitly defined a specification (e.g, you must have a copy constructor, copy assignment operator, etc.). Your test code must hit every one of these requirements, and do so *thoroughly*! I will explain what I mean in lab.
- You do *not* have to implement or test the three getter methods, and in fact you *must not* implement them. I have implemented them inline in the provided header file.
- There must *not* be a `main()` method in `Stack.cpp`!
- The `values` instance variable must point to a *dynamically allocated* array.
- The `resize()` method should double the size of the `values` array each time `resize()` is called.
- You do *not* need to submit a `README` file, but your tester must include documentation with each test describing specifically what you are testing!

Naming:

You should pack up your three source code files (two `Stack` files and the tester) into a tar or gzipped tar ball. If a tar file, it must be named `cmsc240_lab6_netID.tar`. If a gzipped tar ball, you know what it should be named.

Submission:

You know the drill. The the email address for this lab is `lab6.wcojbtbhxmm0iydu@u.box.com`.