Assignment 4: Simple Int Stack

(The Good, The Bad, and The Ugly)

Description: To introduce a simple ADT (abstract data type) and understand the function and purpose of ADTs. To accomplish this, you will study a simple integer stack and code your own, thereby also learning the stack ADT. This is a "toy" problem to introduce stacks. You wouldn't really make an integer stack in real life. In Assignment 5 we will make a more realistic stack which will build on this assignment (so you have to get this one right to do the next one). Also note the purpose of this assignment is to further show you and train you in proper architecture, file structures, and loosely coupled architectures. When this assignment is complete, you should be able to build any ADT properly.

Background

- This assignment assumes you have completed Assignment 3 successfully and have all the necessary tools installed and understand all the processes needed to complete assignments.
- This assignment deals with the stack ADT. The notes from your text for stacks are chapters 6 and 7. Those notes are needed but study them if you find them useful.
- Watch the video lectures on this topic. *Introduction to Stacks. Assignment 4* and *Exception Handling*
- We will be using this site as a reference in this assignment https://www.studytonight.com/data-structures/stack-data-structure (and that code is also attached to this assignment in Blackboard).
  - **The Good**: That page describes a stack ADT in very simple and easy to understand terms. This is good. Stacks are simple, and should not be over complicated.
  - **The Bad**: The code shown has at least one bug and at least one logic error.
  - **The Ugly**: The code for the stack implementation is very bad. It is poor architecture, poor code, logically inconsistent, and violates almost every best practice in software engineering. You should also note this as evidence that ***most*** code you see online is garbage. Never accept what you see online at face value or as "good" code. Always investigate, test, and prove everything to yourself using ***strict*** best practices and proper computer science methodologies and analysis.
- Study these pages carefully:
  - http://katrompas.accprofessors.com/best-practice-procedural-programming
  - http://katrompas.accprofessors.com/best-practice-oop-programming
- Study the notes in the Notes folder on header files if you are still unsure how to use them correctly.

Instructions:

- Follow the Assignment Specific Instructions using this GitHub assignment invite https://classroom.github.com/a/FX4e1oJ1
- The invite will give you one file, **README.md**
- Make, add, and commit a correct **.gitignore**
- Fix the **README.md** to describe this assignment, add it and commit it.
- Make 4 files with the following content, and your comment headers (DO NOT make any other files):
  - **main.h** and **main.cpp**: Put a simple hello world in these files like from Assignment 2, and also add a **#include "stack.h"** to **main.h**.
  - **stack.h** and **stack.cpp**: Put the standard **#ifndef** directives in the header, and put a **#include "stack.h"** in **stack.cpp**
- You now have the stubs for a complete program. Stop and compile and run the code and make sure it works. See below (at the end of this document) for screenshots for what this all should look like at this point (don't forget to put your comment headers in the file).
- When you're sure it works, add and commit the new files with an "initial commit" message.

Ready to code...

- Now you are ready to begin coding. The assignment is to fix the stack from this site. Copy the code from the site (there is also a copy on Blackboard with this assignment). You should compile it and run it separately from your project so you can test it and understand how it works, and also find the bug. Add code to the sample code's main() to use this "bad" stack and see how it works.
- The code has many problems, including at least one bug and one logic error. Find them all, fix them all, and turn in a fully functional integer stack that conforms to all best practices and creates a loosely coupled stack ADT. (see below for hints)
- When you submit your assignment, it should have the following files:
  - **main.h** : all the directives for main.cpp
  - **main.cpp** : your **main()** function and nothing else. main() is simply your test "program." main() is where you will thoroughly test your stack in automated fashion (no user input). Main is your driver file and function. See the notes on Driver Files and Functions on the Best Practices page. Do whatever you need to do in main() to prove your stack works *fully*. You must test every possible operation in every possible combination and explicitly show your stack is fully functional and can handle underflow, overflow, incorrect input, multiple random operations in every combination. Failure to execute a full set of tests that *prove* your stack works *fully* will result in a lowered grade, up to and including a zero. *Developer testing is one of the most serious tasks a developer performs, and a very large part of this class.*
  - **stack.h** : the header file for your stack ADT containing all the directives and prototypes you need for the stack class.
  - **stack.cpp** : your stack implementation
- The only *public* methods your stack should have are pop(), peek(), push(), isEmpty(). DO NOT MAKE ANY OTHER PUBLIC METHODS.

Hints: Here are some of the problems to get you started, but you need to find *all* problems and fix them:

- There is no proper separation of interface from implementation. i.e. the stack does not have a proper header file and cpp file that are separate (neither does main.cpp).
- There is a public attribute (no attribute should ever be public).
- No method should print ← this is a very important rule in ADT programming!
- No method should have more than one return. This applies to **all** methods/functions.
- **push()** should return true/false to indicate success or failure.
- **pop()** is all sorts of wrong (you figure out why and fix it). Hint: You have to indicate an error (underflow) somehow that is logical. There are two ways to do this properly and the currently implemented way is definitely *not* one of them. Why? Think about it and figure out the fix.
- **isEmpty()** is over complicated and all wrong. It should return true/false to indicate empty/not-empty
- The stack is missing a **peek()** operation which simply reports the top value *without* popping it. **peek()** is identical to pop, except that it does not decrement top.
- **main()** as given is useless as a proper test bed.

Grading: Your grade will be graded primarily on exactness to detail and specifications, architecture, and coding logic. You will also be graded on your repo with the following guidelines:

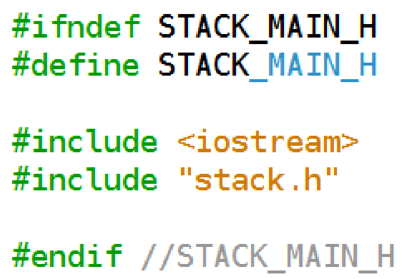
- Failure to commit often, small, and smart will result in an automatic -10% penalty regardless of your code quality.
- Any stray files in your repo will result in an automatic -10% penalty regardless of your code quality.
- Failure to use the correct branch in your repo (main) will result in an automatic -5% penalty regardless of your code quality.

- If you do not have proper comment headers and/or do not use the Write Submission feature (not the comment section), and/or do not submit your link correctly, it's an automatic 10% penalty regardless of your code quality.
- Failure to test your code **thoroughly and exhaustively** will result in a maximum grade of 70% **regardless** of code quality.

Submission: When you are ready for grading, use the write submission feature in Blackboard and submit your repo's link. If you need to fix/change something *after* you submit but *before* it's graded, just fix/change it and push again. **Do not re-submit the assignment before getting a grade**. Only re-submit after you get a grade and want a *re*-grading.

Additional Hints: Here are screenshots of what your stubbed files should look like right before the "***Ready to code***" step...

main.h

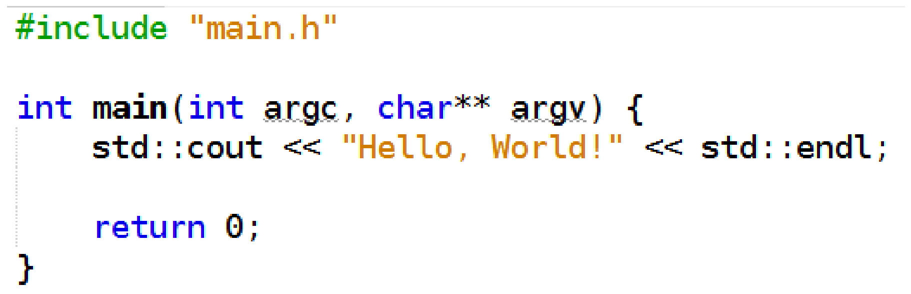


```
#ifndef STACK_MAIN_H
#define STACK_MAIN_H

#include <iostream>
#include "stack.h"

#endif //STACK_MAIN_H
```

main.cpp

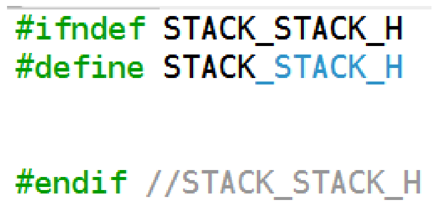


```
#include "main.h"

int main(int argc, char** argv) {
    std::cout << "Hello, World!" << std::endl;

    return 0;
}
```

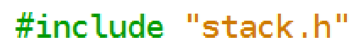
stack.h



```
#ifndef STACK_STACK_H
#define STACK_STACK_H


#endif //STACK_STACK_H
```

stack.cpp



```
#include "stack.h"
```