CS3100/5100: Data Structures and Algorithms

Fall 2019

Programming Assignment #2

1. Overview

The objective of this programming assignment is to practice your C++ programming skills. In this programming assignment, you will implement an abstract data type stack, and test its functionality by writing some sample test programs in main.cpp.

A stack is a container of objects that are inserted and removed according to the last-in-first-out principle. **The header file SStack.h, and Employee.h will be provided to you.** SStack.h is the header file that specifies the interface of the abstract data type stack that you need to implement. You must implement all the functions in SStack.h. **The following basic operations and some other supportive operations must be supported by the ADT Stack.**

**Push -- Add an element onto the stack (to the top of the stack)**

**Pop -- Remove an element from the stack (from the top of the stack)**

**Top -- Look at the element that is on top of the stack (DO NOT remove it)**

**isEmpty -- Check whether the stack is empty or not**

**size: returns the number of elements stored in the stack**

It is important to note that all the push and pop operations are done on the top of the stack. A stack can be implemented in many ways. In this project, we implement a stack using a dynamic array of a given size, i.e., there is a limit on the number of objects the stack can hold. The size of the stack should be specified by the programmer or using a default size while creating the stack.

To test the stack of the string type, a file “Employee-Database.txt” with many employee objects is provided to you. In your main.cpp, you will write code to test all the member functions of the stack ADT. In your main.cpp, you should first create a stack of type string whose capacity is specified by you (should be in the order of 100). Then read the employee objects from Employee-Database.txt and push the employee objects to the stack. The number of employee objects pushed onto the stack should be less than the capacity of the stack. After you pushed the employee objects onto the stack, you should test ALL the other functions in SStack.h. You should print out a message describing the operation you tested.

In the beginning of your main() function, you should open the input file, create a Stack: prompt the user for the capacity of the stack (should be in the order of 100 or larger), and a specified number (should be in the order of 100 or larger) that is less than the capacity of the stack. Then read the specified number of the employee objects from the input file, Employee-Database.txt, and push these employee objects into the stack.

Then create a stack using the copy constructor with the specified capacity.

Then you can create a third stack to test the union( ) function later.

Then in your main() function, you will provide a **user interface** **(MENU)** to test the member functions of the stack ADT and to support the following operations:

1. Push: prompt user for an employee object, and push it onto the top of the stack, then print the number of the elements in the stack to the screen, and only print the top 5 elements in the stack to the screen if the number of elements in the stack is larger than 5.

2. Pop: prompt user for a number, and if the number of the elements in the stack is larger than the number entered by the user, pop the specified number of the elements from the stack, and print the employee objects popped from the stack, and the remaining number of the elements in the stack to the screen. Otherwise, pop all the elements from the stack.

3.Top: print the employee object on the top of the stack and the remaining number of the elements in the stack to the screen.

4. Union: Union two stacks together.

5. Print: Print all the elements in the stack onto the screen.

6.Save: save all the elements in a stack to a file on the disk.

7. Quit: clear all the elements in the stack, so there is no memory leak in your program, and quit.

2. Requirements

The main.cpp file includes code that will test the implementation of your member functions of your Stack ADT.

1. You need to use Employee.h, and Employee.cpp from the first project.
2. In SStack.cpp, you must implement all the functions in SStack.h.

1. Your code should follow the Code Standards posted on Pilot. Your code will be graded according to its correctness, efficiency, organization, and readability.

4. Make sure that each file includes your name and email address in the header comments.

1. Code Submission and Testing
2. The grader will test your source code using visual studio 2019. It is suggested that you test your source code on the computers located in the computer labs in the first floor of Russ Engineering building before you submit your code on Pilot.

2. You must submit an ELECTRONIC COPY of your source program through Pilot before the due date. If for some reason Pilot is unavailable, submit your source code to the instructor Meilin Liu.

1. Submit the source code files including employee.cpp, employee.h, SStack.h, SStack.cpp, main.cpp, a README file. You are recommended to submit a README file to tell the instructor and the grader what they need to know when testing your program.
2. All the submitted project files should have: Course Number / Course Title , Your Name, Prof.’s Name, Date, and the Project Name. If you did not include these required contents in your submitted files, then 5 points will be deducted. It is recommended that your filename include your last name.

5. The instructor or the grader will test your programs under visual studio 2019. It is YOUR responsibility to make your programs workable and runnable under visual studio 2019.

6. The programming assignment is individual. You must do the project by yourself. If you allow others to copy your programs or answers, you will get the same punishment as those who copy yours.

7. Be honest. If your program does not work, SAY SO in your README. The instructor will run your programs with ANY DIFFERENT input files. If your program does not work, but you “claim” yours work, you will have severe penalty!