Data Class, Stack, Dynamic Memory Mangement: Run-time Stack Simulation

The purpose of this assignment is to demonstrate your ability to design and implement a data class that has dynamic content, and to manage data records on a dynamic, resizable, contiguous stack, and to implement your design intelligently in separate header and source files.

This project requires extending the previous project to incorporate dynamic memory allocation in two ways:

First, each allocation record will now use two dynamically allocated arrays to store the values of the parameter and local variables. Those arrays must be precisely the correct size for the number of parameters and locals, respectively.

Second, the underlying physical structure of the stack will be changed. The stack will still be array-based, but that array will be allocated dynamically when the stack object is created, and the stack object will automatically resize the array. Initially, the stack array will be of dimension 1. If a push operation is needed, and the stack is full, then the stack array must be doubled in size. If a pop operation is needed, and the result is that the stack is less than 1/3 full, then the stack array must be halved in size. (Of course, all that is subject to integer rounding.)

The behaviors specified in the previous project are still required. The only change is that there is no longer a limit on the number of parameters or local variables that a simulated function may have.

On startup, your program will initialize the necessary data structures then read commands from a script file and log the results of processing those commands to an output file.

Script file:

This project will involve only one input file, containing the commands that are to be processed. The script file, named `"RTStackScript.txt"`, contains a sequence of commands, one per line. The commands are precisely the same as for the previous project, and their specifications are not repeated here.

All of the sample scripts used for the previous project are still valid for this project.

Log file:

Your program will write all of its output to a log file, named `"RTStackLog.txt"`. The contents of the log file should be similar to the previous project. However, because this project will not be auto-graded, you now have considerable leeway in formatting and setting text for the feedback messages. The only explicit addition is that your output for a display command must include the current dimension of the stack array.

Evaluation:

Your submitted program will be evaluated by one of the TAs, at a demo, which you will schedule. Details about scheduling the demos will be announced later. The TA will also be evaluating your implementation for correctness of operation, as well as for design quality and documentation.

Your implementation must meet the following requirements:

- Choose descriptive identifiers when you declare a variable or constant. Avoid choosing identifiers that are entirely lower-case.
- Use named constants instead of literal constants when the constant has logical significance.
- Use C++ streams for input and output, not C-style constructs.
- Use C++ `string` variables to hold character data, not C-style character pointers or arrays.
- In the procedural part of your implementation, you must make appropriate use of functions.

- None of your functions can include more than 30 executable statements. An executable statement is one that causes something to happen. Comments, constant and variable declarations (even if they initialize) do not count as executable statements.
- When you pass an array to one of your functions, use pass by constant reference unless pass by reference is logically necessary. (Remember that for array parameters, pass by reference is the default.)
- You must use dynamic allocation as described in this specification.
- The stack must be implemented as a C++ class (templates are banned, including all STL containers).
- Each class must be implemented using a header file containing the class declaration and an associated source file containing the implementations of the class functions.
- Any class that allocates memory dynamically must include a destructor to deallocate that memory, and a copy constructor and assignment operator to provide correct deep copy support.

You must implement the activation record type as a C++ class. The class should have appropriate data and function members. Try to provide a useful class interface, but also practice good information hiding.

Read the *Programming Standards* page on the CS 1704 website for general guidelines. You should comment your code in some detail. In particular:

- You should have a header comment identifying yourself, and describing what the program does.
- Every constant and variable you declare should have a comment explaining its logical significance in the program.
- Every major block of code should have a comment describing its purpose.
- Every function implementation must have a header comment. The format of the header comment is described in the course notes, as well as on the *Programming Standards* page on the course website.
- Every class declaration must have a header comment, describing the purpose of the class and anything a client would need to know about the limitations of the class.
- Adopt a consistent indentation style and stick to it.

Understand that the list of requirements here is not a complete repetition of the *Programming Standards* page on the course website. It is possible that requirements listed there will be applied, even if they are not listed here.

Submitting your program:

You will submit this assignment, as a standard zipped archive, to the Curator System (read the *Student Guid*e). Do not use any other format for the archive file. However, this assignment will not be graded automatically. Follow the instructions on the website for submitting projects that are not auto-graded. Be sure to note that more than just your source files are required, and be sure to not submit any extra files.

You will be allowed up to three submissions for this assignment. Only one should be necessary, but this will allow you a chance to fix errors you may find after making your first submission.

The *Student Guide* and submission link can be found at: `http://www.cs.vt.edu/curator/`

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement provided on the course website in the header comment for your program.

Failure to include the pledge statement may result in a substantial grade penalty.