Questions on PA6?
Class recs
study session tn

CptS 122 – Data Structures

Lab 9: Developing a Stack Class Template in C++

Assigned: Week of March 18, 2024
Due: At the end of the lab session

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Design, implement and test a Stack class template in C++
- Apply an array implementation for a Stack
- Compare and contrast value classes versus container classes
- Apply and implement *overloaded* functions and operators

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Analyze a basic set of requirements for a problem
- Create test cases for a program
- Design, implement and test classes in C++
- Declare and define *constructors*
- Declare and define *destructors*
- Compare and contrast *public* and *private* access specifiers in C++
- Describe what is an *attribute* or data member of a class
- Describe what is a *method* of a class
- Apply and implement *overloaded* functions
- Distinguish between pass-by-*value* and pass-by-*reference*
- Discuss *classes* versus *objects*

III. Overview & Requirements:

This lab, along with your TA, will help you navigate through designing, implementing, and testing a Stack class template in C++. It will also help you with understanding how to apply a stack object to an application.

Labs are held in a "closed" environment such that you may ask your TA questions. Please use your TAs knowledge to your advantage. You are required to move at the pace set forth by your TA. Have a great time! Labs are a vital part to your education in CptS 122 so work diligently.

Tasks:

NOTE: Parts of this lab are courtesy of Jack Hagemeister.

For this lab you will develop a Stack class template to evaluate postfix expressions. You will start with the provided code.

Tasks:

Recall with Big-O we want..

- **to determine central unit of work by considering the operations applied in the algorithm**
- **to express unit of work as function of size of input data: How quickly does amount of work grow as size of input grows?**
- **classify algorithms according to how their running *time* and/or *space* requirements grow as input size grows**
- **the analysis of an algorithms is extremely useful when comparing algorithms that solve the same problem!**

0. Provide a Big-O time and space analysis for each!

a. Is it more efficient to delete the last node in an array or linked implementation of a list?
b. Is it more efficient to delete the first node in an array or linked implementation of a list?
c. Is it more efficient to delete a node, in general, in an array or linked implementation of a list?
d. Is it more efficient to insert a node at the end in an array or linked implementation of a list?
e. Is it more efficient to insert a node at the front in an array or linked implementation of a list?
f. Is it more efficient to insert a node, in general, in an array or linked implementation of a list?
g. Is it more efficient to access a node, in general, in an array or linked implementation of a list?

Recall, a class template enables for generic programming. The template provides a stencil or pattern for a class. We can provide different specializations from the template. One way to view this is that the template is the like the stencil from which shapes are traced. Each shape traced could be specialized with different colors, textures, etc.

Starting with the Stack template code found at https://eecs.wsu.edu/~aofallon/cpts122/labs/StackTemplateLab.zip:

1. **Review Stack Class Template Code in C++.**

   First, define what is a stack? What is the purpose of using stacks?

   Next, review and discuss the skeleton code with your teammates. Notice that all of the template code is located only in the header (.h) files, sometimes you will see the use of (.hpp) files instead of (.h)! If you place the template code in .cpp files, you will encounter linker errors! The code currently compiles. **Please also realize that we are using an underlying array for our stack. We are not using a linked list structure!!!**

   Some of the functions are considered "stubs" because they return values, but have not been completely implemented. You will write the code for the "stubs" in task 2. Are there any functions missing from the stack implementation? You may not know until you start implementing the functions for tasks 3 and 4! Feel free to add more functions as you see fit!

2. **Implement Stack Code in C++.**

   Second, start to fill in the code for each of the "stub" functions (found in the Stack class only) and any other functions that you have added to the Stack. The functions that you must implement include:
   - isEmpty () – what is the Big-O of this algorithm?
   - push () - what is the Big-O of this algorithm?
   - pop () - what is the Big-O of this algorithm?
   - peek () - what is the Big-O of this algorithm?

   As you complete a function, test it. As part of the project you will notice that a test class has been provided. Please use this class to test your functions!

3. **Postfix Evaluation with a Stack.**

   Build a function that will complete a postfix evaluation of a given input string using your stack implementation. What is the type of item that will be stored in your stack? An integer for this part! Note: each operand is a *single* digit only. The function should return the value of the expression. Test your implementation on the following inputs; you can assume that the inputs are valid.

   ```
   6 2 + 5 * 8 4 / -
   5 6 2 * + 9 /
   4 5 / 6 8 * -
   ```

   Algorithm for Evaluating Postfix Expressions

   - **What is the Big-O of the following algorithm?**

1. Let **S** be an empty stack.
2. If there is no character to read, then the postfix expression is malformed. This error in the input string should be reported and the program should give up on that string.
3. Read the next character and call it **c**.
4. If **c** is the symbol '='
    1. If **S** is empty, then the postfix expression is malformed. This error in the input string should be reported and the program should give up on that string.
    2. If **S** has more than one element, then the postfix expression is malformed. This error in the input string should be reported and the program should give up on that string.
    3. If **S** has exactly one element, call it **e**. The value of the postfix expression is **e**. Return **e**.
5. If **c** is a digit, then push **c** onto **S** and go to Step 2.
6. If **c** is a binary operator, call it **o**
    1. If **S** does not have at least two elements, then the postfix expression is malformed. This error in the input string should be reported and the program should give up on that string.
    2. Otherwise, pop two elements off of **S**. Call the first one popped $\mathbf{s_2}$ and the second one popped $\mathbf{s_1}$. I.e., $\mathbf{s_1}$ was below $\mathbf{s_2}$ in **S** prior to the popping.
    3. Apply the operator, **o**, to $\mathbf{s_1}$ and $\mathbf{s_2}$ (in that order) to obtain a value called **v**.
    4. push **v** onto **S**.
    5. Go to Step 2.

4. Convert infix to postfix.

- **What is the Big-O of the following algorithm?**

Using your stack class template implementation, create a function that will convert a given infix expression (string) to a postfix expression. In this case the stack must contain character items (representative of the operator)! Recall, each operand is a *single* digit only.

See the page https://eecs.wsu.edu/~aofallon/cpts122/labs/Infix2Postfix.pdf for an explanation and algorithm.

IV. Submitting Labs:

- You are not required to submit your lab solutions. You should keep them in a folder that you may continue to access throughout the semester.

V. Grading Guidelines:

- This lab is worth 10 points. Your lab grade is assigned based on completeness and effort. To receive full credit for the lab you must show up on time, work in a team, complete 2/3 of the problems, and continue to work on the problems until the TA has dismissed you.