

Homework 1

Assigned Sept 4th

Due: Sept 18th

The problem:

1. Implement a character vector ADT class using C++

You need to have the following functions related to the vector class.

- Insert(X, I): add an element X at index I.
- Delete(I): delete an element at the index I.
- Push_front(X): insert an element X at the front of the vector
- Pop_front(): delete the element at the front and return it
- Push_end(X): insert an element X at the end of the vector
- Pop_end(): delete the element at the end and return it
- Size(): return the number of elements in the vector
- isEmpty(): return true if the vector contains no element and false, otherwise.
- getElement(I): return the element at index I
- print(): print out the elements of the vector

Your code should not assume the maximal number of elements that can be inserted into a vector. Basic error handling should be there.

2. Based on YOUR vector class, implementing an character STACK ADT using C++

You need to have the following functions related to the STACK class.

- Push(X): insert an element X at the top of the stack
- Pop(): delete the element at the top of the stack and return it
- Size(): return the number of elements in the stack
- isEmpty(): return true if the stack contains no element and false, otherwise.
- Print(): print out the elements in the stack from top to bottom.

3. Given an arithmetic expression, based on YOUR STACK ADT, develop a code to convert the arithmetic expression to a postfix expression as we discussed in the classroom.

Example: input is (a+b)*c, the output should be ab+c*

How to test your code:

You should name your executables as “vadt” for the vector ADT class, “sadt” for the stack ADT class, and “a2p” for converting an arithmetic operation to the postfix expression.

For the vector class executable vadt, it should accept an in-line parameter, which is the name of the testing file. For example, to run vadt, you should type:

```
$vadt vtest
```

where vtest is a testing file. You do not necessary name your test file as vtest.

The testing file for the vector ADT class is a set of commands, one at each line. Each command is composed of the name of the related function and the related parameters (without parenthesis and comma). For example: the following is a valid testing file for the vector class:

```
Insert a 3
Delete 2
Print
Push_front b
Pop_end
Print
```

Notice that in this case, I will never test functions such as `Size()`, `isEmpty()`. You need to test these functions internally.

For the stack class executable `sadt`, it should accept an in-line parameter, which is the name of the testing file. The testing file for the stack ADT class is a set of commands, one at each line. Each command is composed of the name of the related function and the related parameters (without parenthesis and comma).

For the executable “`a2p`”, it should expect an in-line parameter, which is the name of the testing file. The testing file for the `a2p` executable is a list of arithmetic operations (in infix format). Your executable should convert them to the related postfix expression.

What to hand in:

- (1) Source codes with a makefile. If you do not use makefile before, the following is a good tutorial: <http://frank.mtsu.edu/~csdept/FacilitiesAndResources/make.htm>. Source code needs to be properly commented.
- (2) A report with the following components:
 - a. A brief discussion of the data structure that you use to implement the first two ADTs. Justify your answer.
 - b. A brief discussion of the algorithm that you use to solve the problem 3.
 - c. Show your test files and your testing results. You may use typescript to capture unix output. If you are not familiar with script and typescript, see <http://www.tech-faq.com/capture-unix-terminal-session.shtml>.

For example in Unix/Linux environment, assuming you have an executable named “`vadt`” and a test file “`tvadt`”, you may recode the testing results in the following way:

```
$script          #start scripting
$cat tvadt       #show testing file contents
$vadt tvadt      #run your code
```

```
^D (ctrl-D)      #stop scripting
cat typescript   #see the recorded results
```

d. Briefly discuss error handlings that you have implemented

Grading:

- Correctness of the code (whether it can compile, whether it produces the correct output when correct input files are provided). 50%
- Style of the coding including comments 10%
- Robust of the code with error handling 15%
- Report (style, completeness, clearness) 25%