PONDER 02 : INFIX TO POSTFIX

Due Saturday at 11:59 PM MST

The second programming assignment will be to implement the stack data structure and use it to convert an infix mathematical expression (4 + 8 \* 3) into a postfix one (4 8 3 \* +).

Stack

Create a class encapsulating the notion of a stack. This will work exactly like the [std::stack](http://www.cplusplus.com/reference/stack/stack/)class. Of course, any data-type will need to be supported, so your class will be a template class. It will need to be defined in its own header file (stack.h). The class name must be Stack and will need to support the following operations:

* **Constructors**: Default constructor (create a stack with zero items in it), a non-default constructor (taking a capacity value as a parameter), and the copy constructor. If there is insufficient memory to allocate a new buffer, then the following exception is thrown:  
  ERROR: Unable to allocate a new buffer for Stack.
* **Destructor**: When finished, the class should delete all the allocated memory.
* **operator=**: Assignment operator. This method takes a Stack as a parameter and copies all the elements to this. If the current buffer size is sufficient, no allocation is made. If the current buffer size is not sufficient, enough space is allocated to accomodate the new data. If there is insufficient memory to allocate a new buffer, then the following exception is thrown:  
  ERROR: Unable to allocate a new buffer for Stack. It also returns \*this by-reference as all assignment operators should.
* **empty()**: Test whether the stack is empty. This method takes no parameters and returns a Boolean value.
* **size()**: Return the stack size. This method takes no parameters and returns an integer value.
* **capacity()**: Return the number of elements that can be held in the currently allocated storage. This method takes no parameters and returns an integer value.
* **clear()**: Clear the contents. This method takes no parameters and returns nothing. Note that you do not need to free the allocated memory; just set the size member variable to zero.
* **push()**: Adds an item to the container. This method takes a single parameter (the item to be added to the end of the stack) and has no return value. Note that if the stack is full, then the capacity will be doubled. In the case of an allocation error, the following c-string exception will be throw:  
  ERROR: Unable to allocate a new buffer for Stack
* **pop()**: Removes an item from the end of the stack, serving to reduce the size by one. Note that if the stack is already empty, the following c-string exception will be thrown:  
  ERROR: Unable to pop from an empty Stack
* **top()**: Returns the item currently at the end of the stack. This item is returned by-reference so the last item can be changed through the top() method. If the stack is currently empty, the following exception will be thrown:  
  ERROR: Unable to reference the element from an empty Stack

Note that the only way to access elements in a stack is through the top() method. This means that there is no iterator for Stack.

Driver Program

A driver program is provided. This file (/home/cs235/week02/week02.cpp) will pound-include your header file (stack.h) and expect a template class Stack to be defined therein. This program will exercise your class, filling the container with user input and displaying the results. As with previous assignments, a makefile will be provided (/home/cs235/week02/makefile). A header file (infix.h) will be provided and an implementation file (infix.cpp) as well. You will need to create a stack header file (stack.h).

Infix to Postfix

In addition to passing the four test functions for the Stack class, you will also need to use theStack class to implement a program to convert an infix expression (4 + 8 \* 3) into a postfix expression (4 8 3 \* +). There is a discussion on this in chapter 7.5 of the textbook. Note that you will need to handle the following operators:

1. () The parentheses are the highest level of the order of operations.
2. ^ The exponent operator comes immediately after the parentheses. Note that the algorithm in the textbook cannot handle the exponent operator properly. You will need to handle this case.
3. \* / % Multiplication, division, and modulus are after the exponent operator. They are at the same level of the order of operations.
4. + - Addition and subtraction are after multiplication, division, and modulus.

Your convertInfixToPostfix() function will prompt the user for an infix equation and display the postfix version of the same. The following is an example of the output, with underline text as user input:

Enter an infix equation. Type "quit" when done.  
infix > 2 + 5  
postfix: 2 5 +  
  
infix > (5.0 / 9.0) \* (fahrenheit - 32)  
postfix: 5.0 9.0 / fahrenheit 32 - \*  
  
infix > 1 + 2 \* 3 ^ 4  
postfix: 1 2 3 4 ^ \* +  
  
infix > quit

A few hints that may come in handy when implementing this part of the assignment:

* There is a tab character immediately before postfix.
* When displaying the postfix notation, there is a space before every operator, variable, and number.
* Variable names with more than one letter must be supported. According the syntax for the C Programming Language, variable names are made up of letters and digits; the first character must be a letter. The underscore counts as a letter.
* Numbers may include more than one digit. A number may start with a decimal and may not have more than one decimal.
* You may wish to work through a few examples in chapter 7.5 of the textbook. Do not copy the code on page 381, however. It will not handle all the levels in the order of operations, and it will not handle all identifiers correctly.
* You may wish to create a class to turn the input line into a collection of tokens in an effort to simplify this problem.

To get full credit for this program, you must use your own Stack class. If your class does not work, use the standard template library std::stack from #include <stack>. If you do this, you will loose points for the first half of the assignment, but not the second.

Common Mistakes

The most common mistakes students make with this assignment include the following:

* **Copying the code out of the textbook**. You should never copy code directly out of the textbook or off of the internet. Any ideas you get from other implementations should be cited in an appropriate comment block. That being said, the textbook solution does not handle the exponent in the order of operations.

Test Bed

The testBed for this assignment is:

testBed cs235/week02 week02.tar

You can also run testBed on the executable:

testBed cs235/week02 a.out

Of course, you will need to pass testBed to get full credit on the assignment.

Submitting

You will submit this assignment using the Linux submit command. Please:

1. Create a TAR file built from the makefile, which will contain five files:
   * makefile: Directly from /home/cs235/week02/makefile except with your edits on the comment block.
   * stack.h: Your class definition for Stack.
   * infix.h: Containing the prototype for convertInfixToPostfix() and any other function you may need.
   * infix.cpp: Implementation for all the functions and classes necessary for the infix to postfix conversion.
   * week02.cpp: Unmodified from /home/cs235/week02/week02.cpp.
2. Run the program by hand a few times through all four test cases as well as the infix-to-postfix converter.
3. Verify your solution with testBed.
4. Submit your file using the submit command. The submit command will prompt you for your instructor, the class (cs235), and the assignment (week02). You submit your file with:

submit week02.tar

Your program will be graded according to the following rubric:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Exceptional 100% | Good 90% | Acceptable 70% | Developing 50% | Missing 0% |
| Stack Interface  20% | The interfaces are perfectly specified with respect to const, pass-by-reference, etc. | week02.cpp compiles without modification | All of the methods in Stack match the problem definition | Stack has many of the same interfaces as the problem definition | The public methods in the Stack class do not resemble the problem definition |
| Stack Implementation  20% | Passes all four Stack testBed tests | Passes three testBed tests | Passes two testBed tests | Passes one testBed test | Program fails to compile or does not pass any testBed tests |
| Infix to Postfix  30% | The code is elegant and efficient | Passes the Infix to Postfix testBed test | The code essentially works but with minor defects | Elements of the solution are present | The Infix to Postfix code was not attempted |
| Code Quality  20% | There is no obvious room for improvement | All the principles of encapsulation and modularization are honored | One function is written in a "backwards" way or could be improved | Two or more functions appears "thrown together." | The code appears to be written without any obvious forethought |
| Style  10% | Great variable names, no errors, great comments | No obvious style errors | A few minor style errors: non-standard spacing, poor variable names, missing comments, etc. | Overly generic variable names, misleading comments, or other gross style errors | No knowledge of the BYU-I code style guidelines were demonstrated |

Please make sure to fill out the program header in the makefile