Programming Assignment #6

Postfix Expression Calculator and the DigitQueue

CS 2308.255 + CS5301 Spring 2020
Instructor: Jill Seaman

Due: Wednesday, 4/22/2020: upload electronic copy by 11:59pm!

Postfix Expression Calculator:

Write a program that uses a Stack to evaluate postfix expressions. A postfix expression is a string containing integer literals and arithmetic operators (+, -, *, /). Postfix is a notational system where the operator follows the arguments. For example, "1 2 +" would be postfix notation for adding the numbers 1 and 2. "12 3 -" is postfix notation for 12-3. "1 2 + 18 4 - *" is postfix notation for (1+2) * (18-4).

A stack can be used to evaluate a postfix expression as follows:

- Read the next string from the expression (I recommend putting the expression into a `stringstream`, named `ss`, so that we can use `ss>>str` to read the next sequence of characters into str until a space is encountered).
- If the next string contains a number, convert it to an integer using stoi(str) and push it on to the stack.
- If the next string contains an operator, say "+", then pop the next two integers off of the stack (x and y) and combine them using the operator (y+x) and put that value onto the stack.

When the end of the original expression is encountered (when `ss>>str` fails), the value of the expression can be found on top of the stack.

The implementation of the IntStack from class is provided in IntStack.h and IntStack.cpp. Write a driver, Calculator.cpp that contains the following standalone function: `int evalPostFix(string s)`

This function should take a string s containing an arithmetic expression in postfix notation and use an instance of the IntStack to evaluate it, and return the resulting value. In your main function, test your function on (at least) these examples:

`"8 4 /"` (should be 2)
`"18 4 *"` (should be 72)
`"1 2 + 8 4 - *"` (should be 144)
`"8 4 1 2 + - *"` (should be 8)

Note: if the expression is not a postfix expression, output an error message and return 0. Hint: check to make sure the stack is empty before you pop. If not, it's not a properly formed postfix expression, like 8 + 4.

DigitQueue:

Implement a queue using a C++ integer variable to contain the entire contents of the queue. In other words, instead of an array or linked list, one integer variable is used as a queue that can store several values. You should assume that the queue elements are integers in the range 0 to 9.

I have provided DigitQueue.h containing the DigitQueue class declaration and DigitQueueDriver.cpp, a driver to test your DigitQueue. You must define the member functions for the DigitQueue.

- **constructor**: initialize the member variables appropriately.
- **enqueue(x), dequeue(), isEmpty()** the standard queue functions.
- **print()** neatly prints the contents of a queue on the screen, in a column, one element per line, front to rear. If the queue is empty, print() has no output at all. The queue itself must be in the same state after the print() operation is executed as it was before. The same elements will be in the queue, in the same order.

I recommend doing it this way:
`q.enqueue(1); q.enqueue(2); q.enqueue(3);`
will store `321` in the queue variable.

Some hints (for defining enqueue and dequeue):
- `321/10 = 32`
- `321%10 = 1`
- `7*10^3 + 321 = 7321` // you can use pow(10,3) to get 10^3. include <cmath>

NOTES:

- All provided code files are on Canvas under Files>PA#6ProvidedCode
- The files you submit must compile with the provided files, without requiring an changes.

Logistics:

For this assignment you need to submit only 2 files: **Calculator.cpp** and **DigitQueue.cpp**. Do not change the names of the files.

Submit an **electronic copy** using the Assignments tool on the Canvas website for this class (canvas.txstate.edu).

See the assignment policy on the course website (cs.txstate.edu/~js236/cs2308) for more details, including late deadlines and penalties.