ECE 71/191T – Data Structures and Algorithms

Dr. Gregory R. Kriehn, Fresno State
C++ Homework Assignment: Chapter 17

Code Due By: Midnight on Wed, Mar 29
Write-up Due By: Class on Thu, Mar 30

HOMEWORK #33 – Shunting-Yard and Postfix Algorithms

Implement the linked-list based versions of a stack and queue as template classes, as outlined in Chapter 17 of Malik. If you wish to save time, derive your template classes from the **`unorderedLinkedListType`** class as implemented in Chapter 16. Otherwise, create your stack and queue classes as derived classes from the **`stackADT`** and **`queueADT`** abstract classes (which use virtual functions as outlined on pages 1168 and 1217), respectively, as the basis for creating your **`linkedStackType`** and **`linkedQueueType`** classes. Verify the operation of your stack and queue template classes before proceeding further.

In computer science, the shunting-yard algorithm is a method for parsing mathematical expressions specified in infix notation (standard algebraic notation). The algorithm was invented by Edsger Dijkstra, and can be used to produce a postfix notation string from infix notation. Edsger named the algorithm the "shunting-yard" algorithm because its operation resembles that of a railroad shunting yard.

The algorithm is stack based. For the conversion, there are two text variables (strings) – the input and the output, which should be implemented as linked-list queues. There is also a stack that holds operators not yet added to the output queue. To convert an expression, the program reads each symbol in order and shunts the operators and operands into postfix notation, which is stored in the output queue.

- While there are tokens to be read:
  - Read a token.
  - If the token is a number, then push it to the output queue.
  - If the token is a function token, then push it onto the stack.
  - If the token is a function argument separator (e.g., a comma):
    - Until the token at the top of the stack is a left parenthesis, pop operators off the stack onto the output queue. If no left parentheses are encountered, either the separator was misplaced or parentheses were mismatched.
  - If the token is an operator, o1, then:
    - while there is an operator token o2, at the top of the operator stack and either
      - o1 is left-associative and its precedence is less than or equal to that of o2, or
      - o1 is right associative, and has precedence less than that of o2,
        - pop o2 off the operator stack, onto the output queue;
    - at the end of iteration push o1 onto the operator stack.
  - If the token is a left parenthesis (i.e. "("), then push it onto the stack.

    - If the token is a right parenthesis (i.e. ")"):
        - Until the token at the top of the stack is a left parenthesis, pop operators off the stack onto the output queue.
        - Pop the left parenthesis from the stack, but not onto the output queue.
        - If the token at the top of the stack is a function token, pop it onto the output queue.
        - If the stack runs out without finding a left parenthesis, then there are mismatched parentheses.
- When there are no more tokens to read:
    - While there are still operator tokens in the stack:
        - If the operator token on the top of the stack is a parenthesis, then there are mismatched parentheses.
        - Pop the operator onto the output queue.
- Exit

To analyze the running time complexity of the algorithm, one only has to note that each token will be read once; each number, function, or operator will be printed once, and each function, operator, or parenthesis will be pushed onto the stack and popped off the stack once – therefore, there are at most a constant number of operations executed per token, and the running time is *O(n)*.

Write a program that prompts the user to input an infix expression, such as:

5 + ((1 + 2) * 4) – 3,

and converts the output to postfix notation. The corresponding output in this case will be:

5 1 2 + 4 * + 3 –

Likewise, the expression:

3 + 4 * 2 / (1 – 5) ^ 2 ^ 3

where ^ represents the power operator, results in postfix notation:

3 4 2 * 1 5 – 2 3 ^ ^ / +

Once your shunting-yard algorithm is working, implement the postfix algorithm to perform the algebraic computation on the output queue:

- While there are input tokens left
    - Read the next token from input.
    - If the token is a value
        - Push it onto the stack.
    - Otherwise, the token is an operator (operator here includes both operators and functions).

        - It is already known that the operator takes n arguments.
        - If there are fewer than n values on the stack
            - (Error) The user has not input sufficient values in the expression.
        - Else, Pop the top n values from the stack.
        - Evaluate the operator, with the values as arguments.
        - Push the returned results, if any, back onto the stack.
- If there is only one value in the stack
    - That value is the result of the calculation.
- Otherwise, there are more values in the stack
    - (Error) The user input has too many values.

As indicated in class, the postfix expression can be evaluated via a stack. Your program should therefore take the output of the shunting-yard algorithm and input it into the postfix algorithm to calculate the result.

Both the shunting-yard algorithm and postfix algorithm should be implemented via user-defined functions **`shuntingYard()`** and **`postfix()`**.

In the end, your program should prompt the user to enter an infix expression, convert the infix expression to postfix notation via the shunting-yard algorithm, print the postfix notation to the screen, calculate the answer using the postfix algorithm, and print the calculated result to screen. The program should be able to handle extra white space between operands and operators, in addition to the standard mathematical operators: '`^`', '`*`', '`/`', '`+`', '`-`', as well as parentheses '`(`' and '`)`'.

```
Please Enter an Infix Expression:  5 + ((1 + 2) * 4) – 3
Postfix Notation:  5 1 2 + 4 * + 3 –
Result:  14
```

For additional background information, see:

https://en.wikipedia.org/wiki/Shunting-yard_algorithm
https://en.wikipedia.org/wiki/Reverse_Polish_notation