**Montgomery College**

**CMSC 204**

**Assignment 2 Design**

1.      InvalidNotationFormatException – occurs when a Notation format is incorrect

2.      StackOverflowException – occurs when a top or pop method is called on an empty stack.

3.      StackUnderflowException – occurs when a push method is called on a full stack.

4.      QueueOverflowException – occurs when a dequeue method is called on an empty queue.

5.      QueueUnderflowException – occurs when a enqueue method is called on a full queue.

**Pseuodocode:**

***Infix expression to postfix expression:***

1. If the current character in the infix is a space, ignore it.
2. If the current character in the infix is a digit, copy it to the postfix solution queue
3. If the current character in the infix is a left parenthesis, push it onto the stack
4. If the current character in the infix is an operator,
   1. Pop operators (if there are any) at the top of the stack while they have equal or higher precedence than the current operator, and insert the                         popped operators in postfix solution queue
   2. Push the current character in the infix onto the stack
5. If the current character in the infix is a right parenthesis
   * 1. Pop operators from the top of the stack and insert them in postfix solution queue until a left parenthesis is at the top of the stack, if no left parenthesis-throw an error
     2. Pop (and discard) the left parenthesis from the stack
6. When the infix expression has been read, Pop any remaining operators and insert them in postfix solution queue.

***Postfix expression to infix expression:***

1. If the current character in the postfix is a space, ignore it.
2. If the current character is an operand, push it on the stack
3. If the current character is an operator,
   1. Pop the top 2 values from the stack. If there are fewer than 2 values throw an error
   2. Create a string with 1st value and then the operator and then the 2nd value.
   3. Encapsulate the resulting string within parenthesis
   4. Push the resulting string back to the stack
4. When the postfix expression has been read:
5. If there is only one value in the stack – it is the infix string, if more than one value, throw an error

***Evaluating a postfix expression***

1. If the current character in the postfix expression is a space, ignore it.
2. If the current character is an operand or left parenthesis, push on the stack
3. If the current character is an operator,
   1. Pop the top 2 values from the stack. If there are fewer than 2 values throw an error
   2. Perform the arithmetic calculation of the operator with the first popped value as the right operand and the second popped value as the left operand
   3. Push the resulting value onto the stack
4. When the postfix expression has been read:
5. If there is only one value in the stack – it is the result of the postfix expression, if
6. more than one value, throw an error

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case #** | **Input** | **Actual Input** | **Expected Output** | **Actual Output** | **Did the test pass?** |
| 1 | \*click ‘infix to postfix’ | \*click ‘infix to postfix’ | Hide postfix | Hide postfix | Yes |
| 2 | \*click ‘postfix to infix’ | \*click ‘postfix to infix’ | Hide infix | Hide infix | Yes |
| 3 | ((3\*(5+4))+2) | ((3\*(5+4))+2) | 354+\*2+ | 354+\*2+ | Yes |
| 4 | 354+\*2+ | 354+\*2+ | 29.0 | 29.0 | Yes |