COMPSCI 221: Programming Assignment 3 #2

Due on Tuesday, March 21, 2017

Leyk

Robert Quan

Problem 1

Abstract
In this assignment, we are using the Doubly Linked List ADTs that are implements as a templated Queue and Stack data structure. The Doubly Linked List consists of the header and trailer pointer and nodes that have points to the previous and next element in the list. The node contains the templated element which in this assignment is a double and a string. The algorithms used here are contained within the Parser class and the Evaluator class.

1) The classes that I choose to implement were the LinkedQueue, LinkedStacked, Evaluator and Parser classes. By building upon the templated LinkedQueue and LinkedStack classes, I created the parser class which is used to convert an infix expression that the user inputs into a postFix expression that the Evaluator class will ultimately evaluate. Along the way, the Evaluator class asks the user for numerical entries if the user included variables for the infix notation. These are all called by the main file which displays the user menu and requests that the user submit the equation via keyboard input.

The reason why I choose these classes was due to the example presented in class and the logic that was behind the associated input priority and stack priority of the operators $+, -, *, /, ^\prime (,)$. The Parser class correctly sorts the input and creates an postFix notation of the input.

2) **Parser**
The Parser class contains the following **Private** members:

- $LinkedQueue < std :: string >$ **postFixQueue;** - A queue that holds the postFix notation
- $LinkedQueue < std :: string >$ **inFixQueue;** - A queue that holds the inFix notation
- $LinkedQueue < std :: string >$ **tokenize(std::string input);** - Takes a string as an input and converts the string into nodes that are pushed onto a LinkedQueue. If the parenthesis are not balanced, then an error is thrown and the program closes. This function returns the inFix notation. This function is
$$\boxed{\mathcal{O}(n)} \tag{1}$$
A linear complexity.
- $LinkedQueue < std :: string >$ **toPostFix**$(LinkedQueue < std :: string > in)$**;** - Based on the order of precedence on the slides, this function uses the two data structures, Stack and Queue, and correctly sort and create a postFix notation of the input expression. If the user used variables in their expression, the variables are still present in the postFix notation. This function is
$$\boxed{\mathcal{O}(n)} \tag{2}$$
A linear complexity.

And the following **Public** members:

- $Parser(std :: strings)$ $\{tokenize(s);\}$**;** - The Parser constructor which call the function tokenize.
- $LinkedQueue < std :: string >$ **getPostFix();** - Returns the private variable PostFixQueue. This function is
$$\boxed{\mathcal{O}(1)} \tag{3}$$
A constant complexity.

- *int* **inputPriority(string temp);** - Assigns a specific value to the input operator from the string. This value is then later used by the parse to correctly place the operator in the postFix expression. This function is

$$\boxed{\mathcal{O}(n)} \tag{4}$$

A linear complexity.

- *int* **stackPriority(string temp);** - Assigns a specific value to the stack operator from the string. This value is then later used by the parse to correctly place the operator in the postFix expression. This function is

$$\boxed{\mathcal{O}(n)} \tag{5}$$

A linear complexity.

- *bool* **isBalanced();** - This function counts the instances of the left prenthesis and right parentheses and returns a bool if the expression is balanced or not. This function is

$$\boxed{\mathcal{O}(1)} \tag{6}$$

A constant complexity.

- *void* **printInfix();** - Prints the Infix from inFixQueue. This function is

$$\boxed{\mathcal{O}(1)} \tag{7}$$

A constant complexity.

- *void* **printPostfix();** - Prints the Postfix from postFixQueue. This function is

$$\boxed{\mathcal{O}(1)} \tag{8}$$

A constant complexity.

Evaluator The Evaluator class contains the following **Private** members:

- $LinkedQueue < std :: string >$ **postFixQueue;** - A queue that holds the postFix notation
- $LinkedQueue < std :: string >$ **eval;** - A queue that holds the inFix notation
- *double* **value;** - This returns the final answer as a double.
- *string* **tres;** - Global string for the answer that returns from the double evaluation.
- *string* **before_double;** - String that later gets converted to a double.

And the following **Public** members.

- $Evaluator(LinkedQueue < std :: string > par)$ - This is the default constructor for the Evaluator class. The input is a LinkedQueue object of strings which contains the postFix expression that will be evaluated.

- *double* **getValue();** - This function returns the value of the final answer as a double. This function is

  $$\boxed{\mathcal{O}(1)} \tag{9}$$

  A constant complexity.

- *string* **switchCase(double a, double b, string cur_num);** - This function will evaluate the specific operations that is requested from the postFix notation. The arguments are two strings that will be operated on. This function returns a string that will be pushed onto the stack for later operations. This function is

  $$\boxed{\mathcal{O}(1)} \tag{10}$$

  A constant complexity.

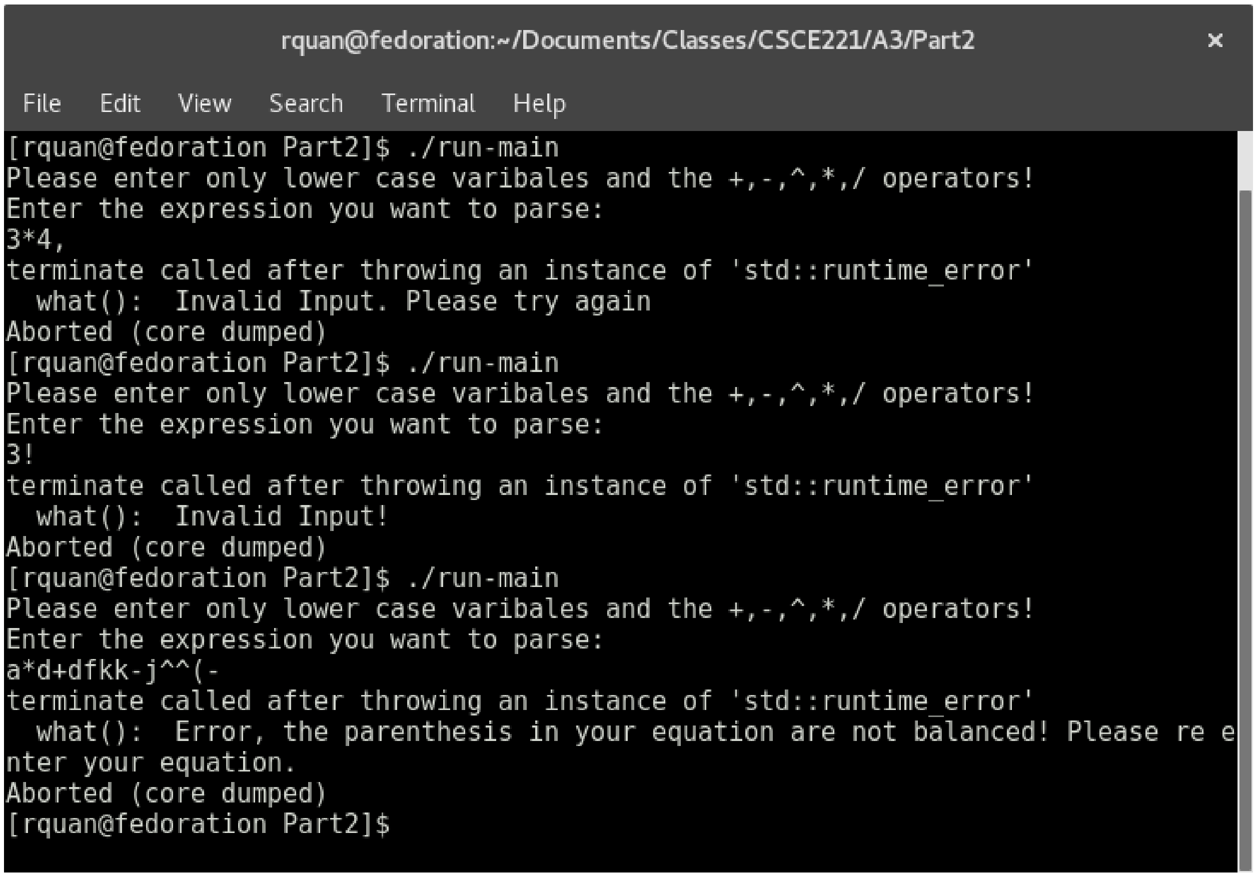
- $LinkedQueue < std :: string >$ **get_digits(**$LinkedQueue < std :: string > in$**);** - If the user submitted the expression using variables, this function will request the user to submit numerical entries for the said variables. This function returns a postFix queue that contains numbers and operators. This function is

  $$\boxed{\mathcal{O}(n)} \tag{11}$$

  A linear complexity.

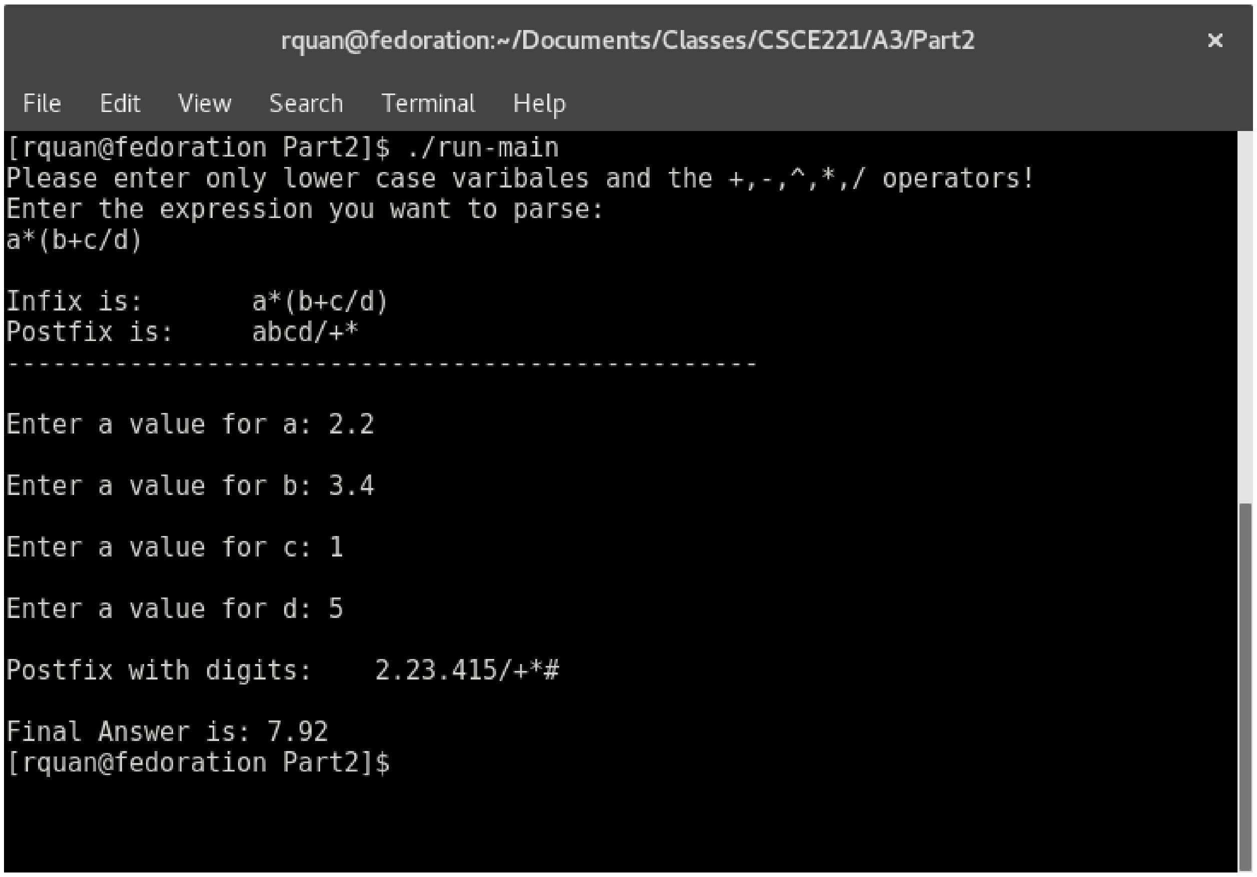
3) For correctness, I inputted various expression that contained illegal characters. The program caught these characters correctly and threw an error message which then ended the program. These characters were caught because I created a test with the use of ASCII characters and their acceptable ranges.

4) Below I have demonstrated a few test cases that use illegal characters and are caught by the program. As shown, two instances of unbalanced parenthesis and illegal expressions are caught from the input, thus proving the correctness of the program!


rquan@fedoration:~/Documents/Classes/CSCE221/A3/Part2
File Edit View Search Terminal Help

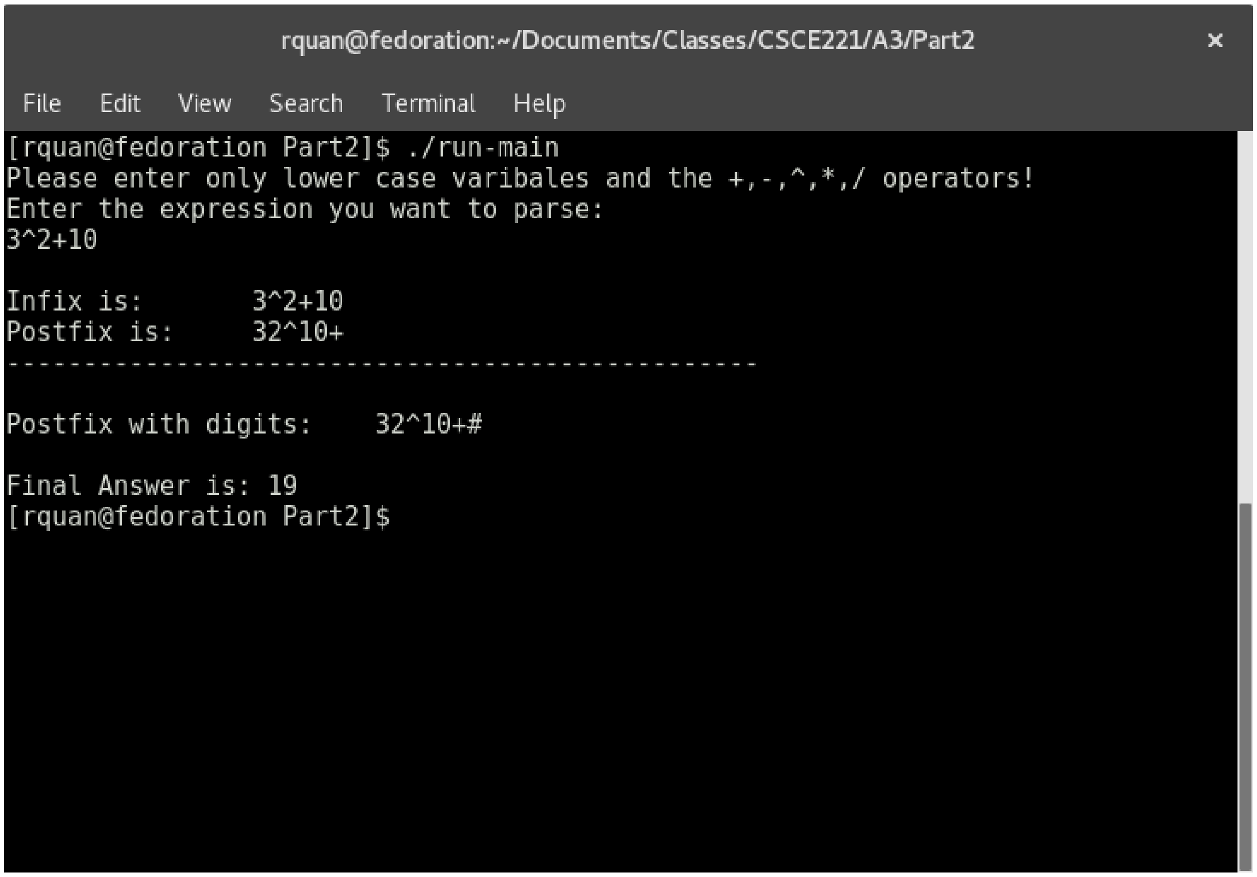
```
[rquan@fedoration Part2]$ ./run-main
Please enter only lower case varibales and the +,-,^,*,/ operators!
Enter the expression you want to parse:
3*4,
terminate called after throwing an instance of 'std::runtime_error'
  what():  Invalid Input. Please try again
Aborted (core dumped)
[rquan@fedoration Part2]$ ./run-main
Please enter only lower case varibales and the +,-,^,*,/ operators!
Enter the expression you want to parse:
3!
terminate called after throwing an instance of 'std::runtime_error'
  what():  Invalid Input!
Aborted (core dumped)
[rquan@fedoration Part2]$ ./run-main
Please enter only lower case varibales and the +,-,^,*,/ operators!
Enter the expression you want to parse:
a*d+dfkk-j^^(-
terminate called after throwing an instance of 'std::runtime_error'
  what():  Error, the parenthesis in your equation are not balanced! Please re e
nter your equation.
Aborted (core dumped)
[rquan@fedoration Part2]$
```

Figure 1: Illegal characters test.


rquan@fedoration:~/Documents/Classes/CSCE221/A3/Part2
File Edit View Search Terminal Help

```
[rquan@fedoration Part2]$ ./run-main
Please enter only lower case varibales and the +,-,^,*,/ operators!
Enter the expression you want to parse:
a*(b+c/d)

Infix is:       a*(b+c/d)
Postfix is:     abcd/+*
-------------------------------------------------

Enter a value for a: 2.2

Enter a value for b: 3.4

Enter a value for c: 1

Enter a value for d: 5

Postfix with digits:    2.23.415/+*#

Final Answer is: 7.92
[rquan@fedoration Part2]$
```

Figure 2: A run of the postFix Evaluator program.


rquan@fedoration:~/Documents/Classes/CSCE221/A3/Part2
File Edit View Search Terminal Help

```
[rquan@fedoration Part2]$ ./run-main
Please enter only lower case varibales and the +,-,^,*,/ operators!
Enter the expression you want to parse:
3^2+10

Infix is:       3^2+10
Postfix is:     32^10+
-------------------------------------------------

Postfix with digits:    32^10+#

Final Answer is: 19
[rquan@fedoration Part2]$
```

Figure 3: Another screenshot of the program