Lab 1: Postfix Conversion

Yichen Dong

To implement the stack in my project, I chose to use an array. I believed that an array would be better in this instance, since each line of expressions was not changing dynamically. That is, we could find the length of the line, and create a stack that was stored in an array of that length. This will always be enough for the expression, since operators are not stored in the stack, and since the size of the stack will only consolidate down to 1 as the program progresses. In addition, since this is a postfix expression, we could have taken advantage of the random access part of an array implementation to get the operand B before A. In the exercise though, I decided to just go with the standard stack pop, storing the results in temp variables until they could be used in the operations.

A stack makes sense in this situation because the operation occurs sequentially. The data is read one at a time and stored. When an operator is encountered, the data is then popped out of the stack for us to use. We never need the top two values at any given time. A stack is also useful because we do not need the popped data once we have used it, since we have stored the values in a TEMPn variable. A stack implementation allows us to ensure that we do not store extraneous data that might confuse us.

One way that this could be implemented recursively is that the function would accept a string/character vector and an N that would be initialized to 0. The function would take string[n] and perform the same operations it would have in the loop format. When it is done, it would call the function again, passing the same string/character vector, but passing n+1. The stopping point for this recursive function would be when n == stringVector.length.

I do not know if this is the best way to implement the recursion. However, I think that this is worse than the iterative function. There would need to be a recursion within the recursion. We would need to recurse for each of the lines until we hit a line that was null. For each of those recursions, we would need to call a recursion for each of the elements in that line. This would be more complicated than just the iterative function. However, the “error checks” could occur easier. Each of the error checks would be programmed in as a stopping point, at which point the recursion would end and the end would be returned.

I learned that stacks are easy to work with, and that it is easy to extend the functionality of stacks with simple solutions. One example was my solution to delete everything in the stack by popping one at a time until the isEmpty() function returned true. I also learned that stacks are very useful in working with data where you would only ever need the most recent one or two data points. Past that number, it becomes a bit unwieldy dealing with the number of temp variables needed. I also realized that half my code was checking for errors.

Next time I would try to separate out the printing functions for each operator. Many of the functions were the same, and could have been modularized. However, I was not sure how to handle errors and break the loop if I was to contain it within its own class. Therefore, the program is a bit more redundant than I would have liked. I think I would also try out linked lists next time, if only for the learning experience. I was also toying around with the idea of using a StringBuilder for the output, but decided against it as it was not necessary. Another enhancement would be to return the line and location of the error.

I decided to use a character array because it was simpler to iterate through the array than through a string. This caused me to have to convert it into a string for the stack, because I had to push the TEMPn variables in, which were not characters. In addition, I decided to show the output for the computer up until the error occurred. This would hopefully make it easier for the user to track down the error and correct it. I chose to not assume anything, even trailing tabs and white spaces, and returned it to the user to check. This why, it is up to the user to decide whether they made a typo or not, not the programmer.

Efficiency wise, I think that this is as good as it will get. The program has to loop through each line, and then through each character, in order to create the instructions for the computer. The only thing I would change is to condense the code, as I mentioned earlier.

Appendix

Output for Provided Input

LD A

AD B

ST TEMP1

LD TEMP1

SB C

ST TEMP2

ERROR: White space or tabs are not accepted.Please check your expression and try again.

LD B

AD C

ST TEMP1

LD A

SB TEMP1

ST TEMP2

LD A

SB B

ST TEMP1

LD TEMP1

AD C

ST TEMP2

LD E

SB F

ST TEMP3

LD D

AD TEMP3

ST TEMP4

ERROR: $ is not accepted. Please check your expression and try again.

LD D

SB E

ST TEMP1

LD C

AD TEMP1

ST TEMP2

ERROR: $ is not accepted. Please check your expression and try again.

LD B

AD C

ST TEMP1

LD A

ML TEMP1

ST TEMP2

LD B

SB A

ST TEMP3

LD C

AD TEMP3

ST TEMP4

LD TEMP2

ML TEMP4

ST TEMP5

LD B

AD C

ST TEMP1

LD A

DV TEMP1

ST TEMP2

LD B

ML A

ST TEMP3

LD C

AD TEMP3

ST TEMP4

ERROR: White space or tabs are not accepted.Please check your expression and try again.

LD A

SB B

ST TEMP1

ERROR: Too many operators. Please check your expression and try again.

LD B

SB C

ST TEMP1

LD A

DV TEMP1

ST TEMP2

LD B

SB A

ST TEMP3

LD TEMP2

AD TEMP3

ST TEMP4

ERROR: Too many operators. Please check your expression and try again.

LD B

AD C

ST TEMP1

ERROR: $ is not accepted. Please check your expression and try again.

ERROR: 0 is not accepted. Please check your expression and try again.

Output: User created input

LD A

AD B

ST TEMP1

ERROR: Too many operators. Please check your expression and try again.

ERROR: 9 is not accepted. Please check your expression and try again.

LD C

DV D

ST TEMP1

ERROR: Items still remaining in stack. Check number of operands and try again.

ERROR: White space or tabs are not accepted.Please check your expression and try again.

LD B

ML A

ST TEMP1

LD A

DV TEMP1

ST TEMP2

LD B

ML TEMP2

ST TEMP3

LD A

DV TEMP3

ST TEMP4

LD C

ML D

ST TEMP1

LD B

DV TEMP1

ST TEMP2

LD F

AD G

ST TEMP3

LD E

SB TEMP3

ST TEMP4

LD TEMP2

SB TEMP4

ST TEMP5

LD A

SB TEMP5

ST TEMP6

ERROR: is not accepted. Please check your expression and try again.

ERROR: 9 is not accepted. Please check your expression and try again.

ERROR: Too many operators. Please check your expression and try again.

LD a

AD B

ST TEMP1

LD e

AD F

ST TEMP2

LD g

SB G

ST TEMP3

LD TEMP2

SB TEMP3

ST TEMP4

LD c

SB TEMP4

ST TEMP5

LD d

SB TEMP5

ST TEMP6

ERROR: Items still remaining in stack. Check number of operands and try again.

LD A

SB B

ST TEMP1

LD TEMP1

SB C

ST TEMP2

LD TEMP2

SB D

ST TEMP3

LD TEMP3

SB E

ST TEMP4

LD TEMP4

SB F

ST TEMP5

LD TEMP5

SB G

ST TEMP6

ERROR: Blank line detected. Please check your expression and try again.

ERROR: @ is not accepted. Please check your expression and try again.

ERROR: (is not accepted. Please check your expression and try again.

ERROR: Prefix operation detected. Please check your expression and try again.