CCDSALG Term 3, AY 2019 – 2020

Project 2 Documentation – Calculator (Stack and Queue Application)

Section	Names	Task 1	Task 2	Task 3	Task 4	Task 5
<S15>	Lim, Christopher	X	X	X	X	X
<S11>	Lim, Ivan	X	X	X	X	X
<S12>	Sy, James	X	X	X	X	X

Fill this part with your section and names. For the tasks, put an X mark if you have performed the specified task. Please refer to the project specifications for the tasks.

1. Programming Language Used: C

2. Why did you choose the programming language above for Project 2? Explain briefly (1 to 2 sentences).

We chose C since it was the language that we thought we could learn most about dynamic memory allocation and pointers. This was also a way for us to learn more about the implementations rather than using pre-made methods.

3. Depending on the programming language used:

a. List the libraries or APIs that you used in your implementation

- stdio.h
- stdlib.h
- string.h
- limits.h
- math.h

b. Indicate how to compile (if it is a compiled language) your codes, and how to RUN (execute) your program from the COMMAND LINE. Examples are shown below highlighted in yellow. Replace them accordingly. Make sure that all your group members test what you typed below because I will follow them verbatim. I will initially test your solution using the sample input text file that you submitted. Thereafter, I will run it again using my own test data:

- How to compile from the command line (for compiled language only):

```
C:\CCDSALG> gcc -o main main.c -Wall  -lm
```

- How to run from the command line

```
C:\CCDSALG>main
```

4. How did you implement your data structures (did you use arrays or linked list)? Why? Explain briefly (1 to 2 sentences).

a. How did you parse the input?

1. While the character is an operand, store/accumulate/concatenate it.
2. Take the next 2 index after the operand (for single[-, +, ...] and double[=>, ==, ...] character operators)
3. Now that we have both the operand and operator, we apply the Infix to Postfix Expression algorithm.

4. Repeat Step 1 until we finish to loop the Infix Expression from left to right

b. How did you specify the priority and precedence of the operators?

- Detect what operator the character(s) was using a function, then give/return an integer weight.

c. How did you implement the conversion and evaluation of the logical not unary operator?

- The not unary operator (!) is considered to have the highest priority among the operator precedence. Thus, when implemented in the conversion from infix to postfix, we made sure that when there is a ! operator, it gets the highest value so that it is always the first to be popped from the stack. On the other hand, dealing with the ! operator on the evaluation needed an if statement to check if the not operation was going to be used so that only one number from the queue will be used and not two (as opposed to usual operators that utilize two operands).

5. Disclose what is NOT working correctly in your solution. Be honest about this. Explain briefly the reason why your group was not able to make it work.

a. unary operator "-" doesn't work in our solution.
   - Although it doesn't exist in the project specification, we saw it in the discussion and decided to put it here since it doesn't work for our solution because there are 2 "-", 1 as a binary operator and the other as a unary operator.

b. sqrt or nˆ(½) doesn't work since ½ = 0

c. multiple "!" doesn't work properly if it's not done with a parenthesis. For example: "!!!0" doesn't work but "!(!(!(0)))" works

6. What do you think is the level of difficulty of the project (was it easy, medium or hard)? Which part is hard (if you answered hard)? Type your answer individually for this question.

Lim, Chritopher:

At first, I thought the difficulty of the project was easy, since we were simply implementing an existing algorithm. But I then realized that the operands may have multiple digits(numbers from 10 and beyond) and operators may have more than 1 character(=>, ||, ==, etc.). This made it a bit more challenging since you had to detect which were operators and which were operands in an infix expression which has no space available. Because of that, the difficulty of the project became medium for me.

Lim, Ivan Jerwin:

For me, the level of the difficulty of the project is hard because it is very heavily dependent on logic in implementing the solutions. I would say that implementing the postfix evaluation was the hardest due to the amount of thought it needed to be implemented.

Sy, James Matthew: From when we started the project with the implementations of the stack and queue, I thought it was going to be easy because I thought we only needed to do little work to finish the project. However, problems came in as soon as we started on the infix to postfix conversion. We realized that various concepts such as the pointer reference and dynamic memory allocation were needed. Moreover, the use of a queue was required for the evaluation. We had to devise an algorithm that would evaluate any expression that is entered. I would say this project was hard and challenging especially on the formulation of the logic in generating the right algorithm.

7. Fill-up the table below. Refer to the rubric in the project specs. It is suggested that you do first an individual self-assessment. Thereafter, compute the average evaluation for your group, and encode it below.

REQUIREMENT	AVE. OF SELF-ASSESSMENT
1. Stack	19.5 (max. 20 points)
2. Queue	19.5 (max. 20 points)
3. Infix-to-Postfix	25 (max. 25 points)
4. Postfix Evaluation	20 (max. 20 points)
5. Documentation	10 (max. 10 points)
6. Compliance with Instructions	5 (max. 5 points)

TOTAL SCORE 100 over 100.

NOTE: The evaluation that the instructor will give is not necessarily going to be the same as what you indicated above. The self-assessment serves primarily as a guide.