

# **EGRE 531 Multicore and Multithread Programming**

## **Laboratory Number 1**

**Date: 02/02/18**

**Luis Barquero**

**PLEDGE:** \_\_\_\_\_ **Luis Barquero** \_\_\_\_\_

**“On my honor, I have neither given nor received  
unauthorized aid on this assignment”**

**Introduction:**

The lab was split into two parts: a C/C++ review program regarding Reverse Polish Notation(RPN), and a benchmark multithreading program where the number of threads were varied and the number of seconds to complete the program were recorded.

**Lab Content:**

**Part 1:** A C/C++ program implementing Reverse Polish Notation(RPN) was created, where the user would enter the operator symbol is placed after the arguments. For example, if the user wanted to have “5 + 8”, the RPN expression would be “5 8 +.” Through RPN, the expression is evaluated left to right, greatly simplifying the computation of the expression within computer programs.

For this program, a stack was implemented that would take the RPN as a string, but would later convert the string into a double to extract the numbers. From there, if statements were implemented that would check which operator was used, and if the condition was met, it would pass the top numbers of the stack to the corresponding function according to their operators. Next, the two top elements in the stack are popped and the corresponding operation is performed, and the answer is pushed to the top of the stack to be displayed to the terminal. Finally, the user can decide to enter more numbers and operators to the stack to be performed, or the user can also clear the stack to enter a new expression. The program also provides to show to enter the expression in multiple lines, as in, the user can enter a number, hit enter, enter another number, and finish by providing the operator. Finally, since the program will constantly ask the user to enter an expression, the exit function is implemented where the user can enter either a lowercase or uppercase q to exit the program.

**Part 2:** A multithreading program was provided that would take the amount of threads provided and would calculate the execution time using the OpenMP application programming interface. From Figure 1, it is evident that as the number of cores increases, the execution times decreases. However, after using two cores, the average execution time for each thread remained within 10 to 30 seconds from the others due to overhead caused by the excess of cores.

In the program, the number of threads was varied from 1 to 8, and each thread received 15 trials, from which the execution time was recorded for each. From there, the average execution time was calculated for each thread and their 15 trial results, and a graph showing the results was implemented.

### **Test Results:**

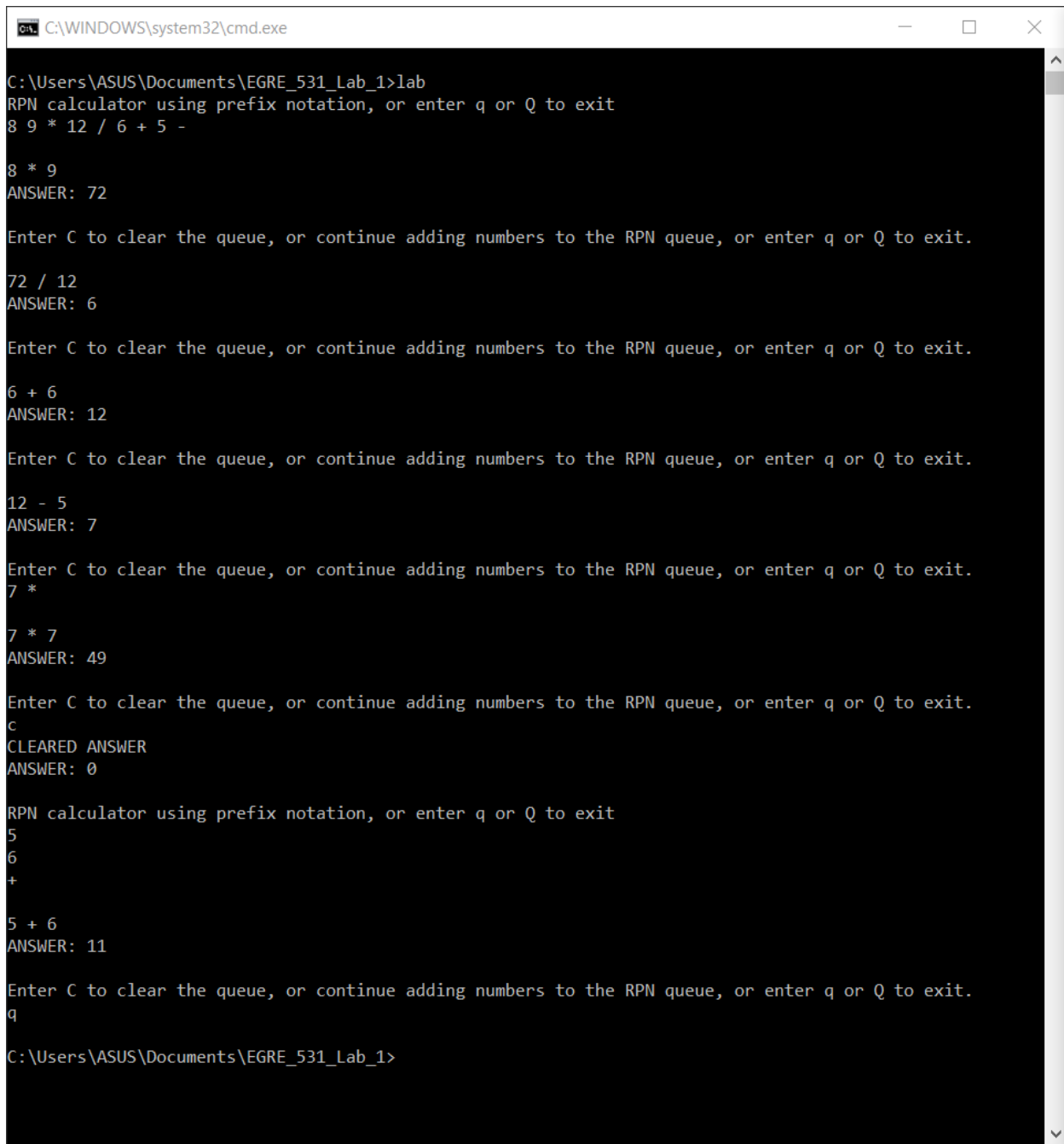
**Part 1:** To fully test the RPN calculator, the following expression was entered:

$$8\ 9 * 12 / 6 + 5 -$$

This expression is first multiplying 8 and 9, giving 72, and then dividing by 12, which will yield 6. Next, the expression will add 6 to the previous result, giving 12, and then finally subtracting 5, providing the final answer of 7. Next, to show that more numbers and operators can be added to the stack, the expression 7 \* is entered, which multiplies the previous answer with 7, providing 49 as the final answer.

Next, the stack is cleared and the expression 5 6 + is entered, with a newline between the numbers and operators, to show that the RPN expression does not have to be entered in one whole line.

Figure 1 shows the terminal expression entered, which each answer in between each step.



```
C:\WINDOWS\system32\cmd.exe

C:\Users\ASUS\Documents\EGRE_531_Lab_1>lab
RPN calculator using prefix notation, or enter q or Q to exit
8 9 * 12 / 6 + 5 -

8 * 9
ANSWER: 72

Enter C to clear the queue, or continue adding numbers to the RPN queue, or enter q or Q to exit.

72 / 12
ANSWER: 6

Enter C to clear the queue, or continue adding numbers to the RPN queue, or enter q or Q to exit.

6 + 6
ANSWER: 12

Enter C to clear the queue, or continue adding numbers to the RPN queue, or enter q or Q to exit.

12 - 5
ANSWER: 7

Enter C to clear the queue, or continue adding numbers to the RPN queue, or enter q or Q to exit.

7 *
7 * 7
ANSWER: 49

Enter C to clear the queue, or continue adding numbers to the RPN queue, or enter q or Q to exit.
C
CLEARED ANSWER
ANSWER: 0

RPN calculator using prefix notation, or enter q or Q to exit
5
6
+
5 + 6
ANSWER: 11

Enter C to clear the queue, or continue adding numbers to the RPN queue, or enter q or Q to exit.
q

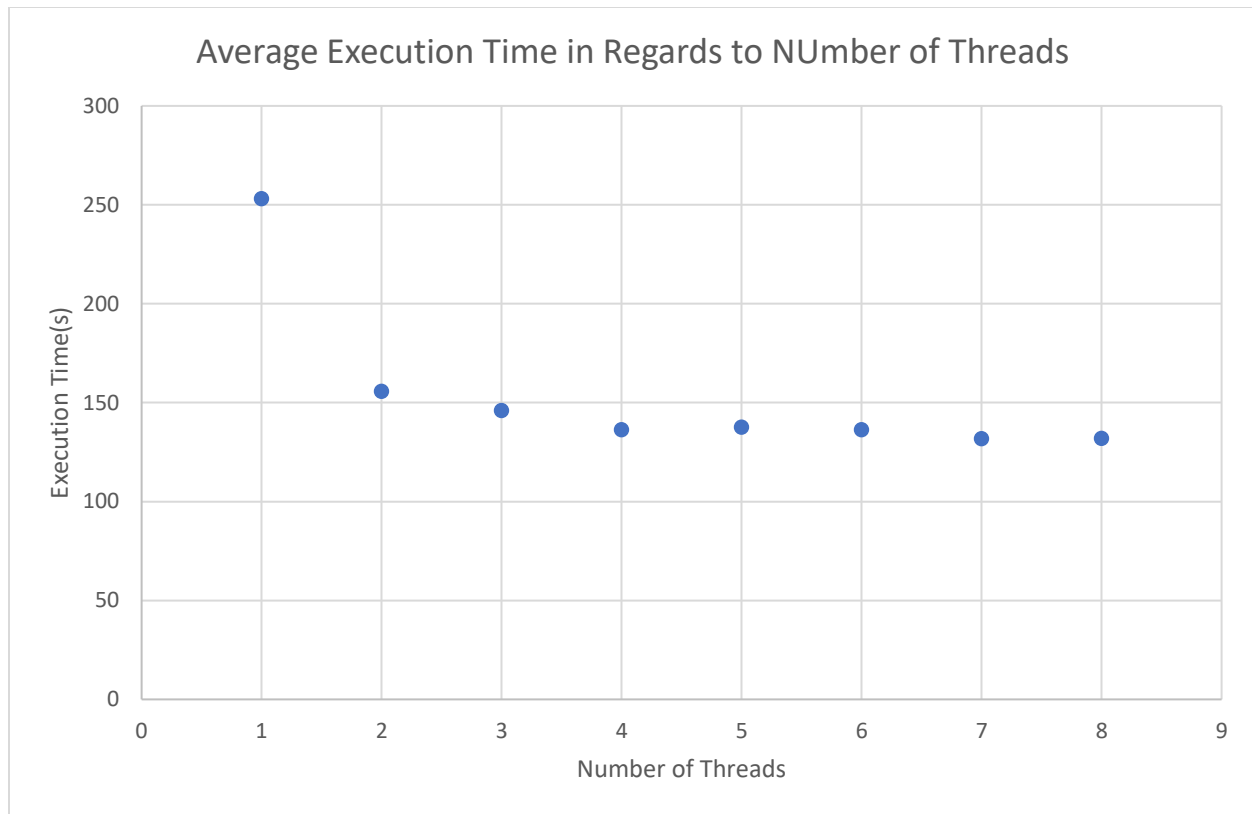
C:\Users\ASUS\Documents\EGRE_531_Lab_1>
```

*Figure 1 – Figure 1 shows the RPN calculator from Part 1 with two expressions and their corresponding answers.*

**Part 2:** Given the benchmark code, the following table and graph show the 15 trials for the eight threads and their average execution time.

Trials	No. of Threads							
	1	2	3	4	5	6	7	8
1	277	139	142	137	143	160	131	130
2	274	178	143	129	134	138	131	145
3	266	168	145	136	136	139	136	134
4	276	159	142	128	136	136	140	135
5	265	144	218	131	140	141	130	152
6	255	167	144	136	136	140	123	133
7	241	158	143	132	134	136	138	139
8	253	157	142	130	140	126	134	137
9	239	154	148	138	138	139	131	131
10	238	144	140	138	138	132	132	138
11	244	157	133	133	136	135	138	119
12	245	153	144	130	143	135	136	122
13	244	153	135	132	137	134	126	121
14	238	151	134	137	135	126	124	118
15	241	154	138	133	139	128	126	126
Avg Execution Time	253.0666667	155.7333	146.0667	136.333	137.667	136.3333	131.7333	132

*Table 1 – Table 1 shows the 15 trials for the 8 threads and their average execution time.*



**Figure 2 – Figure 2 shows how the execution time behaves as the number of threads change.**

From the table and graph, it is evident that the average execution time decreases rapidly from using one core to two, but as the number of threads increase after that, the average execution time remains relative close, within 10 to 30 seconds from each other as opposed to the initial drop. This is due to overhead caused by the excess number of threads.

### **Problems Encountered:**

The main problem encountered was converting the RPN expression string to a double to extract the numbers, push the numbers to the top of the stack, determine the operator, pop the numbers from the stack and perform the corresponding operations(s), and then finally push the answer back to the top so it can be displayed.