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 pram 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.

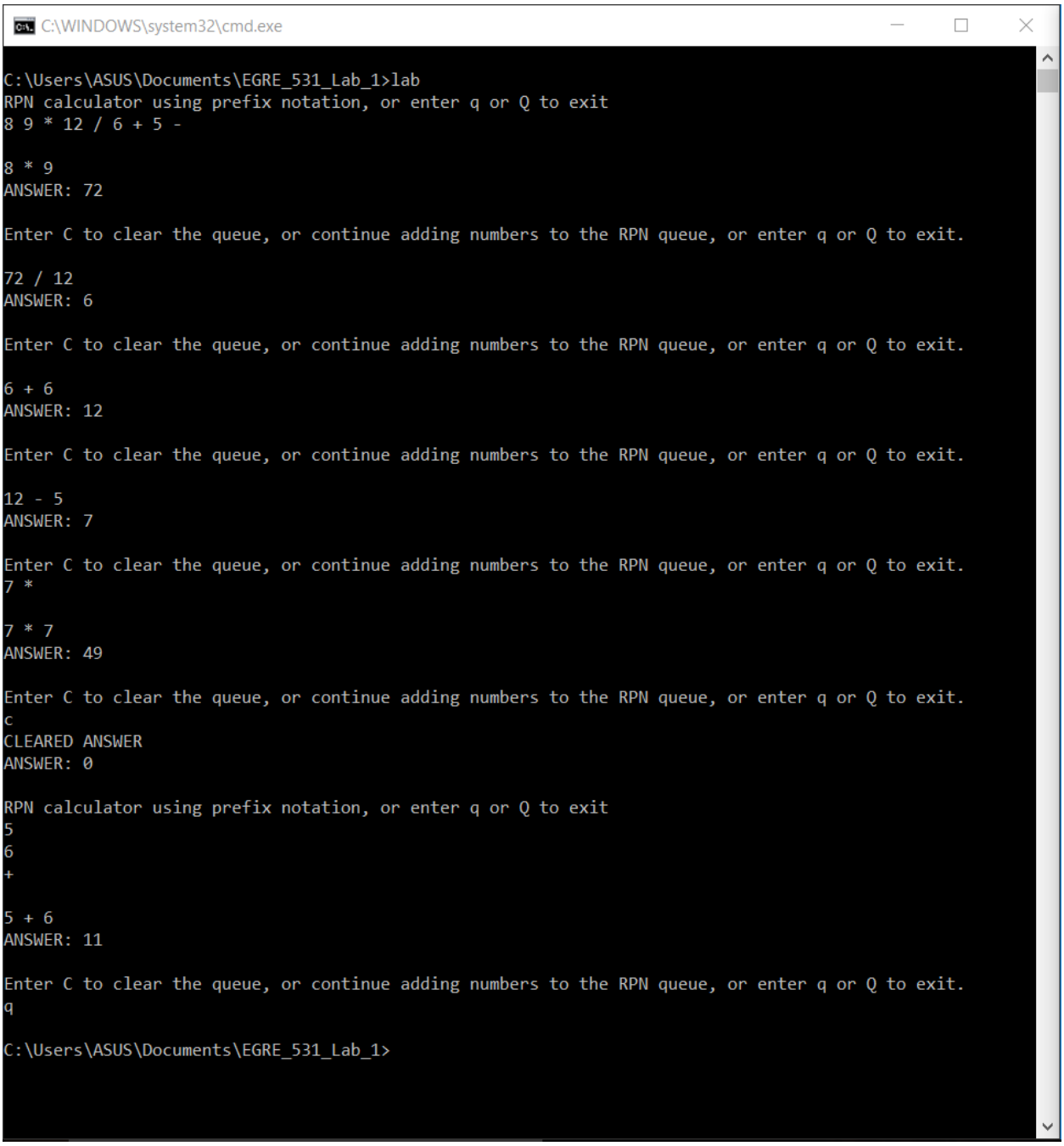

Figure 1 – Figure 1 shows the RPN calculator from Part 1 with two expressions and their corresponding answers.

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

	No. of Threads							
Trials	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.

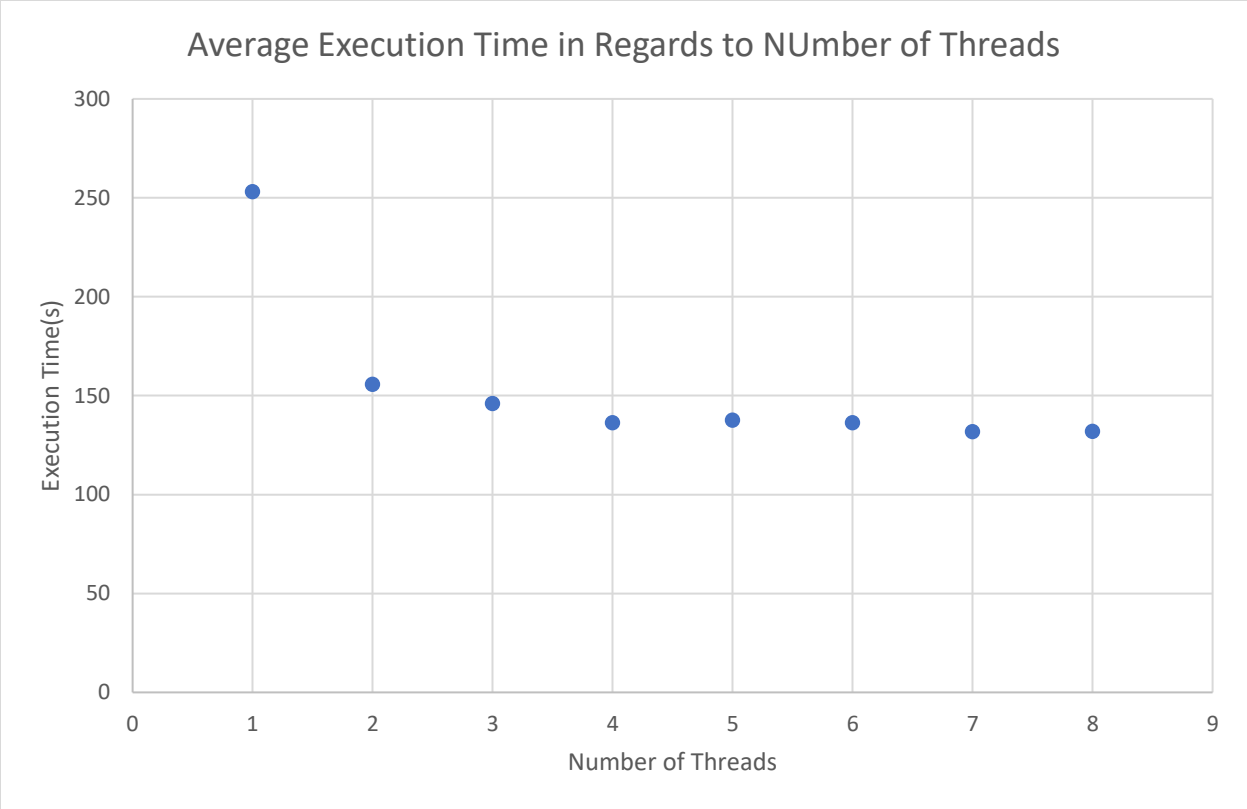

Figure 2 – Figure 2 shows how the execution time behaves as the number of threads change.

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.

Appendix A

RPN Source Code: Lab1.h

```
/*****************************************************************************
EGRE 531: Multithreaded Programming
Lab 1
Programmed by: Luis Barquero
Purpose: Program will calculate the value of an RPN expression.
*****************************************************************************/
#include <iostream>
#include <string>
#include <cmath>
#include <stack>

#ifndef Lab1_H
#define Lab1_H

using namespace std;
class Calculator
{
    private: //Private members.
    double answer;
    stack<double> answer_stack; // Stack the will be used to compute the calculator

    public: //Public members
    Calculator(); //Default constructor.
    void add(double, double); //function that will add two numbers.
    void sub(double, double); //function that subtracts two number.
    void mult(double, double); //function that will add two numbers.
    void div(double, double); //function that divides two numbers.
    void add(double); //function that will add a number to a previous number.
    void sub(double); //function that will subtract a number from a previous answer.
    void mult(double); //function that will multiply a number from a previous answer.
    void div(double); //function that will divide a number froma previous entry.
    void enter(double); //function that will enter a new number into the registry.
    void add(); //Function that will add the two top members in the stack.
    void sub(); //Function that will subtract the two top members in the stack.
    void mult(); //Function that will multiply the two top members in the stack.
    void div(); // Function that will divide the two top members in the stack.
    void prt(); //print function.
    void clear(); //clear function.
};
#endif
```

Appendix B

RPN Source Code: Main.cpp

```
/*****************************************************************************
EGRE 531: Multithreaded Programming
Lab 1
Programmed by: Luis Barquero
Purpose: Program will calculate the value of an RPN expression.
*****************************************************************************/

#include <iostream>
#include <stack>
#include "Lab1.h"
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <algorithm>
using namespace std;
bool str2double(string&, double&);

int main()
{
    Calculator rpn;
    string ptr; //variable that represents the user's input.
    double convert; //variable that will be used in the conversion of the string to the
    double.
    char c; // variable used to set the while loop.
    cout << "RPN calculator using prefix notation, or enter q or Q to exit" << endl;
    while(c == 0) //while loop condition.
    {
        cin >> ptr; // extracts the user's input.
        if(str2double(ptr,convert)) //string to double function.
        {
            rpn.enter(convert); //if the input is a double, it enters the double into
            the stack and prints them.
        }

        if(ptr == "+") //if the user's string involves a + operator
        {
            rpn.add(); //adds the two numbers and places them onto the stack.
            rpn.prt(); // calls the print function.
            cout << "Enter C to clear the queue, or continue adding numbers to the RPN
            queue, or enter q or Q to exit." << endl;
        }
        else if (ptr == "-")//if the user's string involves a - operator
        {
            rpn.sub();//subtracts the two numbers and places them onto the stack.
            rpn.prt();//calls the print function.
            cout << "Enter C to clear the queue, or continue adding numbers to the RPN
            queue, or enter q or Q to exit." << endl;
        }
        else if(ptr == "*") //if the user's string involves a * operator
        {
            rpn.mult();//multiplies the two numbers and places them onto the stack.
            rpn.prt();//calls the print function.
            cout << "Enter C to clear the queue, or continue adding numbers to the RPN
            queue, or enter q or Q to exit." << endl;
        }
        else if(ptr == "/")//if the user's string involves a +/ operator
        {
            rpn.div();//divides the two numbers and places them onto the stack.
            rpn.prt();//calls the print function.
            cout << "Enter C to clear the queue, or continue adding numbers to the RPN
            queue, or enter q or Q to exit." << endl;
        }

        else if ((ptr == "c") || (ptr == "C")) //if the user's string involves an upper
        or lowercase c/
        {
            rpn.clear(); // calls the clear function (which clears the stack.)
            rpn.prt();//calls the print function.
            cout << "RPN calculator using prefix notation, or enter q or Q to exit" <<
```

```
            endl;
        }
        else if ((ptr == "q") || (ptr == "Q"))
        {
            exit(1);
        }
    }
}

bool str2double(string& term, double& operand)
{
  char* ptr;
  // conversion begins at term string address 0 and on success
  // pointer ptr is advanced to end of numeric portion of content
  operand = strtod(term.c_str(), &ptr);
  // return boolean value of comparisonn
  // addr stored in ptr to addr of term string
  return (ptr == (term.c_str()+term.length()));
}

```

Appendix C

RPN Source Code: Functions.cpp

```
/*****************************************************************************
EGRE 531: Multi-threaded Programming
Lab 1
Programmed by: Luis Barquero
Purpose: Program will calculate the value of an RPN expression.
*****************************************************************************/
#include <iostream>
#include <stack>
#include "Lab1.h"
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <algorithm>
using namespace std;

Calculator::Calculator() //Default constructor.
{

};

void Calculator::add(double x, double y) //function that will add 2 numbers.
{
    this -> answer = x + y; //this -> answer to access private member answer. Adds up
    two numbers.
};


void Calculator::add(double x) //function that will add a number to previously marked
answer.
{
    stack<double> answer_stack;
    answer_stack.push(this -> answer); //places the answer at the previous answer at
    the top of the stack.
    answer_stack.pop(); //pops out the answer at the top of the stack.
    this -> answer = this -> answer + x; //performs the + operation and stores the
    answer in this -> answer.
    answer_stack.push(this -> answer); //pushes this -> answer back into the stack.
};

void Calculator::sub(double x) // function that will subtract a number from a previous
answer.
{
    stack<double> answer_stack;
    answer_stack.push(this -> answer); //places the answer at the previous answer at
    the top of the stack.
    answer_stack.pop(); //pops out the answer at the top of the stack.
    this -> answer = this -> answer - x; //performs the - operation and stores the
    answer in this -> answer.
    answer_stack.push(this-> answer); //pushes this -> answer back into the stack.
};

void Calculator::sub(double x, double y) // function that will subtract a number from a
previous answer.
{
    stack<double> answer_stack;
    answer_stack.push(this -> answer); //places the answer at the previous answer at
    the top of the stack.
    answer_stack.pop(); //pops out the answer at the top of the stack.
    this -> answer = x - x; //performs the - operation and stores the answer in this ->
    answer.
    answer_stack.push(this-> answer); //pushes this -> answer back into the stack.
};

void Calculator::div(double x, double y) // function that will subtract a number from a
previous answer.
{
    stack<double> answer_stack;
    answer_stack.push(this -> answer); //places the answer at the previous answer at
    the top of the stack.
```

```
58          answer_stack.pop(); //pops out the answer at the top of the stack.
59          this -> answer = x / y; //performs the - operation and stores the answer in this ->
            answer.
60          answer_stack.push(this-> answer); //pushes this -> answer back into the stack.
61      };
62
63      void Calculator::mult(double x, double y) // function that will subtract a number from
        a previous answer.
64      {
65          stack<double> answer_stack;
66          answer_stack.push(this -> answer); //places the answer at the previous answer at
            the top of the stack.
67          answer_stack.pop(); //pops out the answer at the top of the stack.
68          this -> answer = x * y; //performs the - operation and stores the answer in this ->
            answer.
69          answer_stack.push(this-> answer); //pushes this -> answer back into the stack.
70      };
71
72      void Calculator::mult(double x) // function that will multiply a number to a previous
        answer.
73      {
74          stack<double> answer_stack;
75          answer_stack.push(this -> answer);  //places the answer at the previous answer at
            the top of the stack.
76          answer_stack.pop(); //pops out the answer at the top of the stack.
77          this -> answer = this -> answer * x; //performs the * operation and stores the
            answer in this -> answer.
78          answer_stack.push(this-> answer); //pushes this -> answer back into the stack.
79      };
80
81      void Calculator::clear() //function that clears the inputs.
82      {
83          answer_stack.push(0); //pushes this -> answer into the stack.
84          cout << "CLEARED ANSWER" << endl;
85      };
86
87      void Calculator::enter(double x) //function that enters a number to the registry.
88      {
89          answer_stack.push(x); //places this -> answer at the top of the stack.
90      };
91
92      void Calculator::div(double x) // function that divides the answer and the number x.
93      {
94          stack<double> answer_stack;
95          answer_stack.push(this -> answer); //pushes this -> answer into the stack.
96          answer_stack.pop(); //pops out the answer at the top of the stack.
97          this -> answer = (this -> answer) / (x); //performs the / operator and stores the
            answer in this -> answer.
98          answer_stack.push(this-> answer); // pushes this -> answer into the stack.
99      };
100
101     void Calculator:: prt() //print function.
102     {
103         cout << "ANSWER: " << answer_stack.top() << endl;
104         cout << "\n";
105
106     };
107
108     void Calculator:: mult()
109     {
110         double a = 0;
111         double b = 0;
112         a = answer_stack.top(); // sets a = to the top of the stack.
113         answer_stack.pop();//pops out a;
114         b = answer_stack.top();  // sets b = to the top of the stack.
115         answer_stack.pop(); //pops out the top of the stack.
116         answer_stack.push(a*b);// pushes the expression a*b.;
117         cout << "\n" << b << " * " << a << endl;
118     };
```

```
119
120 void Calculator:: add()
121 {
122     double a = 0;
123     double b = 0;
124     a = answer_stack.top(); // sets a = to the top of the stack.
125     answer_stack.pop();//pops out a;
126     b = answer_stack.top(); // sets b = to the top of the stack.
127     answer_stack.push(a+b); // pushes the expression a+b.
128     cout << "\n" << b << " + " << a << endl;
129 };
130
131 void Calculator:: sub()
132 {
133     double a = 0;
134     double b = 0;
135     a = answer_stack.top(); // sets a = to the top of the stack.
136     answer_stack.pop(); //pops out a;
137     b = answer_stack.top(); // sets b = to the top of the stack.
138     answer_stack.push(b-a); // pushes the expression b-a.
139     cout << "\n" << b << " - " << a << endl;
140 };
141
142 void Calculator:: div()
143 {
144     double a = 0;
145     double b = 0;
146     a = answer_stack.top(); // sets a = to the top of the stack.
147     answer_stack.pop(); //pops out a;
148     b = answer_stack.top(); // sets b = to the top of the stack.
149     answer_stack.pop(); //pops out b;
150     answer_stack.push(b/a);// pushes the expression b/a.
151     cout << "\n" << b << " / " << a << endl;
152     if (a == 0) // condition where if a == 0, it will output an error message, since
    dividing by zero is not allowed.
153     {
154         cout << "Error." << endl;
155         exit(1);
156     }
157 };
```