Samuel Larson

Project 7

10/30/18

The objective of this program is to implement stacks to solve infix expressions. The program should be able to read in 1 or more expressions one character at a time from a file and put operators into a stack and operands into another stack. The stacks will be accessed to solve the expression.

The program will need the stack class from program 6 with these functions push(), pop(), size(), empty(). I will make 2 new functions outside of the class one to display a message if the equation is invalid and another to solve expressions if the program passes an operator and 2 operands to it.

The program should read each line of the file as an equation and place its tokens into stacks one character at a time. When the program runs into a ) symbol it should solve the equation until it hits a ( symbol. Once it hits the end of the line the program should solve the remaining operators to get a result.

The program should make sure of a few things. that numbers have an operator, that operators aren’t next to each other, parentheses aren’t empty, each closing parenthesis has an opening parenthesis, and an operator isn’t at the edge of a parenthesis.

//Samuel Larson

//Project 6

//10/22/18

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int evaluate(int, int, char);//evaluates expression

void invalid(); //displays that equation is invalid

class stack

{

public:

typedef char item;

//member constant

static const int CAPACITY = 20;

//constructor

stack(){used = 0;}

//modification functions

void push(item entry);

item pop();

//constant member functions

int size(){return used;}

bool empty(){return used == 0;}

private:

//data members

item data[CAPACITY];

int used;

};

int main()

{

stack num;

stack op;

ifstream infile;

infile.open("equation.txt");

char ch, func, popped;

string line;

int result, oper1, oper2, pos;

while(getline(infile, line))//reads down file

{

cout << "expression: ";

pos = 0;//sets position in line to 0

while(line[pos] >= ' ')//reads down line

{

ch = line[pos];

cout << ch;

if(ch < '(' || ch > '9' || ch == ',' || ch == '.')//checks if characters are valid

{//displays that the equation is invalid

invalid();

return 0;

}

if(ch == '+' || ch == '-' || ch == '\*' || ch == '/')//checks if an operator

{

if(line[pos - 1] == '(' || line[pos - 1] == '+' || line[pos - 1] == '-'

|| line[pos - 1] == '\*' || line[pos - 1] == '/')

{//displays that the equation is invalid

invalid();

return 0;

}

op.push(ch);

}

if(ch == '(')//checks for closing brackets

op.push(ch);

if(ch >= '0' && ch <='9')//checks for numbers

num.push(ch);

if(ch == ')')

{

if(op.empty())//makes sure there is a closing bracket

{//displays that the equation is invalid

invalid();

return 0;

}

popped = op.pop();

if(popped == '(' || line[pos - 1] == '+' || line[pos - 1] == '-'

|| line[pos - 1] == '\*' || line[pos - 1] == '/')

{//displays that the equation is invalid

invalid();

return 0;

}

while(popped != '(')

{//solves equation until it gets to the opening (

oper2 = num.pop() - '0';

oper1 = num.pop() - '0';

func = popped;

num.push(evaluate(oper1,oper2,func));

popped = op.pop();

}

popped = op.pop();

if(popped != '(')

{//solves equation outside of parenthesis

while(op.size() > 0 && popped != '(')

{

oper2 = num.pop() - '0';

oper1 = num.pop() - '0';

func = op.pop();

num.push(evaluate(oper1,oper2,func));

}

}

op.push(popped);

}

pos++;

}

while(op.size() > 0)

{//solves the remaining expression

oper2 = num.pop() - '0';

oper1 = num.pop() - '0';

func = op.pop();

num.push(evaluate(oper1,oper2,func));

}

cout << endl << "Value = " << num.pop() << endl;

}

return 0;

}

void stack::push(item entry) //pre: data is entered post: data is put into a stack

{

data[used] = entry;

++used;

}

stack:: item stack::pop()//post: data is returned from the stack

{

--used;

return data[used];

}

int evaluate(int oper1, int oper2, char func) //pre: 2 operators and 1 operand are entered post: equation is solved

{

int result;

switch(func)

{

case'+':

result = oper1 + oper2;

return('0' + result);

case'-':

result = oper1 - oper2;

return('0' + result);

case'\*':

result = oper1 \* oper2;

return('0' + result);

case'/':

result = oper1 / oper2;

return('0' + result);

}

}

void invalid()//post: message is displayed

{

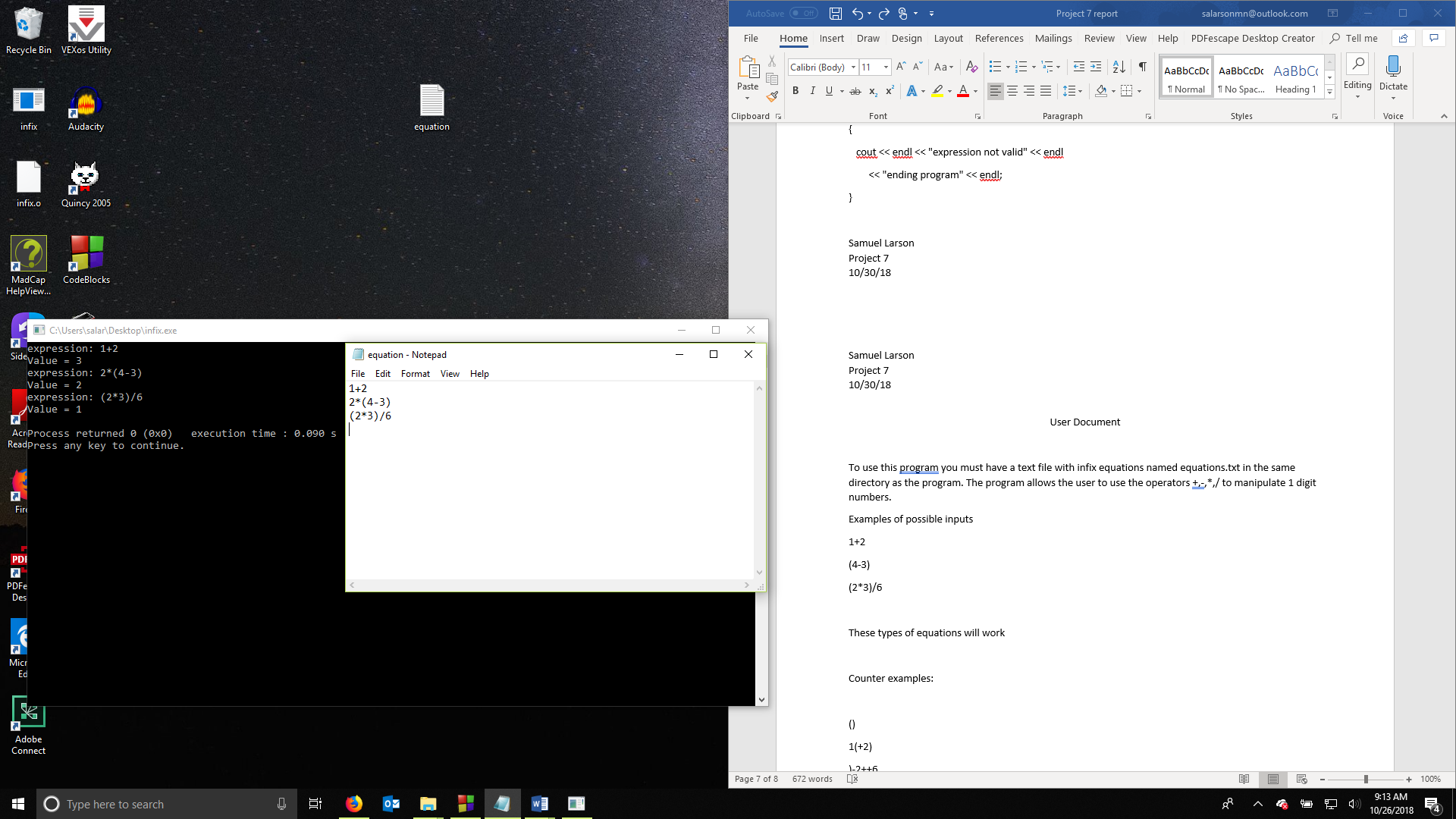
cout << endl << "expression not valid" << endl

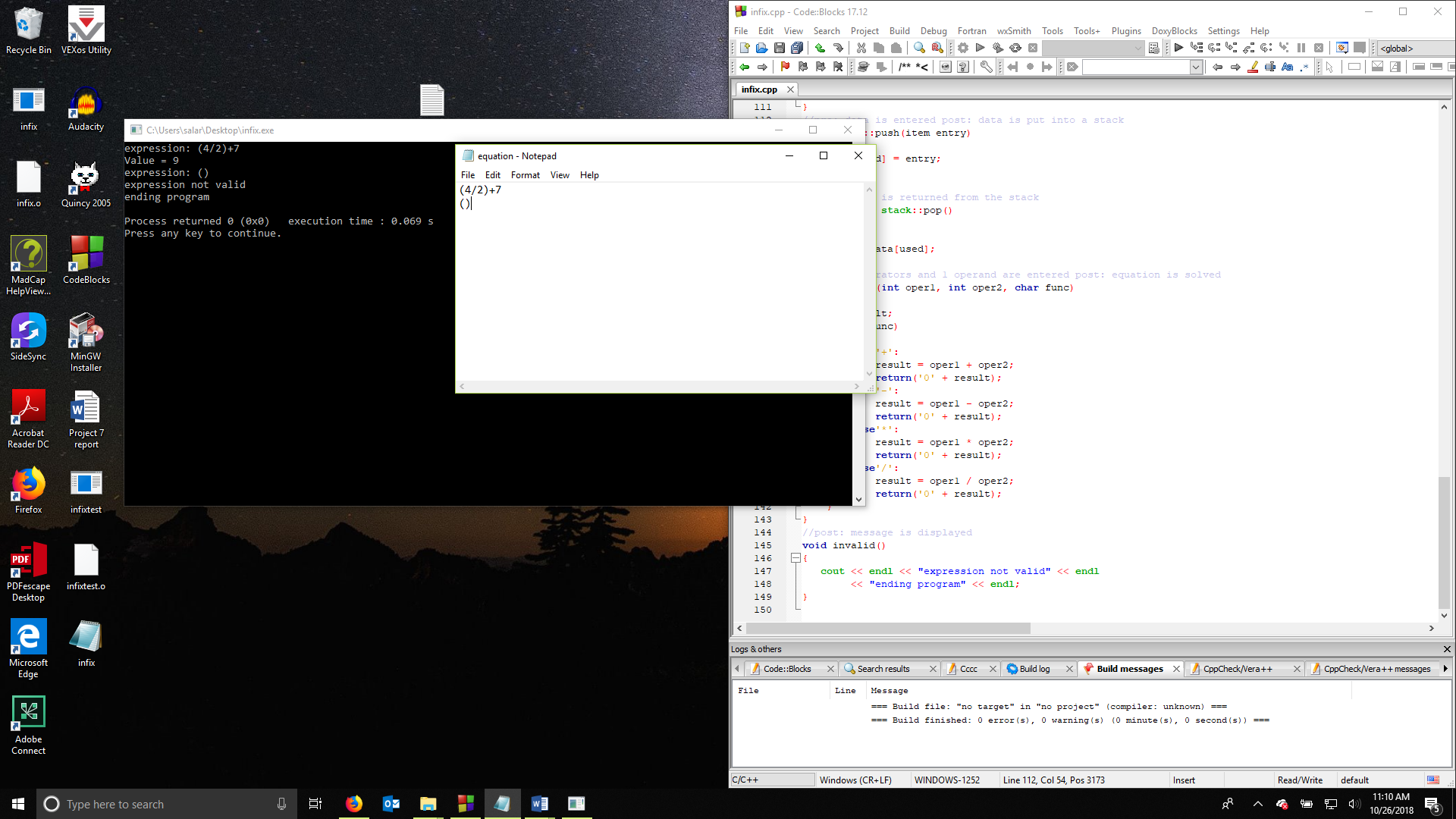
<< "ending program" << endl; }

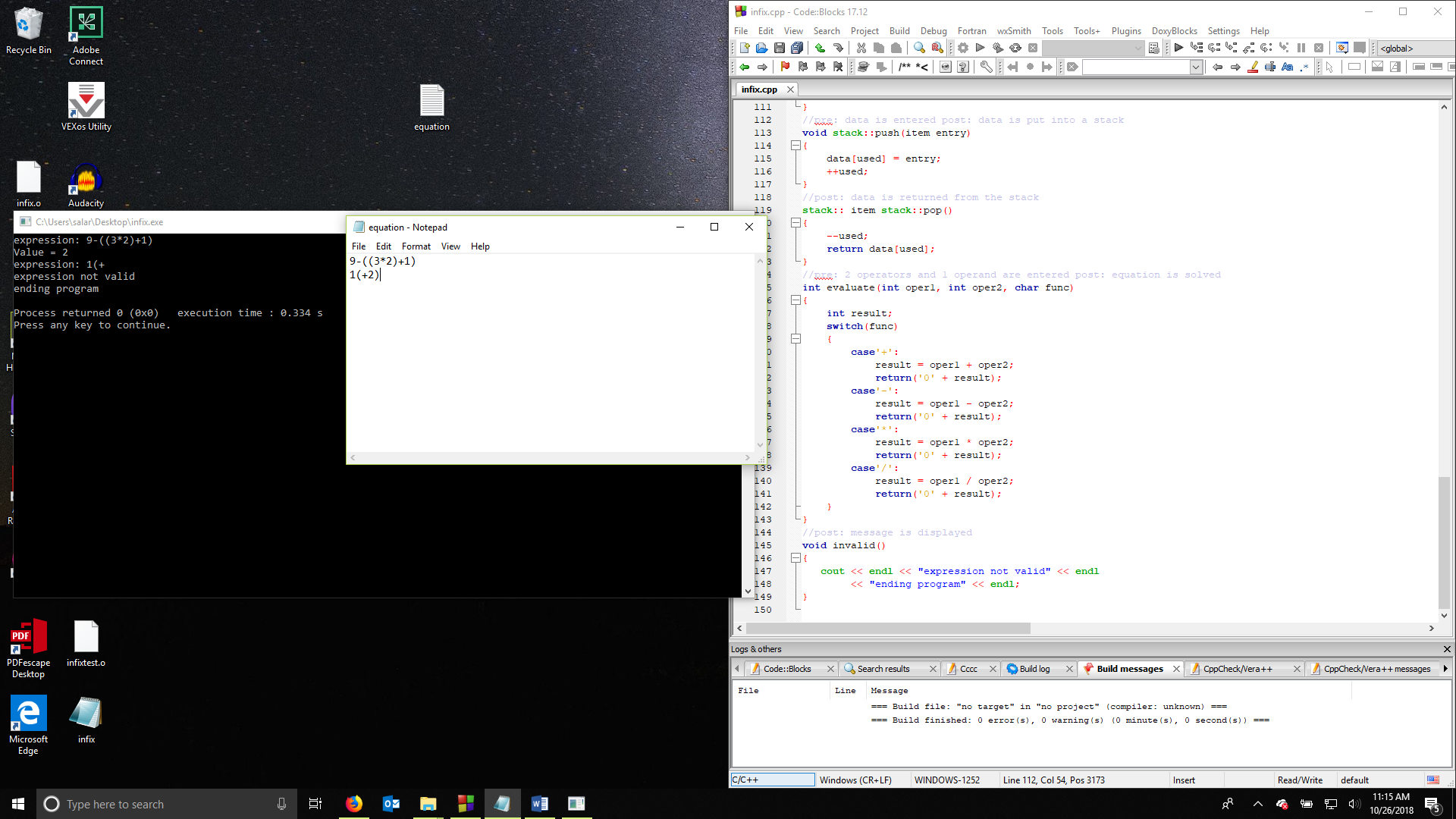
Samuel Larson

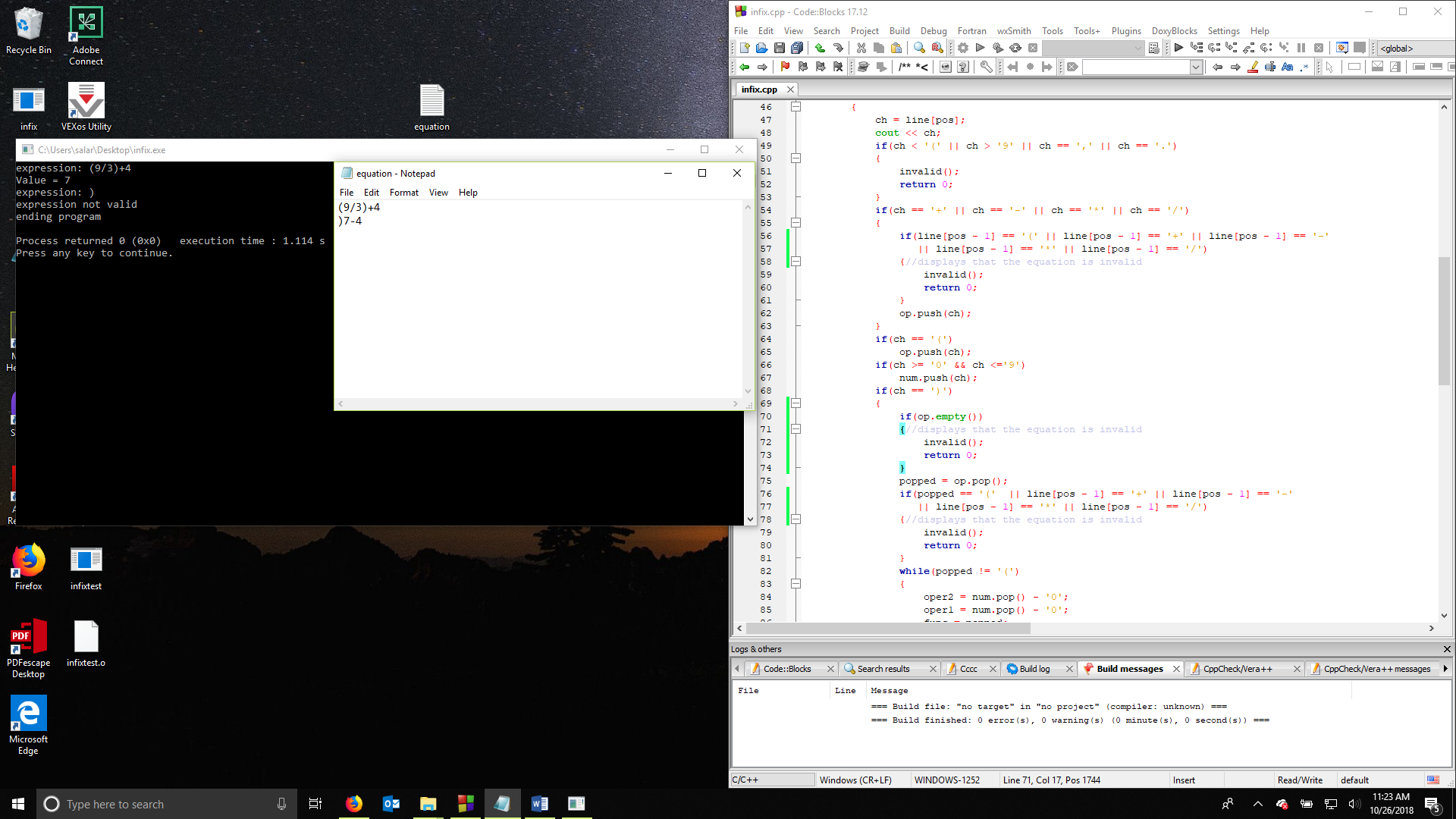
Project 7

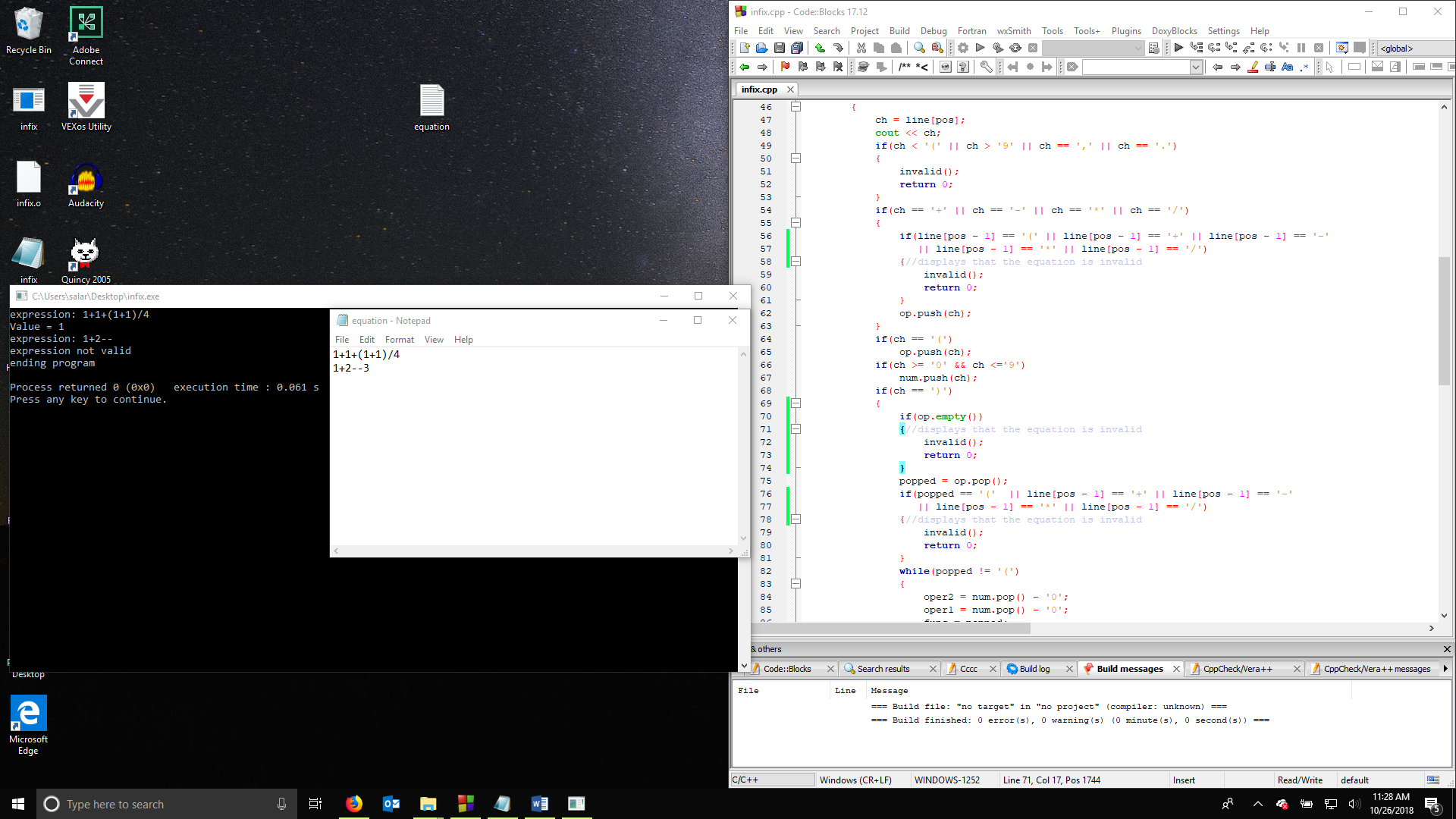
10/30/18











Samuel Larson

Project 7

10/30/18

User Document

To use this program you must have a text file with infix equations named equations.txt in the same directory as the program. The program allows the user to use the operators +,-,\*,/ to manipulate 1 digit numbers.

Examples of possible inputs

1+2

(4-3)

(2\*3)/6

These types of equations will work

Counter examples:

()

1(+2)

)-2++6

These will not work in the program

Samuel Larson

Project 7

10/30/18

Summary

The purpose of this project was to create a program to solve infix expressions by reading from a file and putting the equation 2 different stacks. One stack is for numbers and the other stack is for operators. The program had to solve the equation while also checking for problems in the entered equation. The program had to either return the result of the problem or display a message that the problem is not able to be solved.

The program could have been improved by allowing the user to make corrections on the equation or allow the user to enter equations with multiple digits. Another feature that could have been added to the program is allowing the user to enter equations directly from the terminal. Another thing that would have been good for the program to be able to do is instead of just displaying a message that the equation is invalid it could go onto the next equation.