Samuel Larson

Project 6

10/22/18

The objective of this program is to implement stacks to solve postfix expressions. The program must read in 1 or more expressions one character at a time from a file and put them into a stack, the stack will be used to solve the problems.

The program will be made of a class for the stack and several functions. The class should include data members to hold an array of characters and an integer to tell how many characters are in the stack.

The functions in the class should be push(), pop(), size(), and empty(). The function push() should add a character to the stack and increment the used value. The function pop() should remove a character from the stack and decrement the used value. The size() function should return how many spaces in the stack are used. The empty() function should return if the stack is empty or not.

The program should read a line of text into a string then the string should be read character by character and put into the stack. If the program runs into an operator it should pop the stack 3 times and use the operator on the 2 operands. If there is an operand missing or what is popped is not an operand the program should not try to solve the problem. If the program reads a character that isn’t a number or an operator from the file it should be rejected. Once the expression is solved the program should pop the stack for the result. The program will repeat this until there are no more lines in the file.

//Samuel Larson

//Project 6

//10/22/18

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

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 s;

ifstream infile;

infile.open("equation.txt");

char ch;

string line;

int result, oper1, oper2, pos;

char func;

while(getline(infile, line))

{

cout << "expression: ";

pos = 0;

while(line[pos] >= ' ')

{

ch = line[pos];

cout << ch;

pos++;

s.push(ch);

if(ch < '\*' || ch > '9' || ch == ',' || ch == '.')

{

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

<< "ending program" << endl;

return 0;

}

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

{

func = s.pop();

if(s.empty())

{

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

<< "ending program" << endl;

return 0;

}

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

if(s.empty())

{

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

<< "ending program" << endl;

return 0;

}

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

switch(func)

{

case'+':

result = oper1 + oper2;

ch = '0' + result;

s.push(ch);

break;

case'-':

result = oper1 - oper2;

ch = '0' + result;

s.push(ch);

break;

case'\*':

result = oper1 \* oper2;

ch = '0' + result;

s.push(ch);

break;

case'/':

result = oper1 / oper2;

ch = '0' + result;

s.push(ch);

break;

}

}

}

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

}

return 0;

}

void stack::push(item entry)

{

data[used] = entry;

++used;

}

stack:: item stack::pop()

{

--used;

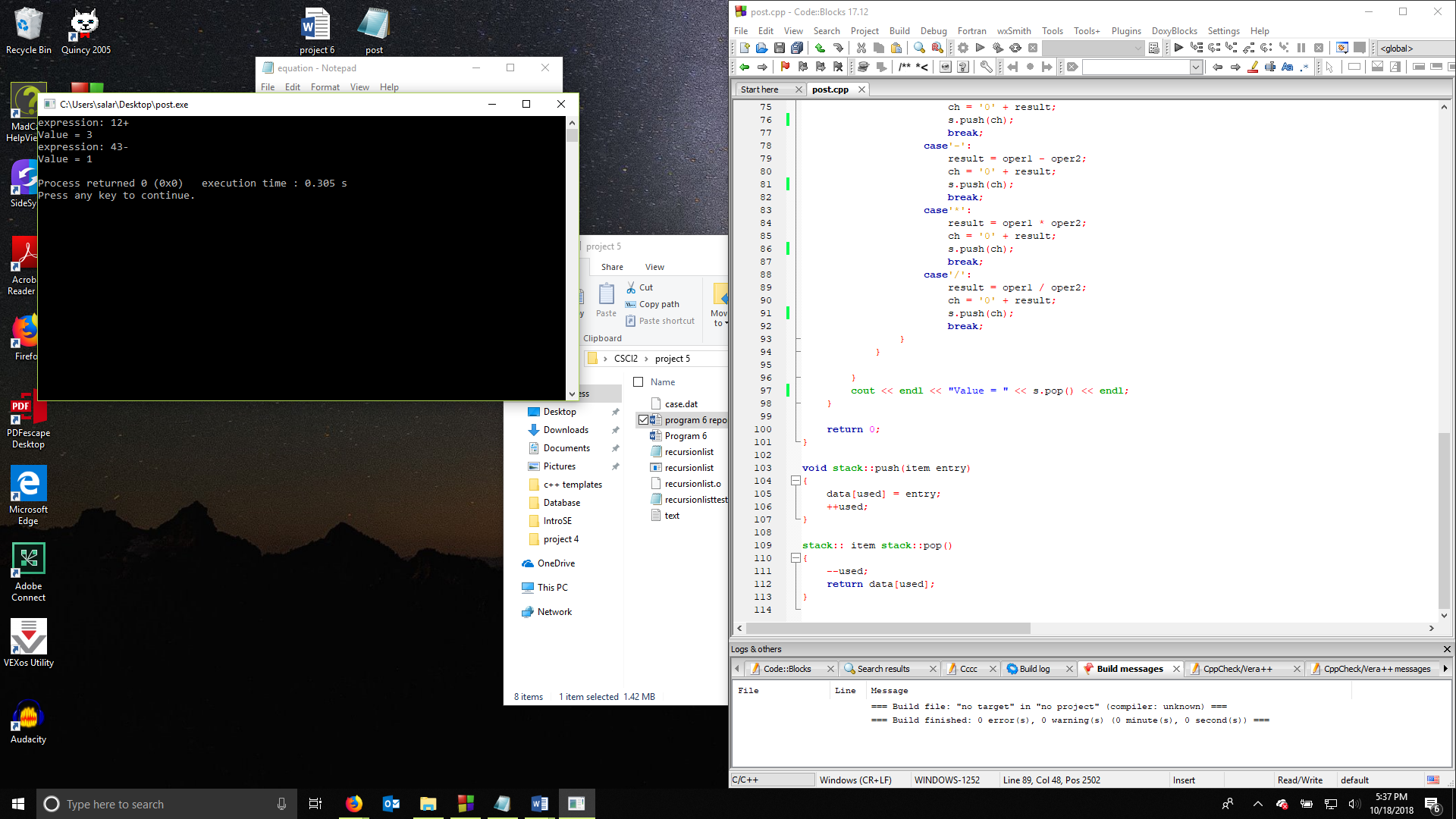
return data[used];

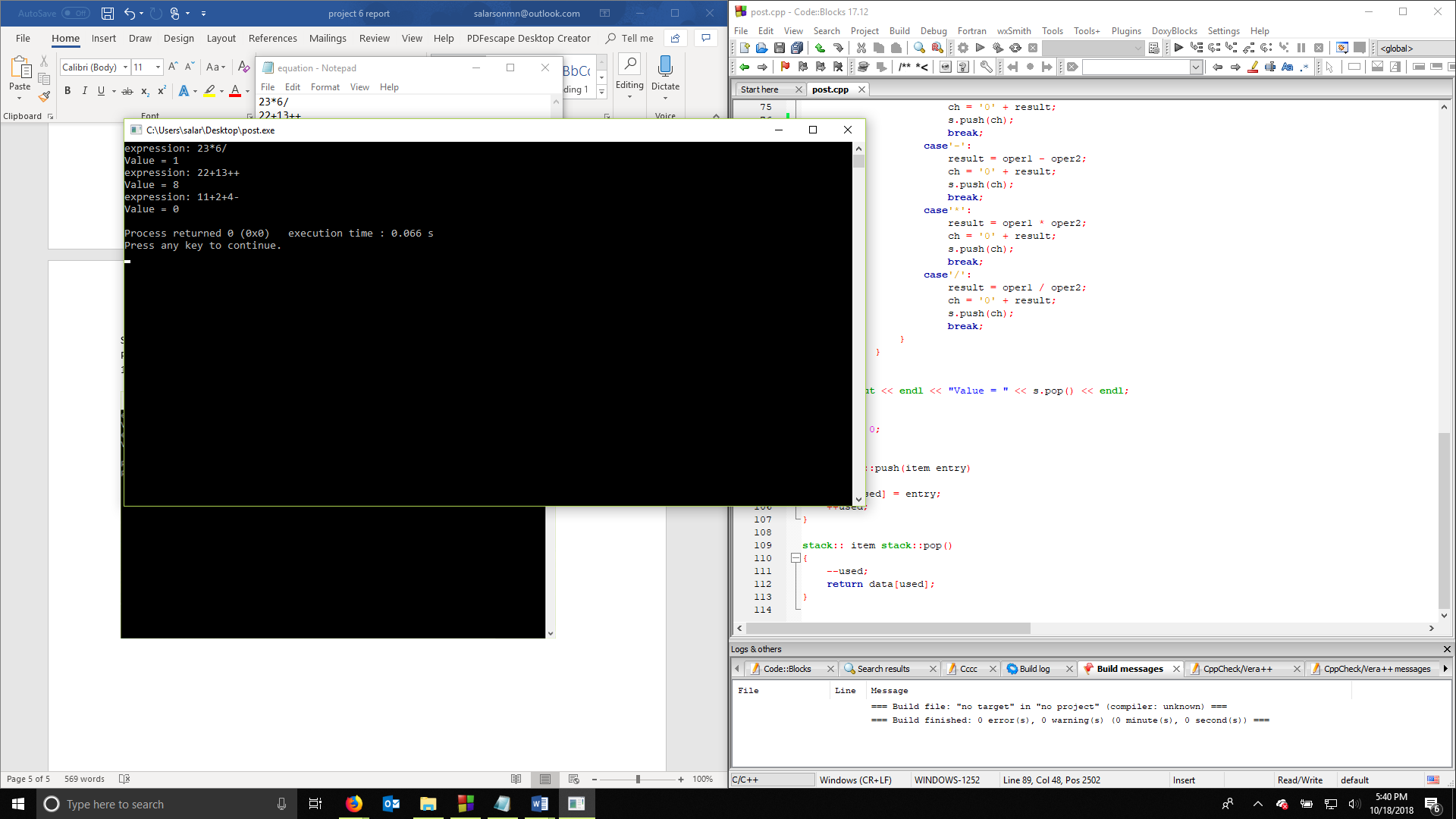
}

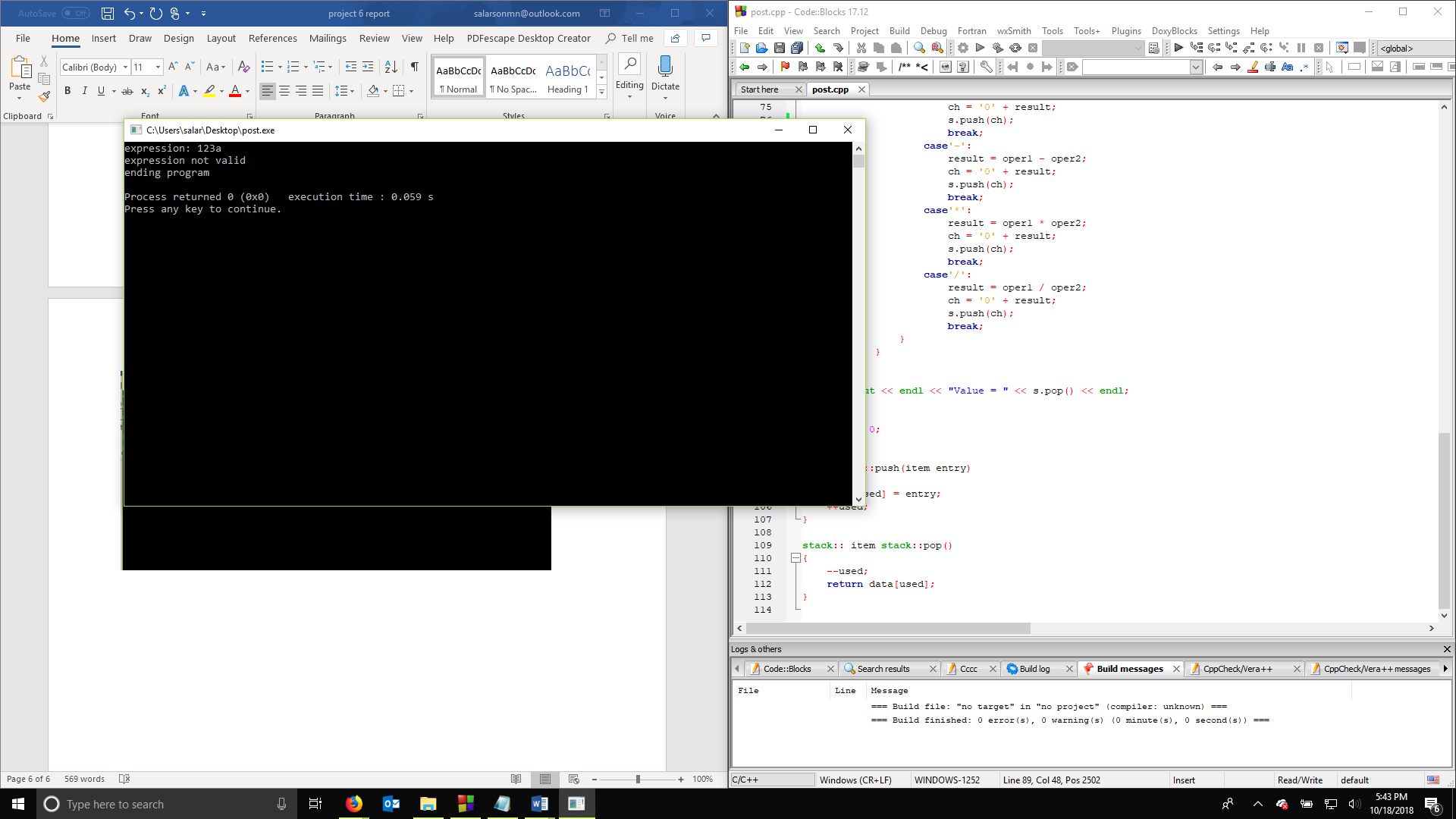
Samuel Larson

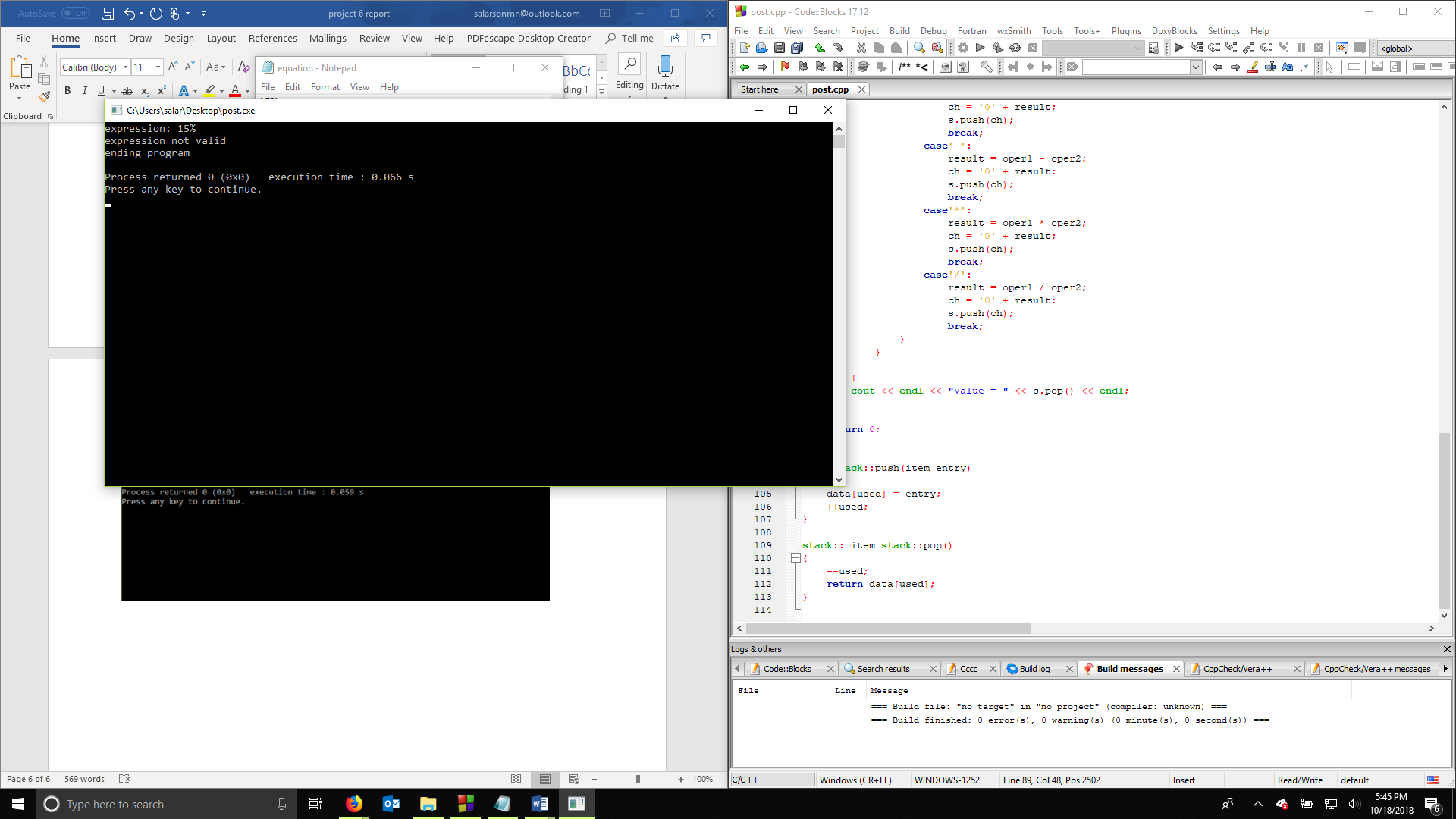
Project 6

10/22/18









Samuel Larson

Project 6

10/22/18

User Document

To use this program you must have a text file with postfix equations 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:

12+

43-

23\*

62/

These will work in the program

Counter examples:

abc

12=

34%

1+2

These will not work in the program

Samuel Larson

Project 6

10/22/18

Summary

The purpose of this project was to create a program that could solve postfix expressions by reading in from a file and putting the equation in a stack. The program had to catch when an operator was entered and use it on the 2 coordinating operands that were entered into the stack. The program had to catch any mistakes in the expression and let the user know that it can’t be solved.

The program could have been improved by allowing the user to correct problems in an expression when an error is found. The program could also have been improved by allowing the user to enter multidigit numbers and negative numbers. I was unable to get the program to read the whole file character by character, so I had to use a string to take one line of the file at a time. The program could have been improved if I found a way to do it without using the string