Anthony Borza

CMSC 330 6380

Due Date: 3/5/17

Project 2 Writeup

a. A Brief Description of your Approach to the Design:

The purpose of this project was to expand our programming skills by extending a C++ program that can evaluate statements of an expression language that was presented in the module 3 case study for this class. The syntax for the expression language is described by, “Black-Naur Form (BNF)”, and regular expressions. The statements of the defined expression language in module 3, consisted of an arithmetic expression followed by a list of assignments. The arithmetic expressions in the language are fully parenthesized infix expressions that contain integer literals, and variables. The arithmetic expressions are stored in a text file, and read in by the program. The program supports multiple expressions, where each expression is on their own line in the text file. When the program reads in the text file, each arithmetic expression is encoded as a binary expression tree. After the expressions in the text file have been read in, the variables and their values of the variables are stored in a symbol table, and then the expression is evaluated recursively.

Furthermore, the valid operators accepted by this program are the following: +, -, *, /. The program also supports, relational, logical, and conditional expression operators, as you will see in the grammar shown below. Each token in the expression can be separated by any number of spaces, without causing complications to the program. The variable names in the program are case sensitive, and start with an alphabetic character, followed by any number of alphanumeric characters. Lastly, for this program to function as intended to, several headers files (.h) and sources files (.cpp) were created.

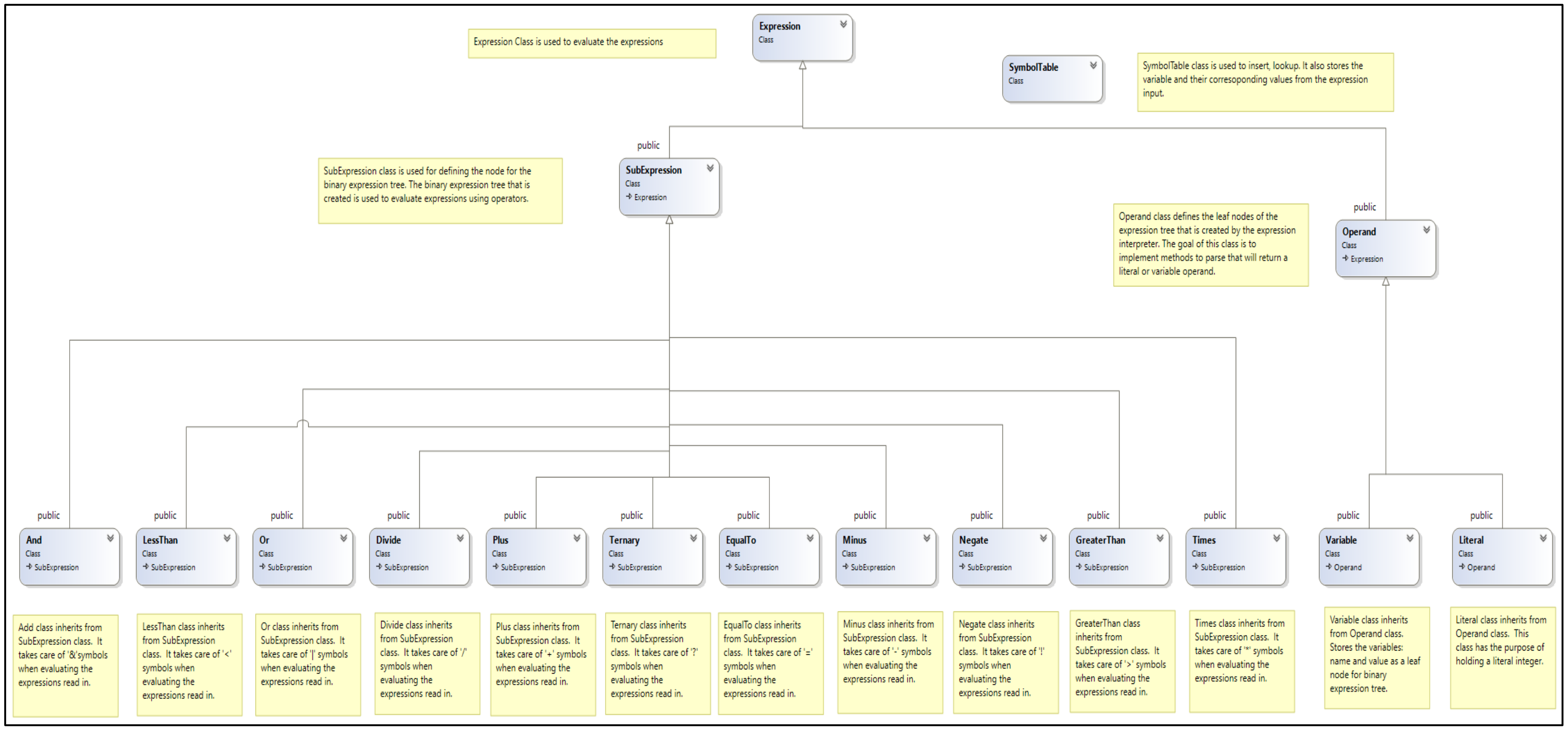
<u>The Expression Language used for this Program:</u>

```
<exp> -> '(' <operand> <op> <operand> ')' |
  '(' <operand> ':' <operand> '?' <operand> ')' |
  '(' <operand> '!' ')'
<op> -> '+' | '-' | '*' | '/' | '>' | '<' | '=' | '&' | '|'
```

b. <u>A UML Class diagram that Includes all Classes Including Any That Were Supplied. Do Not Include Predefined Classes. You Need Only Include the Class Name for Each Individual Class, not the Variables or Methods</u>

The UML class diagram will be attached to the submission box as a separate attachment called: "**ClassDiagram.jpg**." This is for readability purposes. I will also attempt to add the UML class diagram under this section; however, it will probably be hard to read. You will probably need to zoom in on the pdf document. Thanks for your understanding.

Zoom in on PDF for readability purposes. (Recommended Zoom in Size 200% – 225%)



Expression Class is used to evaluate the expressions

SymbolTable class is used to insert, lookup. It also stores the variable and their corresoponding values from the expression input.

SubExpression class is used for defining the node for the binary expression tree. The binary expression tree that is created is used to evaluate expressions using operators.

Operand class defines the leaf nodes of the expression tree that is created by the expression interpreter. The goal of this class is to implement methods to parse that will return a literal or variable operand.

Add class inherits from SubExpression class. It takes care of '&'symbols when evaluating the expressions read in.

LessThan class inherits from SubExpression class. It takes care of '<' symbols when evaluating the expressions read in.

Or class inherits from SubExpression class. It takes care of '|' symbols when evaluating the expressions read in.

Divide class inherits from SubExpression class. It takes care of '/' symbols when evaluating the expressions read in.

Plus class inherits from SubExpression class. It takes care of '+' symbols when evaluating the expressions read in.

Ternary class inherits from SubExpression class. It takes care of '?' symbols when evaluating the expressions read in.

EqualTo class inherits from SubExpression class. It takes care of '=' symbols when evaluating the expressions read in.

Minus class inherits from SubExpression class. It takes care of '-' symbols when evaluating the expressions read in.

Negate class inherits from SubExpression class. It takes care of '!' symbols when evaluating the expressions read in.

GreaterThan class inherits from SubExpression class. It takes care of '>' symbols when evaluating the expressions read in.

Times class inherits from SubExpression class. It takes care of '*' symbols when evaluating the expressions read in.

Variable class inherits from Operand class. Stores the variables: name and value as a leaf node for binary expression tree.

Literal class inherits from Operand class. This class has the purpose of holding a literal integer.

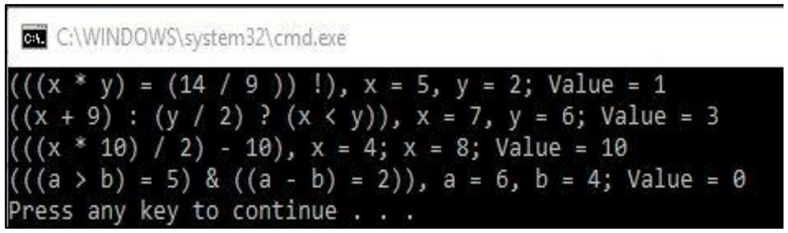
c. A Test Plan that Includes Test Cases that you Have Created Indicating What aspects of the Program Each One is Testing.

- **I will provide three test cases:**

Test Plan Example 1: Reads the following expressions from a text file

Input File:	**Results:**
(((x * y) = (14 / 9))!), x = 5, y = 2;	(((x * y) = (14 / 9)) !), x = 5, y = 2; Value = 1
((x + 9) : (y / 2) ? (x < y)), x = 7, y = 6;	((x + 9) : (y / 2) ? (x < y)), x = 7, y = 6; Value = 3
(((x * 10) / 2) - 10), x = 4; x = 8;	(((x * 10) / 2) - 10), x = 4; x = 8; Value = 10
(((a > b) = 5) & ((a - b) = 2)), a = 6, b = 4;	(((a > b) = 5) & ((a - b) = 2)), a = 6, b = 4; Value = 0

Screenshot of Results:



```
(((x * y) = (14 / 9 )) !), x = 5, y = 2; Value = 1
((x + 9) : (y / 2) ? (x < y)), x = 7, y = 6; Value = 3
(((x * 10) / 2) - 10), x = 4; x = 8; Value = 10
(((a > b) = 5) & ((a - b) = 2)), a = 6, b = 4; Value = 0
Press any key to continue . . .
```

Test Plan Example 2: Reads the following expressions from a text file

Input File:

(x + y), x = 5, y = 7;

(x - y), x = 5, y = 7;

(x * y), x = 5, y = 7;

(x / y), x = 5, y = 7;

(x & y), x = 2, x = 4;

(x > y), x = 9, x = 8;

(x < y), x = 2, x = 4;

(x | y), x = 2, x = 4;

(x!), x = 2;

Results:

(x + y), x = 5, y = 7; Value = 12

(x - y), x = 5, y = 7; Value = -2

(x * y), x = 5, y = 7; Value = 35

(x / y), x = 5, y = 7; Value = 0

(x & y), x = 2, x = 4; Value = 1

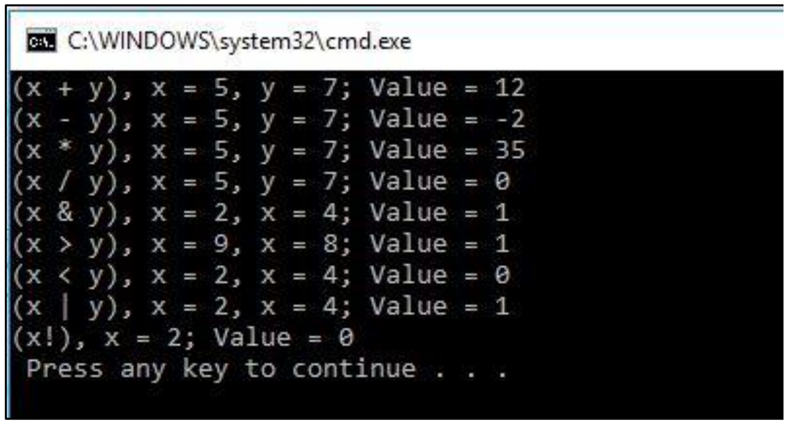
(x > y), x = 9, x = 8; Value = 1

(x < y), x = 2, x = 4; Value = 0

(x | y), x = 2, x = 4; Value = 1

(x!), x = 2; Value = 0

Screenshot of Results:



```
(x + y), x = 5, y = 7; Value = 12
(x - y), x = 5, y = 7; Value = -2
(x * y), x = 5, y = 7; Value = 35
(x / y), x = 5, y = 7; Value = 0
(x & y), x = 2, x = 4; Value = 1
(x > y), x = 9, x = 8; Value = 1
(x < y), x = 2, x = 4; Value = 0
(x | y), x = 2, x = 4; Value = 1
(x!), x = 2; Value = 0
 Press any key to continue . . .
```

Test Plan Example 3: Reads the following expressions from a text file

Input File:

(x & y), x = 1, y = 2;

(x & y), x = 5, y = 6;

(x & y), x = 4, y = 9;

(x & (y & z)), x = 5, y = 4, z = 5;

(x | y), x = 1, y = 2;

(x | y), x = 5, y = 6;

(x | y), x = 4, y = 9;

(x | (y | z)), x = 5, y = 4, z = 5;

(x : y ? z), x = 5, y = 4, z = 5;

(x : y ? z), x = 0, y = 5, z = 6;

(x : y ? z), x = 2, y = 4, z= 8;

(x < y), x = 4, y = 5;

(x > y), x = 5, y = 4;

(x !), x = -1

(x !), x = -2

(x !), x = -4

(x = 2), x = 2;

(x + (4 | 3)), x = 3;

(x - (4 | 3)), x = 4;

(x * (4 | 3)), x = 2;

Results:

(x & y), x = 1, y = 2; Value = 1

(x & y), x = 5, y = 6; Value = 1

(x & y), x = 4, y = 9; Value = 1

(x & (y & z)), x = 5, y = 4, z = 5; Value = 1

(x | y), x = 1, y = 2; Value = 1

(x | y), x = 5, y = 6; Value = 1

(x | y), x = 4, y = 9; Value = 1

(x | (y | z)), x = 5, y = 4, z= 5; Value = 1

(x : y ? z), x = 5, y = 4, z = 5; Value = 5

(x : y ? z), x = 0, y = 5, z = 6; Value = 0

(x : y ? z), x = 2, y = 4, z = 8; Value = 2

(x < y), x = 4, y = 5; Value = 1

(x > y), x = 5, y = 4; Value = 1

(x !), x = -1 Value = 0

(x !), x = -2 Value = 0

(x !), x = -4 Value = 0

(x = 2), x = 2; Value = 1

(x + (4 | 3)), x = 3; Value = 4

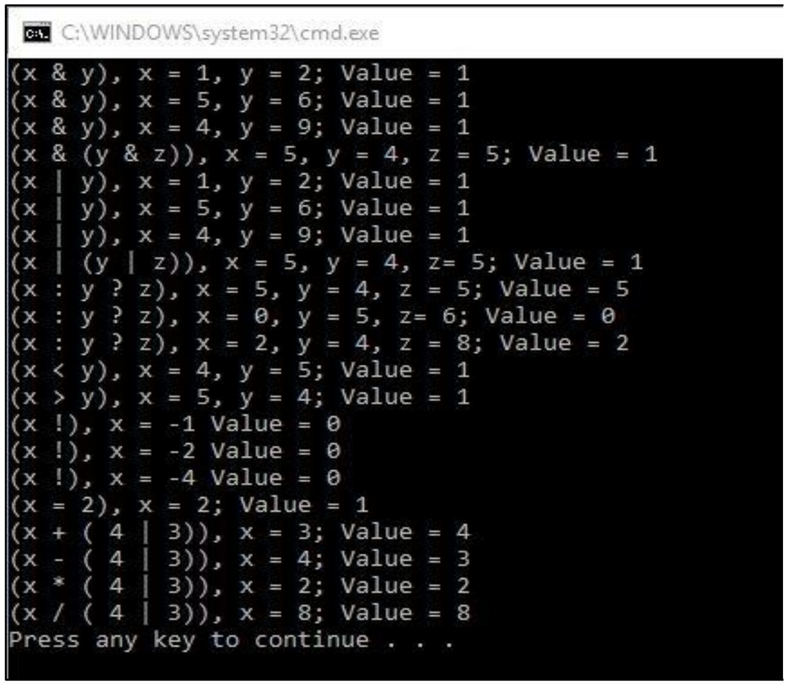
(x - (4 | 3)), x = 4; Value = 3

(x * (4 | 3)), x = 2; Value = 2

(x / (4 | 3)), x = 8; Value = 8

(x / (4 | 3)), x = 8;

Screenshot of Results:



C:\WINDOWS\system32\cmd.exe

```
(x & y), x = 1, y = 2; Value = 1
(x & y), x = 5, y = 6; Value = 1
(x & y), x = 4, y = 9; Value = 1
(x & (y & z)), x = 5, y = 4, z = 5; Value = 1
(x | y), x = 1, y = 2; Value = 1
(x | y), x = 5, y = 6; Value = 1
(x | y), x = 4, y = 9; Value = 1
(x | (y | z)), x = 5, y = 4, z= 5; Value = 1
(x : y ? z), x = 5, y = 4, z = 5; Value = 5
(x : y ? z), x = 0, y = 5, z= 6; Value = 0
(x : y ? z), x = 2, y = 4, z = 8; Value = 2
(x < y), x = 4, y = 5; Value = 1
(x > y), x = 5, y = 4; Value = 1
(x !), x = -1 Value = 0
(x !), x = -2 Value = 0
(x !), x = -4 Value = 0
(x = 2), x = 2; Value = 1
(x + ( 4 | 3)), x = 3; Value = 4
(x - ( 4 | 3)), x = 4; Value = 3
(x * ( 4 | 3)), x = 2; Value = 2
(x / ( 4 | 3)), x = 8; Value = 8
Press any key to continue . . .
```

d. <u>A Short Paragraph on Lessons Learned from the Project:</u>

After completing project 2, there are many things that I learned throughout the process of extending a C++ program that evaluates statements of an expression language that was provided in a module 3 case study earlier in this class. Prior to starting this project, I was already familiar with arithmetic expressions, and how to encode the expressions as a binary tree. I completed a similar assignment in my Data Structures class; however, it was done using the programming language Java. The first thing i did before anything, like with all programs I write, is to first understand what is being asked, and what is the best, and most efficient way for completing it. I really found the weekly lecture: Module 3: Imperative Languages – Control Flow, week 4 PowerPoint and code on Expressions and Statements to be very helpful in the development of my C++ program.

Furthermore, the most valuable parts of the PowerPoint, and weekly lecture were, the step-by-step diagrams. It allowed me to visually see what was happening, and when it came time to coding, I was very happy I took time aside to go through the readings, as it was very beneficial to my success on this project. The hardest part for me during this project was being able to adapt and understand a programming language that I was unfamiliar with. C++ is a little more complex then java, in my opinion; however, I found starting to work on this program weeks (4) in advanced to be very beneficial. I also found guidance and suggestions from the Professor, and other students, to be very helpful. Overall, I feel that I could deliver an efficient, and functional program that meets all requirements asked in the instructions. I am very satisfied with the results of my program, and look forward to expanding my knowledge in future computer science classes, and throughout my career.