Kush Patel
T00640682

CPSC 3383: Programming Languages

Final Report

1. INTRODUCTION

Assignment 1.1 requires a program to be written which acts as a simulator for handling computations in reverse polish notation [1] (RPN). When given an input of a math equation in RPN the program must identify tokens, compare the input to a set of grammar rules, and perform the proper operations relative to the inputs given. This was done by using Flex [2] and Bison [3] which are a lexical analyzer and prayer.

2. TOOLS AND LANGUAGES

The tools used are based on the C language. To edit and write the C code Notepad++ [4] was utilized. To compile MinGW [5] was used. In addition to these tools, the Windows native CMD was used to interact with the tools within MinGW as well as the command line interactions with Lex [6] and Yacc [7].

3. LEX AND YACC

To get this program to work we need a process that takes the input and processes them into tokens we define. To do this we need to use a lexical analyzer called Lex. We give Lex a set of rules in an 'L' file. With this, we can generate a C file called 'lex.yy.c'. Then we use Yacc to create a parser that will execute C snippets associated with each rule as soon as the rule is recognized. These rules are defined in a 'Y' file which will generate the 'y.tab.c' and the 'y.tab.h' files. The 'L' file contains a reference to the 'y.tab.h' which allows communication between the Lex and Yacc. When the two C files are compiled with MinGW we get a '.exe' that contains our final program. To compile we use:

```
gcc lex.yy.c y.tab.c -o rpnCalc.exe
```

in the CMD in the directory where the 'L', 'Y', 'c', and 'h' files are located. It will create a new program called 'rpnCalc.exe'.

4. FLEX

Lex is a tool available for the Unix environment and is not available for Windows, but there is an open-source variant of Lex called Flex which is available for use in Windows. Which is used in this project in place of Lex. The file type remains the same and is still 'L'. This file consists of the definitions for the tokens and variables. One example of a variable is a digit which can be express as:

```
digit           [0-9]
```

another example of this is an integer which is a number that may have more than one digit:

```
integer         {digit}+
```

the operations are stored as the tokens an example of this is plus:

```
"+"             {return PLUS;}
```

This tells the lexer when it sees '+' in the input to return plus so the correct operation can take place. We can use the flex CMD tool to create the 'lex.yy.c' file by running the following in CMD:

```
flex rpnCalc.L
```

This works if the directory is set to the location of the 'rpnCalc.L' file.

5. BISON

Like Lex, Yacc is only available on Unix systems. But like Flex there is an open-source variant called Bison provided by GNU. The file type remains a 'Y' file. It must contain the rules for the tokens and variables, and this file also contains the main function which runs during the program. One example of the rule for float addition is:

```
float_exp float_exp PLUS        { $$ = $1 + $2; }
```

this tells the program when there is a format of "float float +" it will do float + float and return the result. To make the 'h' and 'c' file from the 'Y' file you must run:

```
bison -dy rpnCalc.Y
```

in the CMD when the directory is set to the location of the 'rpnCalc.Y' file. This will create a 'y.tab.c' and 'y.tab.h' file in this directory.

6. FUNCATALITY AND USE CASES

This program takes in a user input and assigns tokens to all variables, operations, and for the left and right parenthesis. Then it will go through the grammar rules and perform the operations as described in the 'Y' bison file. One feature to note or the lack thereof is a rule for handling parenthesis this is because in reverse polish notation the format of the input will be carried out the same whether there is a parenthesis or not. Examples of this are the equations:

[FIGURE 1] (3.141 (2 3 +) (1.571 SIN) *) ~= 15.705 &

[FIGURE 2] 3.141 2 3 + 1.571 SIN * ~= 15.705

```
       TOKEN IDS:
int:            258
float:          259
'+':            260
'-':            261
'*':            262
'**':           263
'/':            264
'(':            265
')'             266
sin:            267
cos:            268
tan:            269
enter:          270
-------------------------
Enter problem:
(3.141 (2 3 +) (1.571 sin)*)
Result: 15.705
```

[FIGURE 1]

```
       TOKEN IDS:
int:            258
float:          259
'+':            260
'-':            261
'*':            262
'**':           263
'/':            264
'(':            265
')'             266
sin:            267
cos:            268
tan:            269
enter:          270
-------------------------
Enter problem:
3.141 2 3 + 1.571 sin *
Result: 15.705
```

[figure 2]

7. CONCLUSION

Lex and Yacc are used for creating compilers for different programming languages which is essentially the assignment. We are required to take a math problem in RPN and have a program understand and solve for the correct answer to do this we have to create a way for the computer to understand what we are inputting. To do this we defined our grammar rules in the parse to identify parts of our input and solve in RPN. These tools are very dependent on the environment for example Lex and Yacc are native to Unix, whereas Flex and Bison work on windows. Regardless of the tools work as expected and work well together given you have the right software.

8. REFERENCES

1. https://en.wikipedia.org/wiki/Reverse_Polish_notation
2. https://en.wikipedia.org/wiki/Flex_(lexical_analyser_generator)
3. https://www.gnu.org/software/bison/
4. https://notepad-plus-plus.org/
5. http://www.mingw.org/
6. https://en.wikipedia.org/wiki/Lex_(software)
7. https://en.wikipedia.org/wiki/Yacc