COMMAND LINE CALCULATOR

18CSC304J – Compiler Design

TEAM MEMBERS

- GELLE THARUN – RA2011003010660
- ANGADA CHANDRA MOULI – RA2011003010664
- EKKULURI RAJESH – RA2011003010669

Introduction

- A command line calculator which supports mathematical expressions with scientific functions is very useful for most developers.
- The calculator available with Windows does not support most scientific functions
- The most difficult part we found when designing such a calculator was the parsing logic.
- Later while working with .NET, the runtime source code compilation made the parsing logic easy and interesting.
- It uses runtime compilation and saves the variables by serializing in a file.
- Thus you can get the values of all the variables used in the previous Calculation.

Problem Statement

The calculate function calculates an expression. It uses the saved variables. I have generated code which has a declaration of the variables

- To Evaluate the given expressions.
- To perform basic calculations

Objectives

The command line calculator is to be capable of parsing a human-readable mathematical expression with units, return the value if it can be evaluated and inform the user about the position of an error if not.

Requirements

SOFTWARE REQUIREMENTS:

- Windows/Ubuntu Operating System
- C Programming Language

HARDWARE REQUIREMENTS :

- Minimum of 4GB RAM

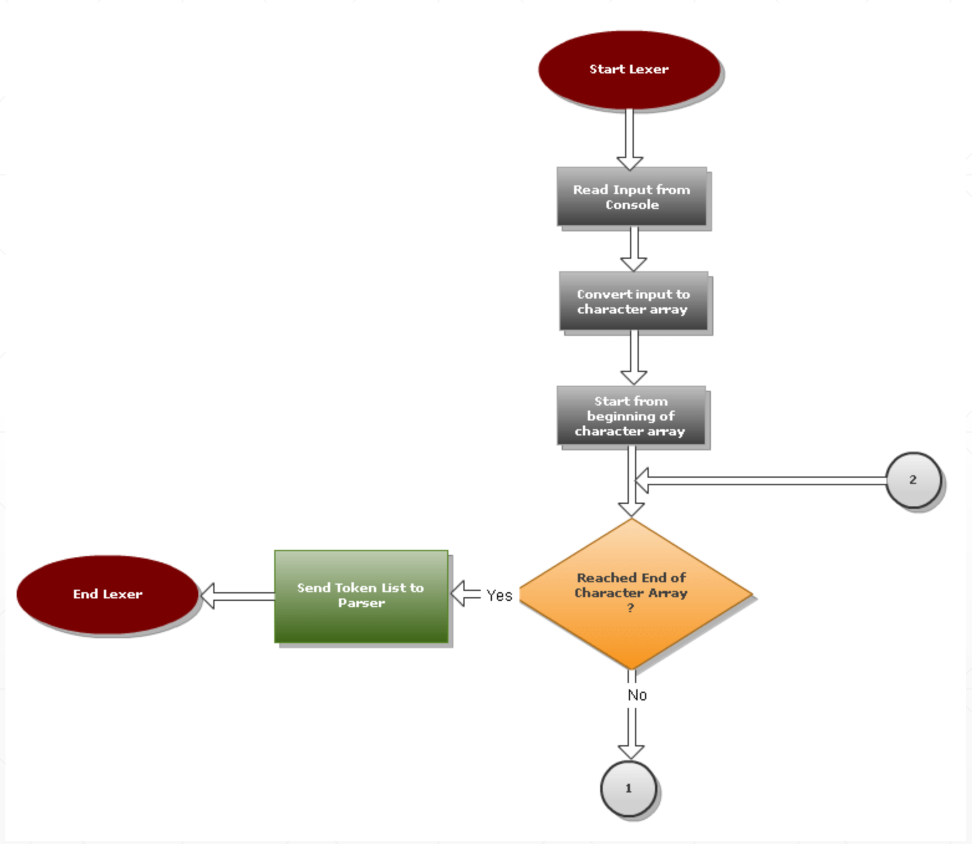
LEXER

- A LEXER is a software program that performs lexical analysis.
- Lexical analysis is the process of separating a stream of characters into different words, which in computer science we call 'tokens' .
- For Example -While reading we are performing the lexical operation of breaking the string of text at the space characters into multiple words.

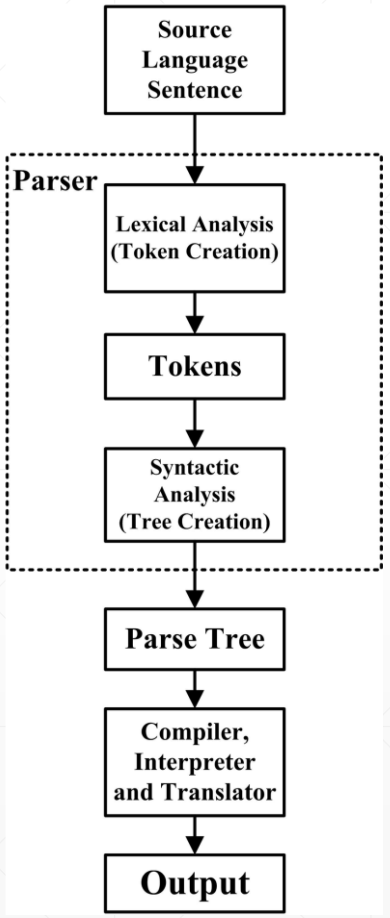
PARSER

- A parser goes one level further than the lexer and takes the tokens produced by the lexer and tries to determine if proper sentences have been formed.
- Parsers work at the grammatical level, lexers work at the word level.
- Generally yacc is used to parse language syntax.
- Yacc uses a parsing technique known as LR-parsing or shift-reduce parsing.

LEXER


Start Lexer
Read Input from Console
Convert input to character array
Start from beginning of character array
2
Reached End of Character Array ?
Yes
Send Token List to Parser
End Lexer
No
1

PARSER


Source Language Sentence
Parser
Lexical Analysis (Token Creation)
Tokens
Syntactic Analysis (Tree Creation)
Parse Tree
Compiler, Interpreter and Translator
Output

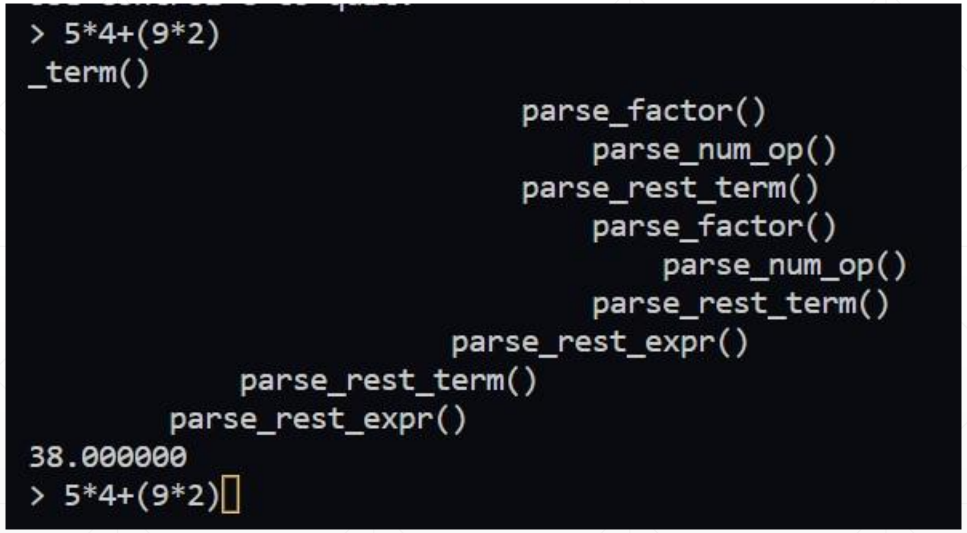
Implementation and Source Code

- Implemented in C
- Source Code: https://github.com/neil-03/CLI-calculator

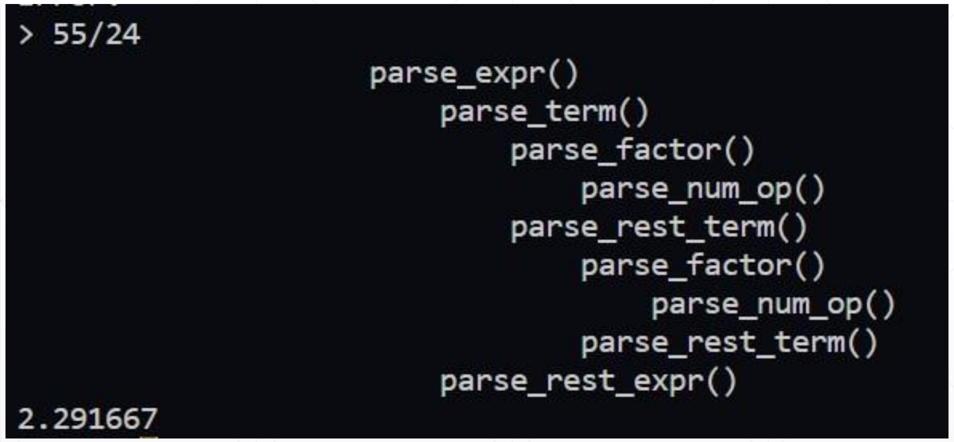
Demo and Results

5*4+(9+2) = 38.000000

55/24 = 2.291667



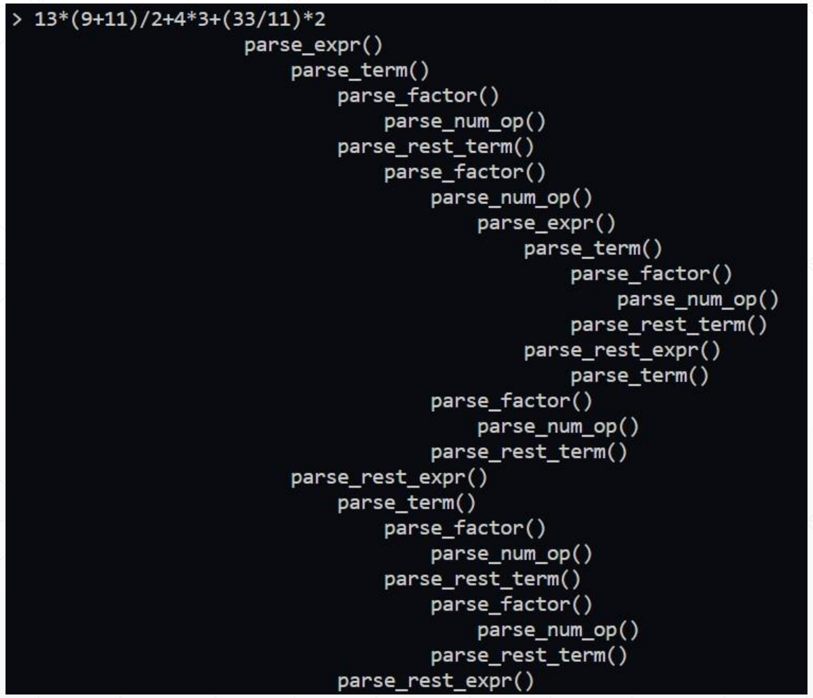
```
> 5*4+(9*2)
_term()
              parse_factor()
                parse_num_op()
              parse_rest_term()
                parse_factor()
                  parse_num_op()
                parse_rest_term()
            parse_rest_expr()
      parse_rest_term()
    parse_rest_expr()
38.000000
> 5*4+(9*2)
```



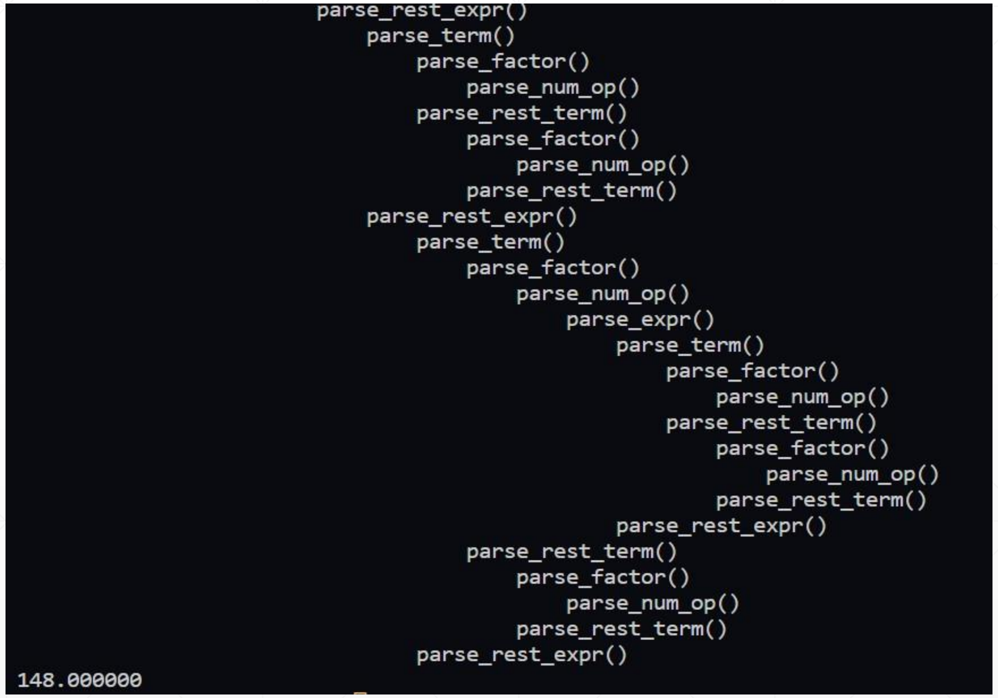
```
> 55/24
          parse_expr()
            parse_term()
              parse_factor()
                parse_num_op()
              parse_rest_term()
                parse_factor()
                  parse_num_op()
                parse_rest_term()
            parse_rest_expr()
2.291667
```

Demo and Results (contd…)

13*(9+11)/2+4*3+(33/11)*2 = 148.000000



```
> 13*(9+11)/2+4*3+(33/11)*2
                    parse_expr()
                        parse_term()
                            parse_factor()
                                parse_num_op()
                            parse_rest_term()
                                parse_factor()
                                    parse_num_op()
                                        parse_expr()
                                            parse_term()
                                                parse_factor()
                                                    parse_num_op()
                                                parse_rest_term()
                                            parse_rest_expr()
                                                parse_term()
                                    parse_factor()
                                        parse_num_op()
                                    parse_rest_term()
                        parse_rest_expr()
                            parse_term()
                                parse_factor()
                                    parse_num_op()
                                parse_rest_term()
                                    parse_factor()
                                        parse_num_op()
                                    parse_rest_term()
                            parse_rest_expr()
```



```
                        parse_rest_expr()
                            parse_term()
                                parse_factor()
                                    parse_num_op()
                                parse_rest_term()
                                    parse_factor()
                                        parse_num_op()
                                    parse_rest_term()
                            parse_rest_expr()
                                parse_term()
                                    parse_factor()
                                        parse_num_op()
                                            parse_expr()
                                                parse_term()
                                                    parse_factor()
                                                        parse_num_op()
                                                    parse_rest_term()
                                                        parse_factor()
                                                            parse_num_op()
                                                        parse_rest_term()
                                                parse_rest_expr()
                                    parse_rest_term()
                                        parse_factor()
                                            parse_num_op()
                                        parse_rest_term()
                                parse_rest_expr()
148.000000
```

Conclusion

This is a powerful and versatile command-line calculator that really lives up to your expectation. Preloaded on all modern Linux distributions, this can make your number crunching tasks much easier to handle without leaving your terminals. Besides, if your shell script requires floating point calculation, can easily be invoked by the script to get the job done. All in all, CLC should definitely be in your productivity tool set.

References

[1] - https://www.codeproject.com/Articles/12395/A-Command-Line-Calculator

[2] - https://fedoramagazine.org/bc-command-line-calculator/

```
p 15:53 .
ep 15:53 ..
ep 2015 bin -> usr/bin
Sep 09:31 boot
Sep 15:50 dev
Sep 09:32 etc
Sep 15:52 home
Sep 2015 lib -> usr/lib
. Sep 2015 lib64 -> usr/lib
. Jul 10:01 lost+found
1. Aug 22:45 mnt
0. Sep 2015 opt
1. Sep 15:52 private -> /home/encry
21. Sep 08:15 proc
12. Aug 15:37 root
21. Sep 15:50 run
30. Sep 2015 sbin -> usr/bin
30. Sep 2015 srv
21. Sep 15:51 sys
21. Sep 15:45 tmp
12. Aug 15:39 usr
23. Jul 10:25 var
```

THANK YOU

GELLE THARUN – RA2011003010660

ANGADA CHANDRA MOULI-RA2011003010664

EKKULURI RAJESH – RA2011003010669