18CSC304J

COMPILER DESIGN

COMMAND LINE CALCULATOR

MINOR PROJECT REPORT

Submitted by

G. Tharun (RA2011003010660)

A.Chandra Mouli(RA2011003010664)

E. Rajesh (RA2011003010669)

Under the guidance of

Dr.G.ABIRAMI

for the course

18CSC304J-Compiler Design

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

S.R.M. Nagar, Kattankulathur, Kancheepuram

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report "**Command Line Calculator**" is the bonafide work of G. Tharun(RA2011003010660),A.Chandra Mouli(RA2011003010664)and E. Rajesh(RA2011003010669) who carried out the project work under my supervision.

SIGNATURE

Dr.G.Abirami

CD Faculty

CSE

SRM Institute of Science and Technology,
Potheri, SRM Nagar, Kattankulathur
Tamil Nadu 603203

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy**, for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal**, for bringing out novelty in all executions.

We would like to express my heartfelt thanks to **Chairperson, School of Computing Dr. Revathi Venkataraman**, for imparting confidence to complete

my course project

We wish to express my sincere thanks to **Course Audit Professor** for their constant encouragement and support.

We are highly thankful to our Course project Internal guide **Dr.G.Abirami , Compiler Design Faculty , CSE**, for her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to the **Dr. M. PUSHPALATHA, Ph.D HEAD OF THE DEPARTMENT** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project

TABLE OF CONTENT

S.no	Content	Page no.
1.	Introduction	5
2.	Problem Statement	5
3.	Objectives	6
4.	Algorithms	8
5.	Demo & Results	9
6.	Conclusion	11
7.	References	11

COMMAND LINE CALCULATOR

18CSC304J – Compiler Design Mini Project Report

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:

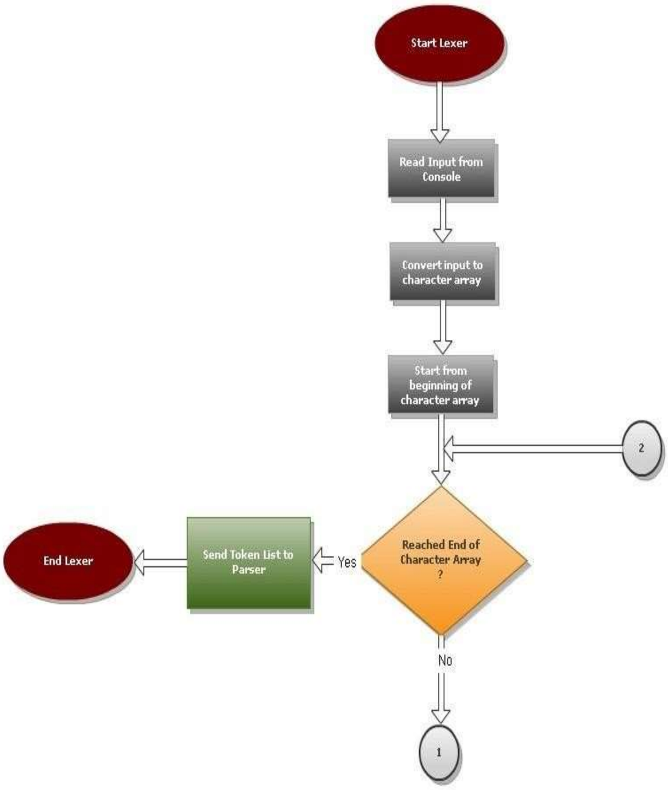
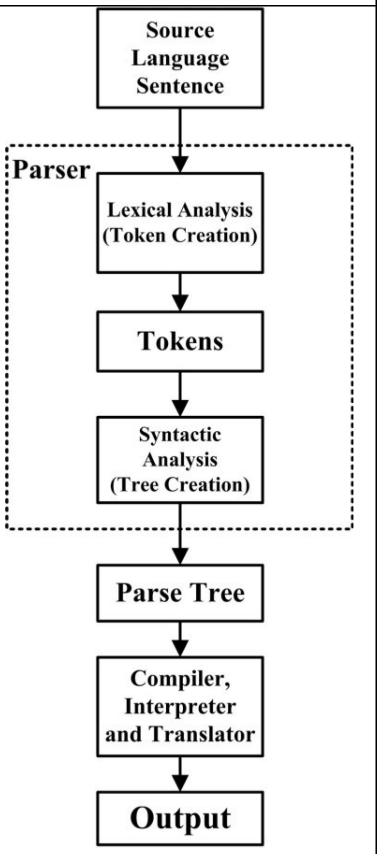
- Minimum of 4GB RAM

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

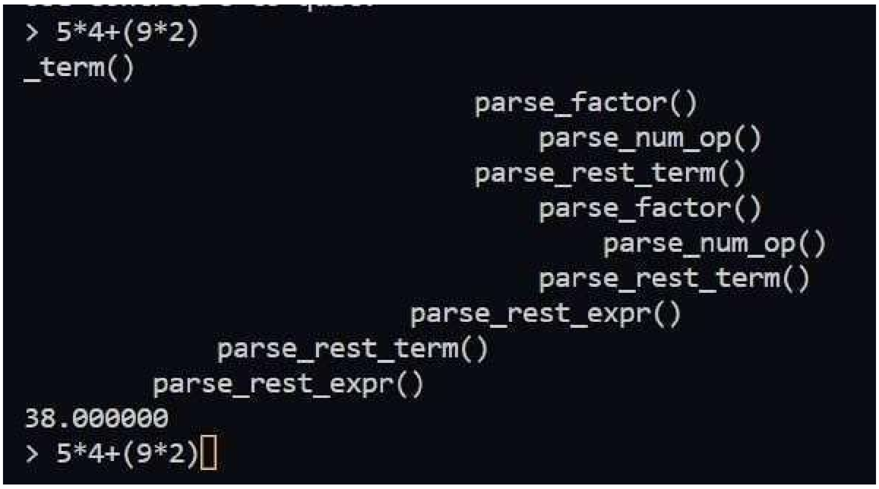
IMPLEMENTATION AND SOURCE CODE

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

ALGORITHM

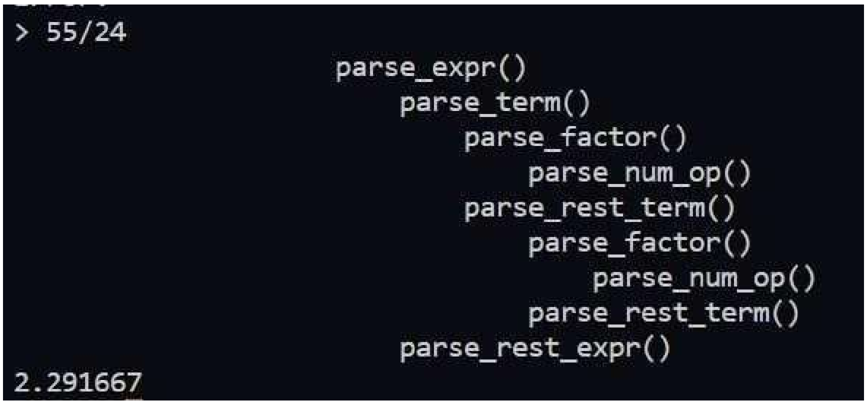
```
/*
expr := var rest_var
     term rest_expr
rest_expr := + term rest_expr
        - term rest_expr
        (nil)
term := factor rest_term
rest_term := * factor rest_term
        / factor rest_term
        % factor rest_term
        <nil>
factor := - factor
      num_op
num_op := num rest_num_op
      variable rest_num_op
      ( expr )  rest_num_op
rest_num_op := ^ num_op rest_num_op
          <nil>
rest_var := '=' expr
        rest_num_op
*/
```

DEMO AND RESULTS



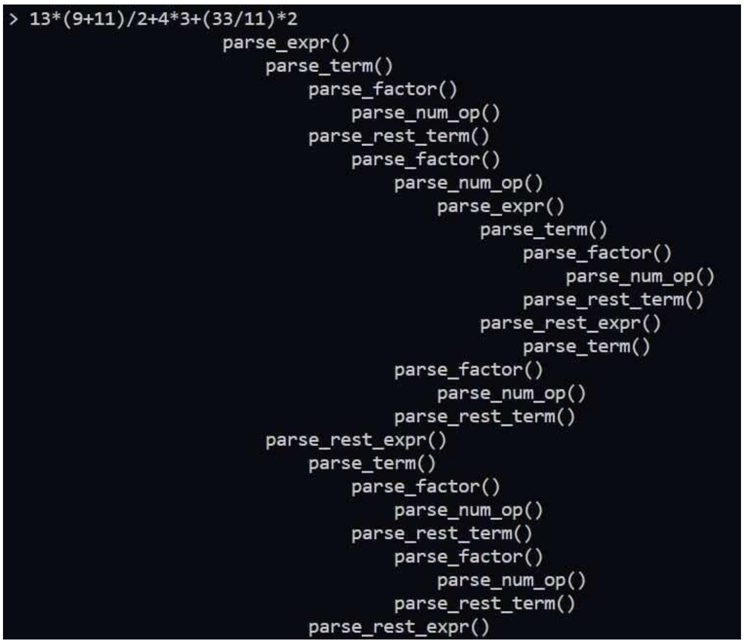
```
> 5*4+(9*2)
_term()
                            parse_factor()
                                parse_num_op()
                            parse_rest_term()
                                parse_factor()
                                    parse_num_op()
                                parse_rest_term()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

5*4+(9*2) = 38.000000

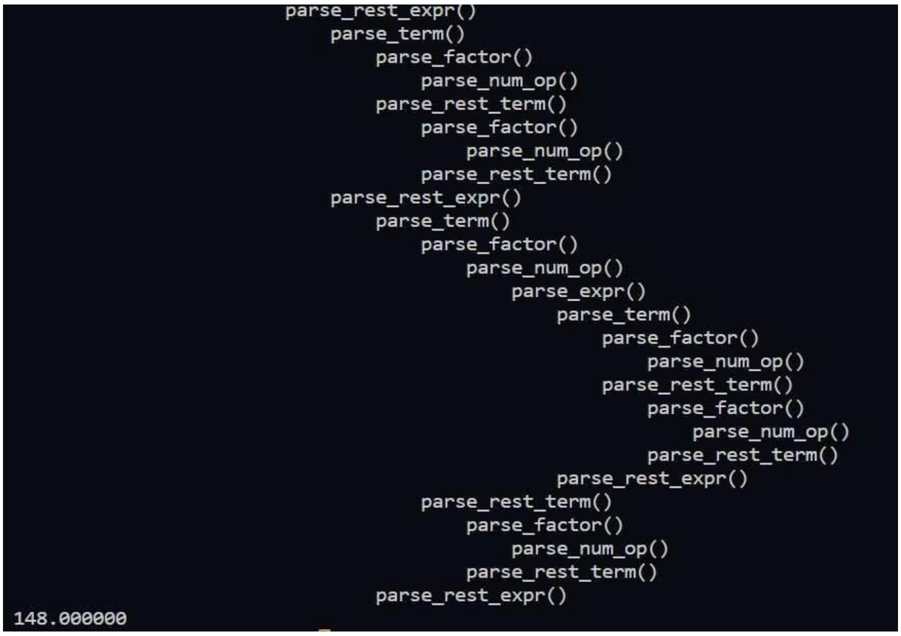


```
> 55/24
                    parse_expr()
                        parse_term()
                            parse_factor()
                                parse_num_op()
                            parse_rest_term()
                                parse_factor()
                                   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

55/24 = 2.291667



```
> 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
```

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

SUBMITTED BY

GELLE THARUN – RA2011003010660

ANGADA CHANDRA MOULI – RA2011003010664

EKKULURI RAJESH – RA2011003010669