OS-CalcCompiler

This project involves a calculator which is built using YACC and LEX to illustrate compiling process and how compilers identify tokens and enforce grammar rules during a compilation of a program.

This project elevates the traditional calculator experience by introducing advanced features such as variable assignments and chained operations. This project not only showcases the fundamental aspects of compiler design but also illustrates the practical implementation of versatile and user-centric functionalities.

The calculator supports addition, subtraction, multiplication, division, modulus and exponentiation operations. It provides up to 3 decimal point precision and can perform arithmetic operations on any number of variables and constants.

The calculator program supports 2 basic keywords print and exit followed by a semicolon and a new line character. The program can have up to 52 variables which are identified by letters from English alphabet (a-z and A-Z).

Lexical analyzers identify tokens in the program using regular expressions and pass them to the parser. The program displays error messages for invalid tokens. YACC is used to build the semantic tree with the given grammar rules and generate the compiler for the calculator program.

TABLE OF CONTENT

About the Author

Pubudu Perera is a third-year Computing student at Coventry University, with a strong interest in compiler design, software security, and application development. As the primary developer for the OS-CalcCompiler project, Pubudu contributed to designing and implementing the core functionalities, including syntax analysis, token parsing, and error handling using Lex and YACC. This project allowed him to deepen his understanding of compiler construction principles and the practical application of lexical analysis and syntax parsing.

For questions, feedback, or further information, feel free to contact Pubudu:

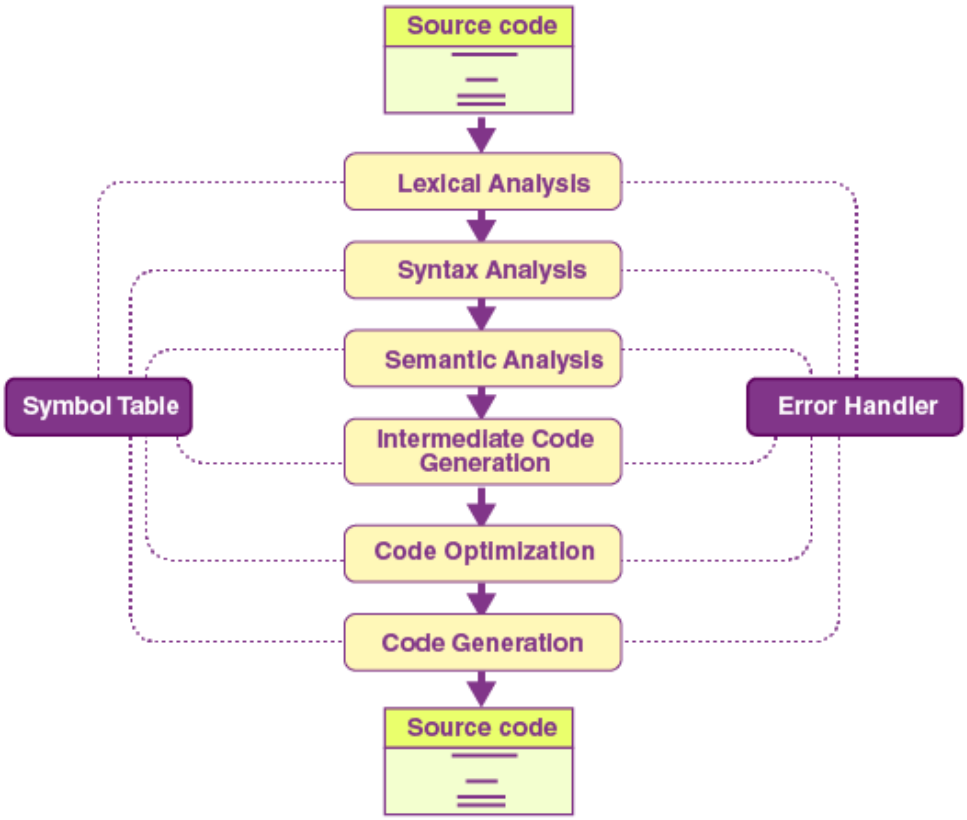
- Email: pubupere32@gmail.com
- LinkedIn: https://www.linkedin.com/in/pubudu-perera-se/
- GitHub: https://github.com/Pamod45

CHAPTER 01

1.1 Introduction

This report describes the design and implementation of a syntax analyzer for a simple calculator using the LEX and YACC tools in the vi editor. The syntax analyzer is a program that checks the source code of a program to ensure that it is syntactically correct. This is important because syntactically incorrect code cannot be compiled and run correctly.

This program supports exponentiation, modulus and BODMAS in addition to performing basic operations.



1.2 Features of the Product

- It is able to check for the following arithmetic operations with any number of operands: addition, subtraction, multiplication, division, modulus and exponentiation.
- Bracketed statements are supported, following the BODMAS (Brackets, powers, Division and Multiplication, and Addition and Subtraction) rule.
- Supports the assignment of variables, accommodating up to 52 variable assignments.
- Semicolon is used as the delimiter for separating and for marking the end of a statement.
- “print” and “exit” keywords are provided for user convenience.
- It is implemented using the LEX and YACC tools, which are widely used and supported tools for implementing lexical analyzers and parsers.
- It is designed to be easy to use and understand.
- It is well-documented and includes screenshots to demonstrate how to compile and run the program, as well as screenshots of the output with valid and invalid expressions.

CHAPTER 02

2.1 Lex Code

```
%{
   #include "y.tab.h"
   void yyerror(char *s);
   int yylex();
%}

%%

[\n]       { return '\n';}
"print"    { return print;}
"exit"     { return exit_command;}
[a-zA-Z]   { yylval.id=yytext[0]; return identifier;}
[0-9]+(\.[0-9]+)?    { yylval.num=strtod(yytext,NULL); return number;}
[ \t]      { ; }
[()\-+=*/%^;] { return yytext[0];}
.          { fprintf(stderr,"Unexpected character: %c \nError at line no:%d
\n",yytext[0],yylineno); exit(EXIT_FAILURE);}


%%

int yywrap(void){return 1;}
```

2.2 Yacc Code

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
int symbols[52];
void yyerror (char *s);
int yylex();
int symbolVar(char symbol);
double getExponentiation(double base, double exponent);
void updateSymbolVal(char symbol, double val);
double getModulus(double num1,double num2);
%}

%union { double num; char id; }
%start line
%token print
%token exit_command
%token <num> number
%token <id> identifier
%left '+' '-'
%left '*' '/' '%'
%left '^'
%left '(' ')'
%type <num> line exp term primary factor statement
%type <id> assignment

%%

line       :   statement '\n'             {$$=$1;}
           |   line statement '\n'        {$$=$1;}
           ;
statement   :   assignment ';'             {;}
           |   print exp ';'              { printf("%.3f\n", $2); }
           |   exit_command ';'           { exit(EXIT_SUCCESS); }
           |   statement assignment ';'    {;}
           ;

assignment  :   identifier  '=' exp        { updateSymbolVal($1, $3); }
           |   identifier                {yyerror("Variable declare without assignment");
exit(EXIT_FAILURE);}
           |   identifier '='             {yyerror("Variable assigned without a value");
exit(EXIT_FAILURE);}
           ;

exp        :   term                     { $$ = $1 ;}
           |   exp '+' term             { $$ = $1 + $3 ;}
           |   exp '-' term             { $$ = $1 - $3 ;}
           ;
```

```
term      :   factor                { $$ = $1; }
          |   term '*' factor       { $$ = $1 * $3; }
          |   term '/' factor       { $$ = $1 / $3; }
          |   term '%' factor       { $$ = getModulus($1, $3); }
          ;

factor    :   primary               {$$ = $1;}
          |   factor '^' primary    {$$ = getExponentiation($1, $3);}
          ;

primary   :   number                { $$ = $1; }
          |   identifier            { $$ = symbolVar($1); }
          |   '(' exp ')'           { $$ = $2; }
          ;

%%
double getModulus(double num1,double num2){
   double mod = num1-(int)(num1/num2)*num2;
   return mod;
}
double getExponentiation(double base, double exponent){
   double num = base;
   for(int i = 1; i < exponent; i++){
      num = base * num;
   }
   return num;
}

int computeSymbolIndex(char token){
   int index = -1;
   if(islower(token)){
      index = token - 'a' + 26;
   }
   else if(isupper(token)){
      index = token - 'A';
   }
   return index;
}

int symbolVar(char symbol){
   int bucket = computeSymbolIndex(symbol);
   return symbols[bucket];
}

void updateSymbolVal(char symbol, double val){
   int bucket = computeSymbolIndex(symbol);
   symbols[bucket] = val;
}
```

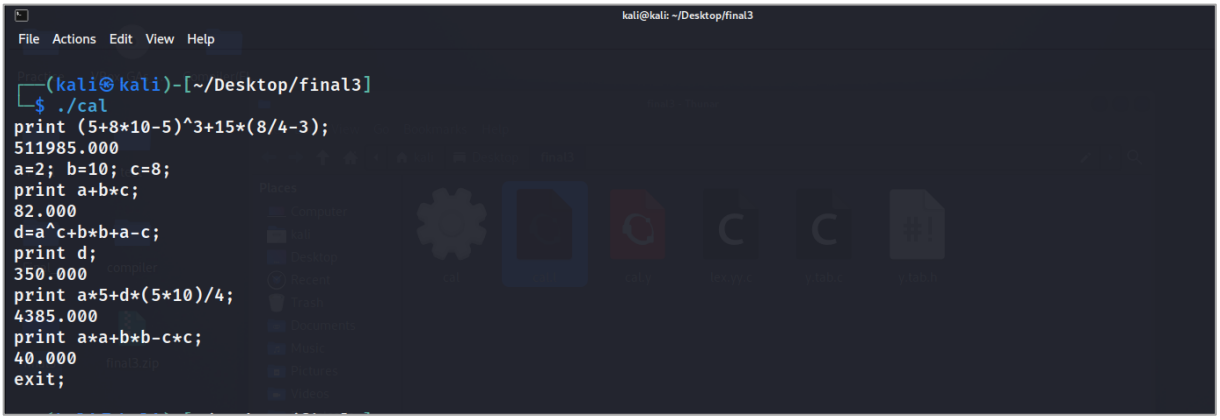
```
void yyerror(char *s){
    fprintf(stderr, "Syntax Error :%s\n",s);
}

int main(void){
    for(int i = 0; i < 52; i++){
        symbols[i] = 0;
    }
    return yyparse();
}
```

CHAPTER 03

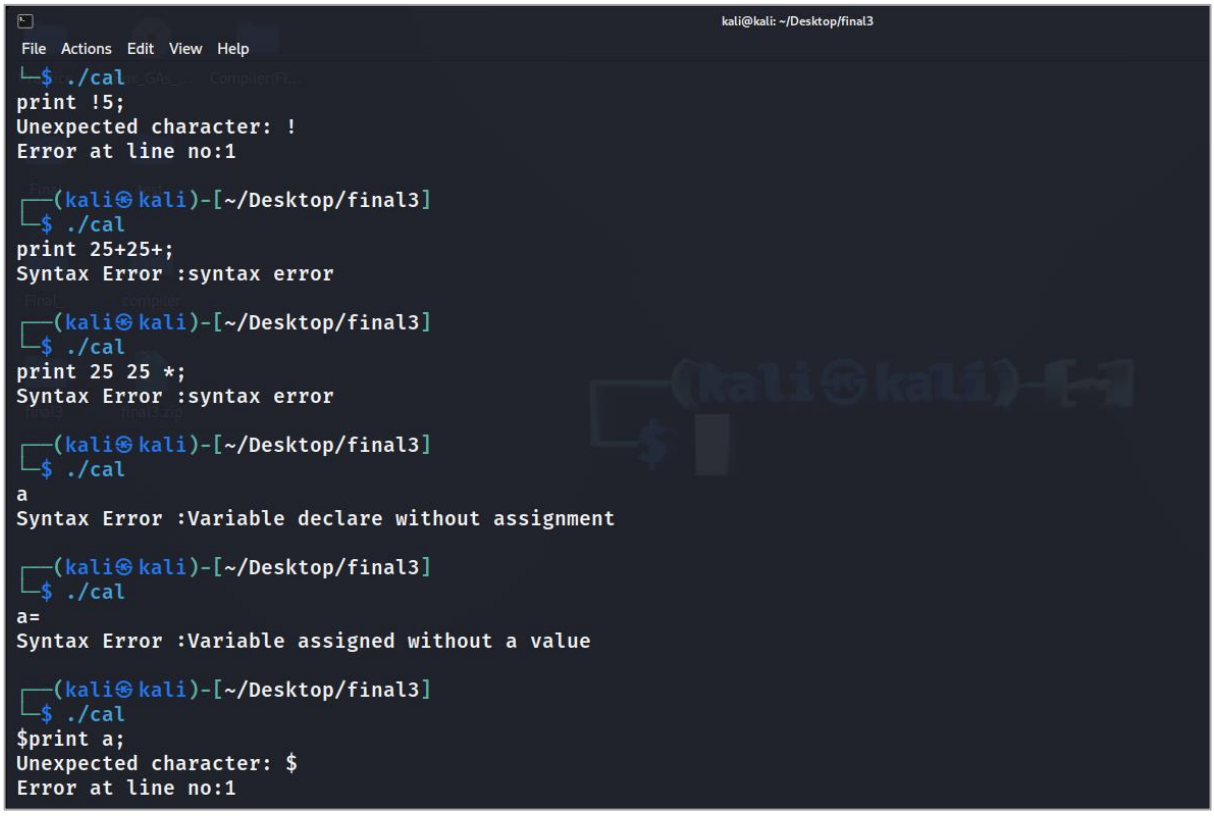
3.1 Sample Input and Output

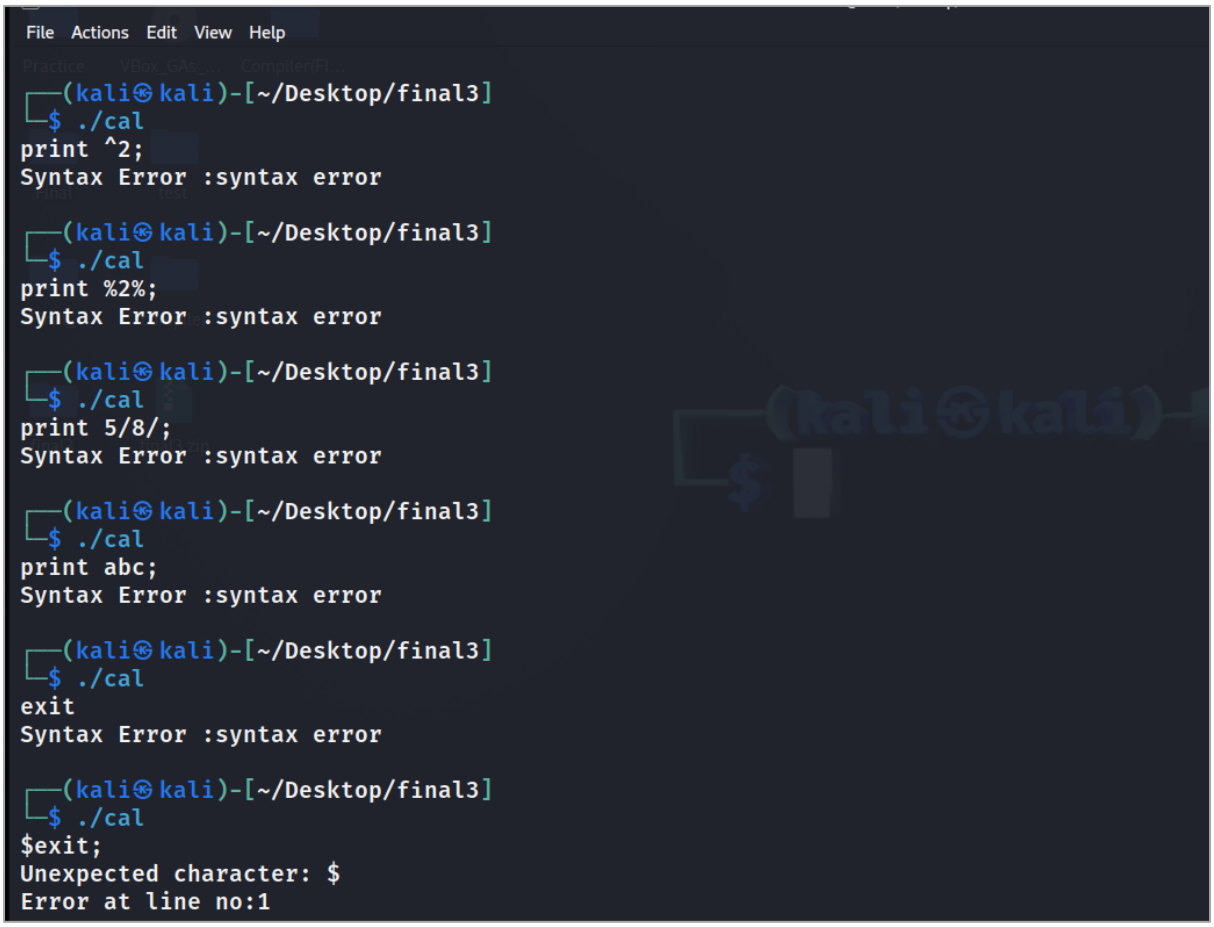
Input	Output
print 45*85+(15/3);	3830.000
print 5+56+89+74+89;	313.000
print 480-120-50-60-15;	235.000
print 8*4*78*9*63;	1415232.000
print 625/25/5/1;	5.000
print 28%3;	1.000
print 2^4^2;	256.000
print (5+8*10-5)^3+15*(8/4-3);	511985.000
a=2;b=10;c=8;	Valid Expression
print a+b*c;	82.000
d=a^c+b*b+a-c;	Valid Expression
print d;	350.000
print a*5+d*(5*10)/4;	4385.000
Print a*a+b*b-c*c	40.000
exit ;	Valid Expression
print !5;	Unexpected character: ! Error at line no 1
print 25+25+;	Syntax Error
Print 25 25 *;	Syntax Error
a	Syntax Error :Variable declare without an assignment
a=	Syntax Error :Variable assignment without a value
$print a;	Unexpected character: $ Error at line no 1
print ^2;	Syntax Error
print %2%;	Syntax Error
print 5/8/;	Syntax Error
print abc;	Syntax Error
exit	Syntax Error
$exit;	Unexpected character: $ Error at line no 1

3.2 Screenshots of compilation and running the Lex and Yacc program with valid expressions



3.3 Screenshots of the Syntax errors and invalid characters





CHAPTER 04

4.1 Limitations, Recommendations and Conclusion

Limitations

The following are some of the limitations of the syntax analyzer for the simple calculator:

- English alphabet single character variables are only allowed.
- Source code can be compiled only inside a Linux operating system.

Recommendations

The following are some recommendations for improving the syntax analyzer for the simple calculator:

- Add support for more arithmetic operations, such as unary minus.

Conclusion

We have successfully created a syntax analyzer for a simple calculator using the LEX and YACC tools in the vi editor. The syntax analyzer is able to check for the following arithmetic operations: addition, subtraction, multiplication, division, modulus, exponentiation. The syntax analyzer is well-documented and includes screenshots to demonstrate how to compile and run the program, as well as screenshots of the output with valid and invalid expressions.

The syntax analyzer has some limitations, such as the fact that it is only designed for a limited set of arithmetic operations. However, the syntax analyzer can be improved by adding support for more features, such as additional arithmetic operations.

This project has been a valuable learning experience for us, and we have gained a better understanding of how to use the LEX and YACC tools to implement lexical analyzers and parsers.

REFERENCES

GeeksforGeeks. (2019). *Introduction to YACC*. [online] Available at: https://www.geeksforgeeks.org/introduction-to-yacc/.