**Experiment No - 4**

**Aim**: Introduction to YACC and generate Calculator Program

**Date:**

# Competency and Practical Skills:

* Understanding of YACC and its role in compiler construction
* Ability to write grammar rules and YAAC programs for a given language
* Ability to develop program using YACC

# Relevant CO: CO2

# Objectives:

By the end of this experiment, the students should be able to:

* Understand the concept of YACC and its significance in compiler construction
* Write grammar rules for a given language
* Implement a calculator program using YACC

**Software/Equipment:** Windows/Linux Operating System, YACC Compiler, Text editor, Command prompt or terminal

# Theory:

YACC (Yet Another Compiler Compiler) is a tool that is used for generating parsers. It is used in combination with Lex to generate compilers and interpreters. YACC takes a set of rules and generates a parser that can recognize and process the input according to those rules.

The grammar rules that are defined using YACC are written in BNF (Backus-Naur Form) notation. These rules describe the syntax of a programming language.

INPUT FILE:

→ The YACC input file is divided into three parts. /\* definitions \*/

....

%%

/\* rules \*/

....

%%

/\* auxiliary routines \*/

....

Definition Part:

→ The definition part includes information about the tokens used in the syntax definition.

Rule Part:

→ The rules part contains grammar definition in a modified BNF form. Actions is C code in { } and can be embedded inside (Translation schemes).

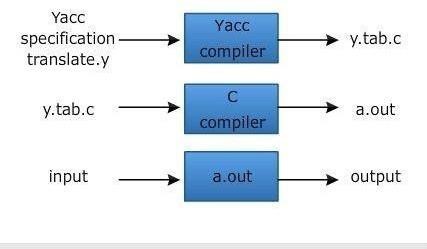
Auxiliary Routines Part:

→ The auxiliary routines part is only C code.

→ It includes function definitions for every function needed in the rules part.

→ It can also contain the main() function definition if the parser is going to be run as a program.

→ The main() function must call the function yyparse().



For Compiling YACC Program:

1. Write lex program in a file file.l and yacc in a file file.y
2. Open Terminal and Navigate to the Directory where you have saved the files.
3. type lex file.l
4. type yacc file.y
5. type cc lex.yy.c y.tab.h -ll
6. type ./a.out

The program for generating a calculator using YACC involves the following steps:

* + Defining the grammar rules for the calculator program
  + Writing the Lex code for tokenizing the input
  + Writing the YACC code for parsing the input and generating the output

# Program:

1. Create a file named "calculator.l":

%{

#include<stdio.h> #include<ctype.h>

int result;

%}

%%

[0-9]+ { yylval = atoi(yytext); return INTEGER; }

[ \t] ; /\* skip whitespace \*/ \n { return EOL; }

. { return yytext[0]; }

%%

int yywrap(void) { return 1;

}

2. Create a file named "calculator.y":

%{

#include<stdio.h>

%}

%token INTEGER EOL

%%

line: /\* empty \*/

| line exp EOL { printf("= %d\n", $2); }

;

exp: INTEGER { $$ = $1; } | exp '+' exp { $$ = $1 + $3; }

| exp '-' exp { $$ = $1 - $3; }

| exp '\*' exp { $$ = $1 \* $3; }

| exp '/' exp { $$ = $1 / $3; }

| '(' exp ')' { $$ = $2; }

;

%%

int main(void) { yyparse(); return 0;

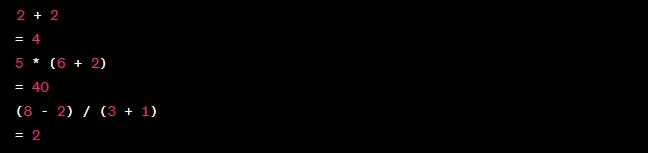
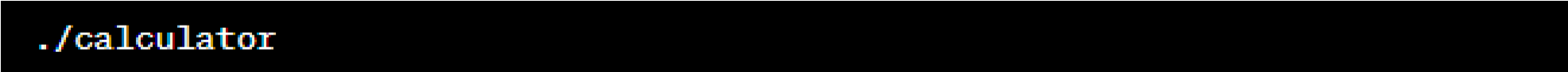
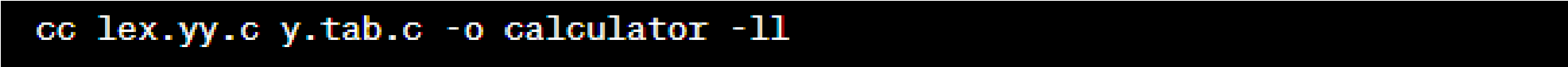
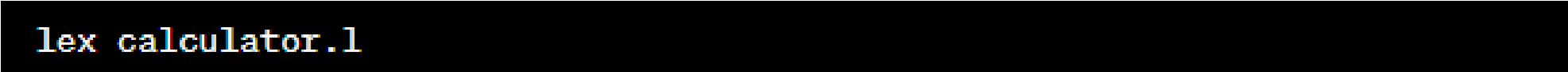
}

void yyerror(char\* s) {

fprintf(stderr, "error: %s\n", s);

}

# Observations and Conclusion:



After executing the program, we observed that the calculator program was successfully generated using YACC. It was able to perform simple arithmetic operations such as addition, subtraction, multiplication, and division. The program was also able to handle negative numbers and brackets.

# Quiz:

1. What is YACC?
2. What is the purpose of YACC?
3. What is the output of YACC?
4. What is a syntax analyzer?
5. What is the role of a lexical analyzer in YACC?

# Suggested Reference:

1. "Lex & Yacc" by John R. Levine, Tony Mason, and Doug Brown
2. "The Unix Programming Environment" by Brian W. Kernighan and Rob Pike

# References used by the students:

# Rubric wise marks obtained:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Understandi ng of YAAC**  **(2)** | | **Grammar**  **Generation**  **(2)** | | **Implementat ion (2)** | | **Testing & Debugging (2)** | | **Ethics (2)** | | **Total** |
| **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** |
| **Marks** |  |  |  |  |  |  |  |  |  |  |  |