**Practical No-3(a)**

**Aim:** Write Desk-calculator program that performs addition, subtraction, multiplication, and division operations. This calculator program also allows you to assign values to variables (each designated by a single, lowercase letter) and then use the variables in calculations. The files that contain the example **lex** and **yacc** programs are as follows:

| **File** | **Content** |
| --- | --- |
| **calc.lex** | Specifies the **lex** command specification file that defines the lexical analysis rules. |
| **calc.yacc** | Specifies the **yacc** command grammar file that defines the parsing rules, and calls the **yylex** subroutine created by the **lex** command to provide input |
|  |  |

**Theory:**

**Compiling the Example Program**

Perform the following steps, in order, to create the desk calculator example program:

1. Process the **yacc** grammar file using the **-d** optional flag (which tells the **yacc** command to create a file that defines the tokens used in addition to the C language source code):

**yacc -d calc.yacc**

1. Use the **li** command to verify that the following files were created:

|  |  |
| --- | --- |
| **y.tab.c** | The C language source file that the **yacc** command created for the parser. |
| **y.tab.h** | A header file containing define statements for the tokens used by the parser. |

1. Process the **lex** specification file: **lex calc.lex**
2. Use the **li** command to verify that the following file was created:

|  |  |
| --- | --- |
| **lex.yy.c** | The C language source file that the **lex** command created for the lexical analyzer. |

1. Compile and link the two C language source files: **cc y.tab.c lex.yy.c**
2. Use the **li** command to verify that the following files were created:

|  |  |
| --- | --- |
| **y.tab.o** | The object file for the **y.tab.c** source file |
| **lex.yy.o** | The object file for the **lex.yy.c** source file |
| **a.out** | The executable program file |

1. To then run the program directly from the **a.out** file, enter:
2. $ **a.out**

Or, to move the program to a file with a more descriptive name, as in the following example, and run it, enter:

1. $ **mv a.out calculate**
2. $ **calculate**
3. In either case, after you start the program, the cursor moves to the line below the $ (command prompt). Then enter numbers and operators in calculator fashion. When you press the Enter key, the program displays the result of the operation. After you assign a value to a variable:

m=4 <enter>

\_

the cursor moves to the next line. When you use the variable in subsequent calculations, it will have the assigned value:

m+5 <enter>

9

\_

**Program:**

1. **Calc.l**

%{

#include "y.tab.h"

#include <stdlib.h>

void yyerror(char \*);

%}

%%

[a-z] {

yylval = \*yytext - 'a';

return VARIABLE;

}

[0-9]+ {

yylval = atoi(yytext);

return INTEGER;

}

[-+()=/\*\n] { return \*yytext; }

[ \t] ; /\* skip whitespace \*/

. yyerror("Unknown character");

%%

int yywrap(void) {

return 1;

}

1. **Calc.yacc**

%{

#include <stdio.h>

void yyerror(char \*);

int yylex(void);

int sym[26];

%}

%token INTEGER VARIABLE

%left '+' '-'

%left '\*' '/'

%%

program:

program statement '\n'

| /\* NULL \*/

;

statement:

expression { printf("%d\n", $1); }

| VARIABLE '=' expression { sym[$1] = $3; }

;

expression:

INTEGER

| VARIABLE { $$ = sym[$1]; }

| expression '+' expression { $$ = $1 + $3; }

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

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

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

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

;

%%

void yyerror(char \*s)

{

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

}

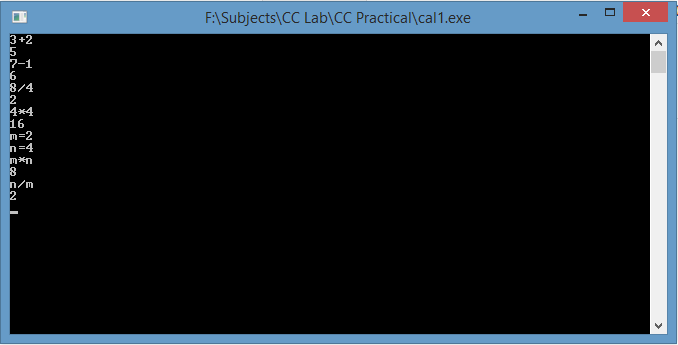
int main(void)

{

yyparse();

}

**Output:**

****

**Conclusion:** A simple calculator that can perform basic operations like addition, subtraction, multiplication and division has been programmed using FLEX and YACC

**Viva questions:-**

1. What is yacc? Find its characteristics?
2. How yacc works?
3. What is format of typical file in yacc?
4. The YACC takes C code as input and outputs\_\_\_\_\_\_\_\_\_  
   a) Top down parsers *b) Bottom up parsers*  
   c) Machine code d) None of the mentioned
5. The original YACC as written in \_\_\_\_\_\_\_\_\_\_ language  
   a) R programming language b) C programming language  
   *c) B programming language* d) None of the mentioned
6. Which of the following version of Unix came up with YACC first?  
   *a) V3* b) V5 c) CB UNIX d) Unix-RT