SSC LAB ASSIGNMENT NO.8
CALCULATOR

NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1

Title: Parser for Arithmetic Grammar using YACC.

Aim: Write a program using LEX and YACC to create Parser for Arithmetic Grammar ---

-Design Calculator.

Objective:

1. To understand Yacc Tool.
2. To study how to use Yacc tool for implementing Parser.
3. To understand the compilation and execution of *. y file.

Theory: Write in brief for following:

1. **Introduction to Yacc –**
   - YACC stands for Yet Another Compiler Compiler.
   - YACC provides a tool to produce a parser for a given grammar.
   - YACC is a program designed to compile a LALR (1) grammar.
   - It is used to produce the source code of the syntactic analyzer of the language produced by LALR (1) grammar.
   - The input of YACC is the rule or grammar and the output is a C program

Yacc specification describes a CFG, that can be used to generate a parser.
Elements of a CFG:
1. Terminals: tokens and literal characters,
2. Variables (nonterminals): syntactical elements,
3. Production rules, and
4. Start rule. Format of a production rule: symbol: definition {action} ;

2. Study of *. y file(specification of y file)

Input: A CFG- file.y

Output: A parser y.tab.c (yacc)

- The output file "file.output" contains the parsing tables.
- The file "file.tab.h" contains declarations.
- The parser called the yyparse ().
- Parser expects to use a function called yylex () to get tokens.

3. Description of Each Section of *. y file with example.

YACC input file is divided into three parts.

```
/* definitions */
 ....

%%
/* rules */
....
%%

/* auxiliary routines */
....
```

1) Input File: **Definition Part:**

    - The **definition** part includes information about the tokens used in the syntax definition:

      ```
      %token NUMBER
      %token ID
      ```

    - Yacc automatically assigns numbers for tokens, but it can be overridden by

      ```
      %token NUMBER 621
      ```

    - Yacc also recognizes single characters as tokens. Therefore, assigned token numbers should not overlap ASCII codes.
    - The definition part can include C code external to the definition of the parser and variable declarations, within %{ and %} in the first column.
    - It can also include the specification of the starting symbol in the grammar:

      ```
      %start nonterminal
      ```

2) Input File: **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).

3) Input File: **Auxiliary Routines Part:**

- The auxiliary routines part is only C code.
- It includes function definitions for every function needed in 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().

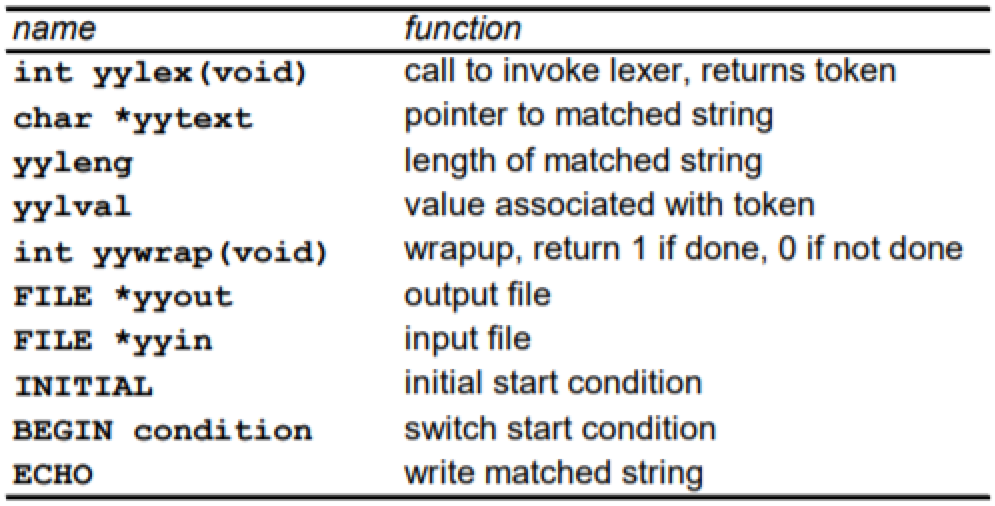
EXAMPLE : INFIX TO POSTFIX EXPRESSION

```
%{
/*** Auxiliary declarations section ***/

#include<stdio.h>
#include<stdlib.h>

/* Custom function to print an operator*/
void print_operator(char op);

/* Variable to keep track of the position of the number
in the input */
int pos=0;

%}

 /*** YACC Declarations section ***/
%token DIGIT
%left '+'
%left '*'
%%

/*** Rules Section ***/
start : expr '\n'          {exit(1);}
      ;

expr: expr '+' expr     {print_operator('+');}
    | expr '*' expr     {print_operator('*');}
    | '(' expr ')'
    | DIGIT             {printf("NUM%d ",pos);}
    ;

%%


/*** Auxiliary functions section ***/

void print_operator(char c){
    switch(c){
        case '+'  : printf("PLUS ");
                    break;
        case '*'  : printf("MUL ");
                    break;
    }
    return;
}

yyerror(char const *s)
{
    printf("yyerror %s",s);
}

yylex(){
    char c;
    c = getchar();
    if(isdigit(c)){
        pos++;
        return DIGIT;
    }
    else if(c == ' '){
        yylex();         /*This is to ignore whitespaces
in the input*/
    }
    else {
        return c;
    }
}

main()
{
     yyparse();
     return 1;
}
```

Sample Input/Output:

```
I: 2+2

O: NUM1 NUM2 PLUS
```

4. **Description of standard inbuilt variables and functions.**



name	*function*
`int yylex(void)`	call to invoke lexer, returns token
`char *yytext`	pointer to matched string
`yyleng`	length of matched string
`yylval`	value associated with token
`int yywrap(void)`	wrapup, return 1 if done, 0 if not done
`FILE *yyout`	output file
`FILE *yyin`	input file
`INITIAL`	initial start condition
`BEGIN condition`	switch start condition
`ECHO`	write matched string

5. **Compilation and Execution Process-**

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

Input: Source specification (*. y) file for arithmetic expression statements.

Output: Result of Arithmetic Expression

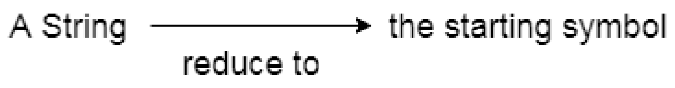
FAQs:

1. **Differentiate between top down and bottom-up parsers.**

S.No	Top Down Parsing	Bottom Up Parsing
1.	It is a parsing strategy that first looks at the highest level of the parse tree and works down the parse tree by using the rules of grammar.	It is a parsing strategy that first looks at the lowest level of the parse tree and works up the parse tree by using the rules of grammar.
2.	Top-down parsing attempts to find the left most derivations for an input string.	Bottom-up parsing can be defined as an attempts to reduce the input string to start symbol of a grammar.
3.	In this parsing technique we start parsing from top (start symbol of parse tree) to down (the leaf node of parse tree) in top-down manner.	In this parsing technique we start parsing from bottom (leaf node of parse tree) to up (the start symbol of parse tree) in bottom-up manner.
4.	This parsing technique uses Left Most Derivation.	This parsing technique uses Right Most Derivation.
5.	It's main decision is to select what production rule to use in order to construct the string.	It's main decision is to select when to use a production rule to reduce the string to get the starting symbol.

2. Explain working of shift-reduce parser.

- Shift reduce parsing is a process of reducing a string to the start symbol of a grammar.
- Shift reduce parsing uses a stack to hold the grammar and an input tape to hold the string.



- Sift reduce parsing performs the two actions: shift and reduce. That's why it is known as shift reduces parsing.
- At the shift action, the current symbol in the input string is pushed to a stack.
- At each reduction, the symbols will replaced by the non-terminals. The symbol is the right side of the production and non-terminal is the left side of the production.

3. Explain how communication between LEX and YACC is carried out.

- lex and yacc often work well together for developing compilers.
- As noted, a program uses the lex-generated scanner by repeatedly calling the function **yylex()**. This name is convenient because a yacc-generated parser calls its lexical analyzer with this name.
- To use lex to create the lexical analyzer for a compiler, end each lex action with the statement return token, where token is a defined term with an integer value.
- The integer value of the token returned indicates to the parser what the lexical analyzer has found. The parser, called **yyparse()** by yacc, then resumes control and makes another call to the lexical analyzer to get another token.
- In a compiler, the different values of the token indicate what, if any, reserved word of the language has been found or whether an identifier, constant, arithmetic operator, or relational operator has been found. In the latter cases, the analyzer must also specify the exact value of the token: what the identifier is, whether the constant is, say, 9 or 888, whether the operator is + or *, and whether the relational operator is = or >.

4. How YACC resolves ambiguities within given grammar.

The parser generator tool YAAC resolves conflicts due to ambiguous grammars as follows,

- Shift/reduce conflict in the parsing table is resolved by giving priority to shift move over reduce move. If the string is accepted for shift move, then reduce move is removed, otherwise shift move is removed.
- Reduce/reduce conflict in the parsing table is resolved by giving priority to first reduce move over second reduce move. If the string is accepted for first reduce move, then second reduce move is removed, otherwise first reduce move is removed.

PROGRAM :

```
/*
 SSC LAB ASSIGNMENT NO.8
CALCULATOR
NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1
 */
```

Lex file :

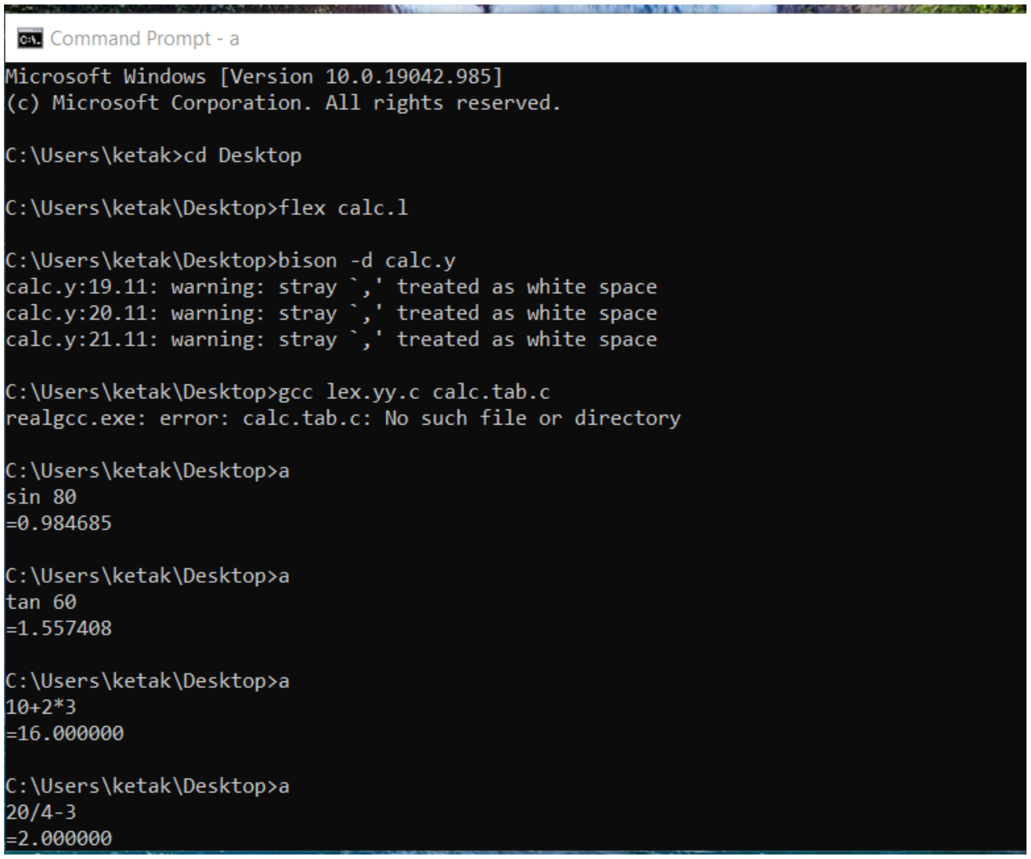
```
%{
    #include<stdlib.h>
    #include "calc.tab.h"
    void yyerror(char *error);
%}
```

```
%%
[0-9]+  {yylval.intval=atoi(yytext);
        return NUMBER; }
"sin"   {return SIN;}
"cos"   {return COS;}
"tan"   {return TAN;}
[a-z]+  {strcpy(yylval.fchar,yytext);
        return NAME;}

[\t ];
\n      return 0;

.   {return yytext[0];}

%%

yywrap()
{
    return 1;
}
```

Yacc file:

```
%{
    #include<stdlib.h>
    #include<math.h>
    #include <stdio.h>
%}

%union{
    char fchar;
    double fval;
    int intval;
};

%token SIN
%token COS
%token TAN
%token <fchar>NAME
%token <intval>NUMBER
%type <fval>exp
%left '+' ,'-'
%left '*' ,'/'
%left '^' ,' '

%%

stmt: NAME'='exp { printf("=%f\t\n",$3);}
    | exp { printf("=%f\n",$1);}
    ;
exp : exp'+'exp { $$ = $1 + $3;}
    | exp'-'exp { $$ = $1 - $3;}
    | exp'*'exp { $$ = $1 * $3;}
    | SIN' 'exp { $$ = sin($3*3.14/180);}
    | COS' 'exp { $$ = cos($3*3.14/180);}
    | TAN' 'exp { $$ =tan($3*(22/7)/180);}

    | exp'/'exp {
                if($3==0)
                {
                    printf("\nDivide by zero.");
                }
                else
                {
                    $$ = $1 / $3;
                }
        }
    | NUMBER { $$ = $1;}
    ;
%%

void yyerror(char *error)
{
    printf("%s",error);
}
main()
{
    yyparse();
    getch();
}
```

OUTPUT SS :



```
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ketak>cd Desktop

C:\Users\ketak\Desktop>flex calc.l

C:\Users\ketak\Desktop>bison -d calc.y
calc.y:19.11: warning: stray `,' treated as white space
calc.y:20.11: warning: stray `,' treated as white space
calc.y:21.11: warning: stray `,' treated as white space

C:\Users\ketak\Desktop>gcc lex.yy.c calc.tab.c
realgcc.exe: error: calc.tab.c: No such file or directory

C:\Users\ketak\Desktop>a
sin 80
=0.984685

C:\Users\ketak\Desktop>a
tan 60
=1.557408

C:\Users\ketak\Desktop>a
10+2*3
=16.000000

C:\Users\ketak\Desktop>a
20/4-3
=2.000000
```