Pandit Deendayal Energy University, Gandhinagar

School of Technology

Department of Computer Science & Engineering

System Software & Compiler Design Lab (20CP302P)

Name: **Sutariya Smit Dharmendrabhai**

Enrolment No: **21BCP142**

Semester: **V**

Division: **3 (G5)**

Branch: **Computer Science Engineering**

Practical: 7

Aim:

a. Write a YACC program for desktop calculator with ambiguous grammar (evaluate arithmetic expression involving operators: +, -, *, / and ^).

Code:

7a.l

```
%{
#include "7a.tab.h"
extern int yylex();
extern void yyerror(const char* msg);
%}

%%

[0-9]+      { yylval = atoi(yytext); return NUMBER; }
"+"         { return ADD; }
"-"         { return SUBTRACT; }
"*"         { return MULTIPLY; }
"/"         { return DIVIDE; }
"^"         { return POWER; }
"("         { return LPAREN; }
```

```
")"         { return RPAREN; }
\n          { return EOL; }
[ \t]       ; // Skip whitespace
.           { yyerror("Invalid character"); }

%%

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

7a.y

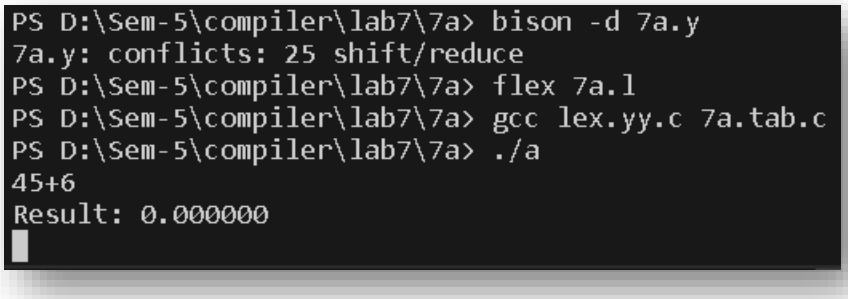
```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int yylex();
void yyerror(const char* msg);
```

```
%}

%token NUMBER
%token ADD SUBTRACT MULTIPLY DIVIDE POWER LPAREN RPAREN
EOL

%%

calculation:
    expression EOL   { printf("Result: %f\n", $1); }
    | EOL            { /* Ignore empty lines */ }
    | calculation EOL { /* Allow multiple calculations */ }
    ;

expression:
    NUMBER           { $$ = $1; }
    | expression ADD expression     { $$ = $1 + $3; }
    | expression SUBTRACT expression { $$ = $1 - $3; }
    | expression MULTIPLY expression { $$ = $1 * $3; }
    | expression DIVIDE expression   {
```

```
        if ($3 == 0) {
            yyerror("Division by zero");
            $$ = 0;  // Handle division by zero
        } else {
            $$ = $1 / $3;
        }
    }
    | expression POWER expression     { $$ = pow($1, $3); }
    | LPAREN expression RPAREN      { $$ = $2; }
    ;

%%

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

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

}

Output:


PS D:\Sem-5\compiler\lab7\7a> bison -d 7a.y
7a.y: conflicts: 25 shift/reduce
PS D:\Sem-5\compiler\lab7\7a> flex 7a.l
PS D:\Sem-5\compiler\lab7\7a> gcc lex.yy.c 7a.tab.c
PS D:\Sem-5\compiler\lab7\7a> ./a
45+6
Result: 0.000000

Aim:

b. Write a YACC program for desktop calculator with ambiguous grammar and additional information.

Code:

7b.y

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

%token NAME num
%left '+' '-'
%left '*' '/'
%right '^'
%nonassoc UMINUS

%%
s: NAME '=' Ex
   | Ex { printf("= %d\n", $1); }
   ;
Ex: Ex '+' Ex {$$ = $1 + $3;}
```

```
    | Ex '-' Ex {$$ = $1 - $3;}
    | Ex '*' Ex {$$ = $1 * $3;}
    | Ex '/' Ex {if($3 == 0)


          yyerror("divide by zero");
        else
          $$ = $1 / $3;
       }
    | Ex '^' Ex {$$ = pow($1,$3);}
    | '-' Ex %prec UMINUS {$$ = -$2;}
    | '(' Ex ')' {$$ = $2;}
    | num {$$ = $1;}
    ;
%%


int main() {
    yyparse();
    return 0;
}
```

7b.l

```
%{
#include "7b.tab.h"
%}

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

[ \t] ; /* Ignore whitespace */

\n return 0; /* Logical EOF */

. return yytext[0];
%%
int yywrap() {
   return 1;
}
void yyerror(char *s) {
   printf("error");
}
```

Output:

```
PS D:\Sem-5\compiler> cd d:\Sem-5\compiler\lab7\7b
PS D:\Sem-5\compiler\lab7\7b> bison -d 7b.y
PS D:\Sem-5\compiler\lab7\7b> flex 7b.l
PS D:\Sem-5\compiler\lab7\7b> gcc lex.yy.c 7b.tab.c
7b.tab.c: In function 'yyparse':
7b.tab.c:599:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
 # define YYLEX yylex ()
                ^
7b.tab.c:1244:16: note: in expansion of macro 'YYLEX'
       yychar = YYLEX;
                ^~~~~
7b.y:21:13: warning: implicit declaration of function 'yyerror' [-Wimplicit-function-declaration]
             yyerror("divide by zero");
             ^~~~~~~
PS D:\Sem-5\compiler\lab7\7b> ./a
5+5
= 10
PS D:\Sem-5\compiler\lab7\7b> ./a
4++5
error
PS D:\Sem-5\compiler\lab7\7b>
```

Aim:

c. Design, develop and implement a YACC program to demonstrate Shift Reduce Parsing technique for the grammar rules:

E→ E + T | T

T→ T * F | F

F → P ↑ F | P

P → (E) | id

And parse the sentence: id + id * id.

Code:

7c.l

```
%{
#include "7c.tab.h"
%}

%%
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
[ \t] ; /* Ignore whitespace */
\n return 0; /* Logical EOF */
. return yytext[0];
%%
```

```
int yywrap() {
   return 1;
}
void yyerror(char *s) {
   printf("Syntax error\n");
}
```

7c.y

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

%token NUMBER
%right '^'
%%
statement: E { printf("Result: %d\n", $1); }
        | statement E { printf("Result: %d\n", $2); }
        ;
```

```
E : E '+' T { $$ = $1 + $3; }
  | E '-' T { $$ = $1 - $3; }
  | T { $$ = $1; }
  ;

T : T '*' F { $$ = $1 * $3; }
  | T '/' F { if ($3 == 0) yyerror("division by zero"); else $$ = $1 / $3; }
  | F { $$ = $1; }
  ;

F : P '^' F { $$ = pow($1, $3); }
  | P { $$ = $1; }
  ;

P : '(' E ')' { $$ = $2; }
  | NUMBER { $$ = $1; }
  ;
%%
```

```
int main() {
    yyparse();
    return 0;
}
```

Output: