## 9. CALCULATOR

**AIM: -**

To implement a calculator using Lex and Yacc.

**PROGRAM**

**LEX**

```
DIGIT [0-9]+\.?|[0-9]*\.[0-9]+
%option noyywrap
%%
[ ]
{DIGIT} { yylval=atof(yytext); return NUM;}
\n|. {return yytext[0];}
```

**YACC PROGRAM**

```
%{
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
#define YYSTYPE double
%}
%token NUM
%left '+''-'
%left '*''/'
%right UMINUS
%%
S : S E '\n' { printf("Answer: %g \nEnter:\n", $2); }
| S '\n'
|
| error '\n' { yyerror("Error: Enter once more…\n" );yyerrok; }
;
E : E '+' E { $$ = $1 + $3; }
| E'-'E { $$=$1-$3; }
| E'*'E { $$=$1*$3; }
| E'/'E { $$=$1/$3; }
| '('E')' { $$=$2; }
| '-'E %prec UMINUS { $$= -$2; }
| NUM

;

%%


#include "lex.yy.c"


int main()

{

printf("Enter the expression: ");

yyparse();

}
yyerror (char * s)

{

printf ("% s \n", s);

exit (1);

}
```

**OUTPUT**

```
[student@localhost ]$ lex calculator.l
[student@localhost ]$ yacc -d calculator.y
[student@localhost ]$ gcc -o calculator y.tab.c
[student@localhost ]$ ./calculator
Enter the expression: 10/2-1
Answer: 4
```

# 10. CONVERSION OF E NFA TO NFA

**AIM:**

To convert NFA with ε transition to NFA without ε transition

**PROGRAM**

```
  #include <stdio.h>
  #include <string.h>
  #include <stdlib.h>
  #define STATES 256
  #define SYMBOLS 3  // 0, 1, and epsilon (at index 2)
  int N_symbols = 2;      // Number of input symbols (excluding epsilon)
  int NFA_states = 0;
  char *NFAtab[STATES][SYMBOLS];          // Original NFA (with ε)
  char *NewNFAtab[STATES][SYMBOLS];       // New NFA (without ε)
  /* Merge strings without duplicates, in sorted order */
  void string_merge(char *s, const char *t) {
  char temp[STATES], *r = temp;
  const char *p = s;
  while (*p && *t) {
   if (*p == *t) {
   *r++ = *p++; t++;
   } else if (*p < *t) {
   *r++ = *p++;
   } else {
   *r++ = *t++;
    }
   }
   strcpy(r, (*p) ? p : t);
   strcpy(s, temp);
  }
  /* Compute epsilon-closure of a single state */
  void epsilon_closure(char *result, int state) {
   char stack[STATES], visited[STATES] = {0};
   int top = 0;
   result[0] = state + '0';
   result[1] = '\0';
  stack[top++] = state;
  visited[state] = 1;
  while (top > 0) {
  int s = stack[--top];
  char *eps_moves = NFAtab[s][2];
  if (!eps_moves) continue;
 for (int i = 0; i < strlen(eps_moves); i++) {
 int next = eps_moves[i] - '0';
if (!visited[next]) {
visited[next] = 1;
stack[top++] = next;
char temp[2] = {eps_moves[i], '\0'};
 string_merge(result, temp);
}
```

```
}
}
}
/* Compute epsilon-closure of a set of states */
void epsilon_closure_set(char *result, const char *states) {
 result[0] = '\0';
 for (int i = 0; i < strlen(states); i++) {
char temp[STATES];
epsilon_closure(temp, states[i] - '0');
 string_merge(result, temp);
}}
/* Get next state set for input symbol, starting from a state set */
void get_next_states(char *result, const char *states, int symbol) {
 char temp[STATES] = "";
 for (int i = 0; i < strlen(states); i++) {
char *move = NFAtab[states[i] - '0'][symbol];
if (move) {
string_merge(temp, move);
}}
 // Take ε-closure of the result
epsilon_closure_set(result, temp);
}
/* Remove epsilon transitions and build new NFA */
void remove_epsilon_transitions() {
for (int state = 0; state < NFA_states; state++) {
char closure[STATES];
 epsilon_closure(closure, state);

for (int symbol = 0; symbol < N_symbols; symbol++) {
char next[STATES];
get_next_states(next, closure, symbol);
if (strlen(next) > 0) {
NewNFAtab[state][symbol] = strdup(next);
}
}
}
}

/* Print NFA table */
void print_NFA(char *table[STATES][SYMBOLS], int states, int symbols) {
 printf("STATE TRANSITION TABLE (symbols: 0, 1)\n");
printf("    |");
for (int i = 0; i < symbols; i++) {
 printf(" %d ", i);
}
printf("\n----+------------------\n");
for (int i = 0; i < states; i++) {
 printf(" %c  |", '0' + i);
for (int j = 0; j < symbols; j++) {
 if (table[i][j])
printf(" %s", table[i][j]);
 else
printf(" - ");
```

```
}
printf("\n");
}
}

/* Initialize an NFA with epsilon transitions */
void init_NFA_with_epsilon() {
   /*
 Example NFA:
 State 0: on ε → 1
 State 1: on 0 → 1, on 1 → 2
 State 2: on 1 → 3
 */

 NFAtab[0][2] = "1";    // ε-transition from 0 to 1
 NFAtab[1][0] = "1";    // 0 → 1
 NFAtab[1][1] = "2";    // 1 → 2
 NFAtab[2][1] = "3";    // 1 → 3
 NFA_states = 4;
}

int main() {
   init_NFA_with_epsilon();
   printf("Original NFA with ε-transitions:\n");
   print_NFA(NFAtab, NFA_states, 3);
   remove_epsilon_transitions();
   printf("\nNew NFA without ε-transitions:\n");
   print_NFA(NewNFAtab, NFA_states, N_symbols); // print only for symbols 0 and 1

   return 0;
}
```

## Input & Output:

```
student@cse-hwl-030:~$ cc nfatonfa.c
student@cse-hwl-030:~ ./a.out
Original NFA with ε-transitions:
STATE TRANSITION TABLE (symbols: 0, 1)
   | 0  1  2
----+--------------------
 0  | -  -  1
 1  | 1 2 -
 2  | -  3 -
 3  | -  -  -

New NFA without ε-transitions:
STATE TRANSITION TABLE (symbols: 0, 1)
   | 0  1
----+--------------------
 0  | 1 2
 1  | 1 2
 2  | -  3
 3  | -  -
```