Symbiosis Institute of Technology, Pune

Faculty of Engineering

CSE- Academic Year 2025-26

Compiler Construction Lab Batch 2022-26

Lab Assignment No: - 11	
Name of Student	Kshitij Gurbuxani
PRN No.	22070122097
Batch	22-26
Class	CS-B
Academic Year & Semester	25-26 7th
Date of Performance	15-10-2025
Title of Assignment	LEX program to implement a simple calculator.
Practice Questions	1. LEX program to implement a simple calculator. 2. LEX program for only checking digits and operators.

Source Code	1.Simple Calculator
	
	%{
	#include <stdio.h>
	#include <stdlib.h>
	#include <ctype.h>
	#include <string.h>
	
	#define MAXSTACK 100
	
	int stack[MAXSTACK];
	int top = -1;
	
	void push(int val) {
	  if (top == MAXSTACK - 1) {
	  printf("Error: Stack overflow\n");
	  exit(1);
	  }
	  stack[++top] = val;
	}

```
int pop() {
   if (top == -1) {
      printf("Error: Stack underflow\n");
      exit(1);
   }
   return stack[top--];
}

int eval(char *expr);
int precedence(char op);
int applyOp(int a, int b, char op);
int compute(const char *expr);
%}

%%
[0-9]+      {
               push(atoi(yytext));
            }

[+\-*/()]   {
               push(yytext[0]);
            }

[\n]        {

            }

.           {

            }
%%

int precedence(char op) {
   switch(op) {
      case '+':
      case '-': return 1;
      case '*':
      case '/': return 2;
      default: return 0;
   }
}

int applyOp(int a, int b, char op) {
   switch(op) {
      case '+': return a + b;
      case '-': return a - b;
      case '*': return a * b;
      case '/':
         if (b == 0) {
            printf("Error: Division by zero\n");
            exit(1);
```

```
            }
            return a / b;
    }
    return 0;
}

int compute(const char *expr) {
    int values[MAXSTACK], valTop = -1;
    char ops[MAXSTACK];
    int opTop = -1;

    for (int i = 0; i < strlen(expr); i++) {
        if (expr[i] == ' ')
            continue;


        if (isdigit(expr[i])) {
            int val = 0;
            while (i < strlen(expr) && isdigit(expr[i])) {
                val = (val * 10) + (expr[i] - '0');
                i++;
            }
            i--;
            values[++valTop] = val;
        }


        else if (expr[i] == '(') {
            ops[++opTop] = expr[i];
        }

        else if (expr[i] == ')') {
            while (opTop != -1 && ops[opTop] != '(') {
                int b = values[valTop--];
                int a = values[valTop--];
                char op = ops[opTop--];
                values[++valTop] = applyOp(a, b, op);
            }
            opTop--;
        }


        else {
            while (opTop != -1 && precedence(ops[opTop]) >=
precedence(expr[i])) {
                int b = values[valTop--];
                int a = values[valTop--];
                char op = ops[opTop--];
                values[++valTop] = applyOp(a, b, op);
            }
            ops[++opTop] = expr[i];
```

```
        }
    }

    while (opTop != -1) {
        int b = values[valTop--];
        int a = values[valTop--];
        char op = ops[opTop--];
        values[++valTop] = applyOp(a, b, op);
    }

    return values[valTop];
}

int main() {
    char expr[256];
    printf("Simple Calculator\n");
    printf("Enter expression (or type 'exit' to quit):\n");

    while (1) {
        printf("> ");
        if (!fgets(expr, sizeof(expr), stdin))
            break;

        if (strncmp(expr, "exit", 4) == 0)
            break;

        expr[strcspn(expr, "\n")] = 0;

        int result = compute(expr);
        printf("Result = %d\n", result);
    }

    return 0;
}
int yywrap(){
return 1;
}
```

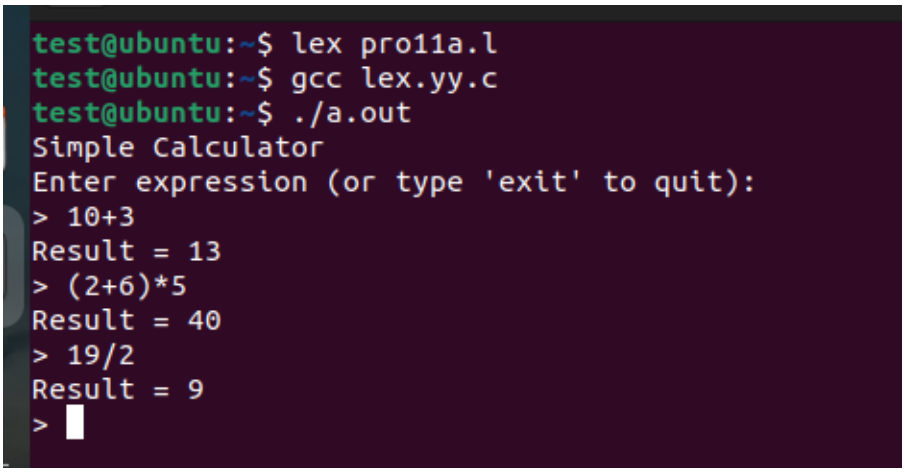
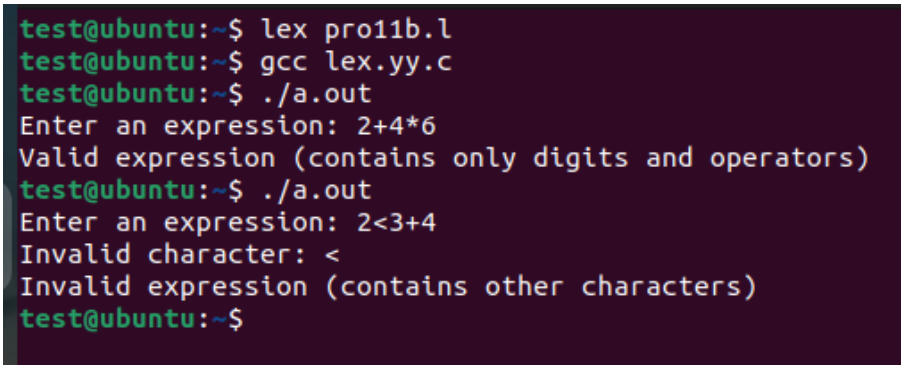
2.Check valid expression with only operators and digits

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

int valid = 1;
%}

%%
[0-9]+          { /* digits are valid */ }
[+\-*/]         { /* operators are valid */ }
[ \t]           {  }
\n              { return 0; }
```

	. { valid = 0; printf("Invalid character: %s\n", yytext); }
	%%
	
	int main() {
	  printf("Enter an expression: ");
	  yylex();
	
	  if (valid)
	  printf("Valid expression (contains only digits and operators)\n");
	  else
	  printf("Invalid expression (contains other characters)\n");
	
	  return 0;
	}
	
	int yywrap() { return 1; }

Output Screenshot	1.  2. 
Post lab questions	1. LEX based calculator for three operands and two operators based expression evaluation.

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

int numbers[3];
char ops[2];
int nCount = 0, oCount = 0;

int compute(int a, int b, char op) {
   switch (op) {
      case '+': return a + b;
      case '-': return a - b;
      case '*': return a * b;
      case '/':
         if (b == 0) {
            printf("Error: Division by zero!\n");
            exit(1);
         }
         return a / b;
   }
   return 0;
}
%}

%%
[0-9]+      {
            if (nCount < 3)
               numbers[nCount++] = atoi(yytext);
            else {
               printf("Error: Too many operands!\n");
               exit(1);
            }
         }

[+\-*/]     {
            if (oCount < 2)
               ops[oCount++] = yytext[0];
            else {
               printf("Error: Too many operators!\n");
               exit(1);
            }
         }

[ \t]
\n          { return 0; }
.           {
            printf("Invalid character: %s\n", yytext);
            exit(1);
         }
%%
```

```
int main() {
    printf("Enter expression with 3 operands and 2 operators: ");
    yylex();

    if (nCount != 3 || oCount != 2) {
        printf("Error: Expression must contain exactly 3 operands and 2
operators.\n");
        return 1;
    }

    int result;

    if (ops[1] == '*' || ops[1] == '/') {
        int temp = compute(numbers[1], numbers[2], ops[1]);
        result = compute(numbers[0], temp, ops[0]);
    } else if (ops[0] == '*' || ops[0] == '/') {
        int temp = compute(numbers[0], numbers[1], ops[0]);
        result = compute(temp, numbers[2], ops[1]);
    } else {
        int temp = compute(numbers[0], numbers[1], ops[0]);
        result = compute(temp, numbers[2], ops[1]);
    }

    printf("Result = %d\n", result);
    return 0;
}

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

```
test@ubuntu:~$ lex pro11c.l
test@ubuntu:~$ gcc lex.yy.c
test@ubuntu:~$ ./a.out
Enter expression with 3 operands and 2 operators: 8/2+6
Result = 10
test@ubuntu:~$ ./a.out
Enter expression with 3 operands and 2 operators: 3*2-6
Result = 0
test@ubuntu:~$ ./a.out
Enter expression with 3 operands and 2 operators: 8-5*9
Result = -37
test@ubuntu:~$
```

Conclusion	We learnt to implement LEX to make a simple calculator , check valid expressions and perform operations with 3 operands and 2 operators