31. Implement a C program to perform symbol table operations

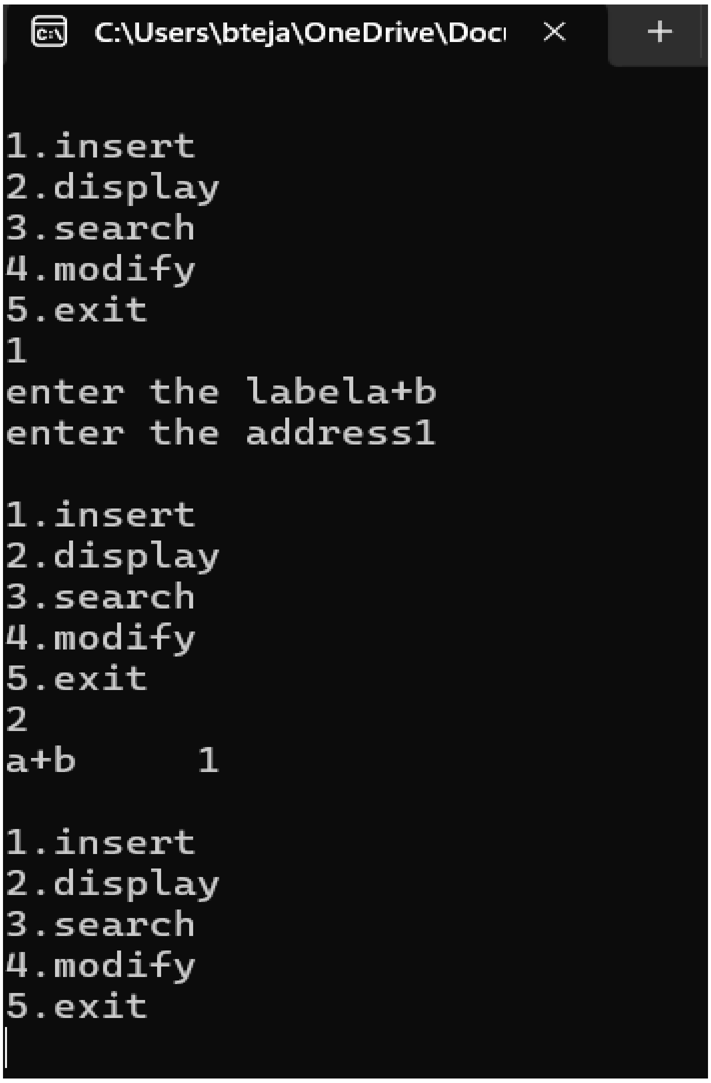
PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int cnt=0;
struct symtab
{
	char label[20];
	int addr;
}
sy[50];
void insert();
int search(char *);
void display();
void modify();
int main()
{
int ch,val;
char lab[10];
do
{
	printf("\n1.insert\n2.display\n3.search\n4.modify\n5.exit\n");
	scanf("%d",&ch);
	switch(ch)
	{
		case 1:
			insert();
			 break;
			case 2:
```

```
                display();
                break;
        case 3:
printf("enter the label");
            scanf("%s",lab);
            val=search(lab);
            if(val==1)
            printf("label is found");
            else
            printf("label is not found");
        break;
    case 4:
            modify();
        break;
    case 5:
            exit(0);
            break;
        }
    }while(ch<5);
}
void insert()
{
int val;
    char lab[10];
    int symbol;
    printf("enter the label");
    scanf("%s",lab);
    val=search(lab);
    if(val==1)
    printf("duplicate symbol");
```

```
        else
        {
                strcpy(sy[cnt].label,lab);
                printf("enter the address");
                scanf("%d",&sy[cnt].addr);
                cnt++;
        }
}
int search(char *s)
{
        int flag=0,i; for(i=0;i<cnt;i++)
        {
                if(strcmp(sy[i].label,s)==0)
                flag=1;
        }
return flag;
}
void modify()
{
        int val,ad,i;
        char lab[10];
        printf("enter the labe:");
        scanf("%s",lab);
        val=search(lab);
        if(val==0)
        printf("no such symbol");
        else
        {
                printf("label is found \n");
                printf("enter the address");
                scanf("%d",&ad);
                for(i=0;i<cnt;i++)
                {
                        if(strcmp(sy[i].label,lab)==0)
                        sy[i].addr=ad;
                }
        }
}
void display()
{
        int i;
        for(i=0;i<cnt;i++)
        printf("%s\t%d\n",sy[i].label,sy[i].addr);
}
```

OUTPUT:



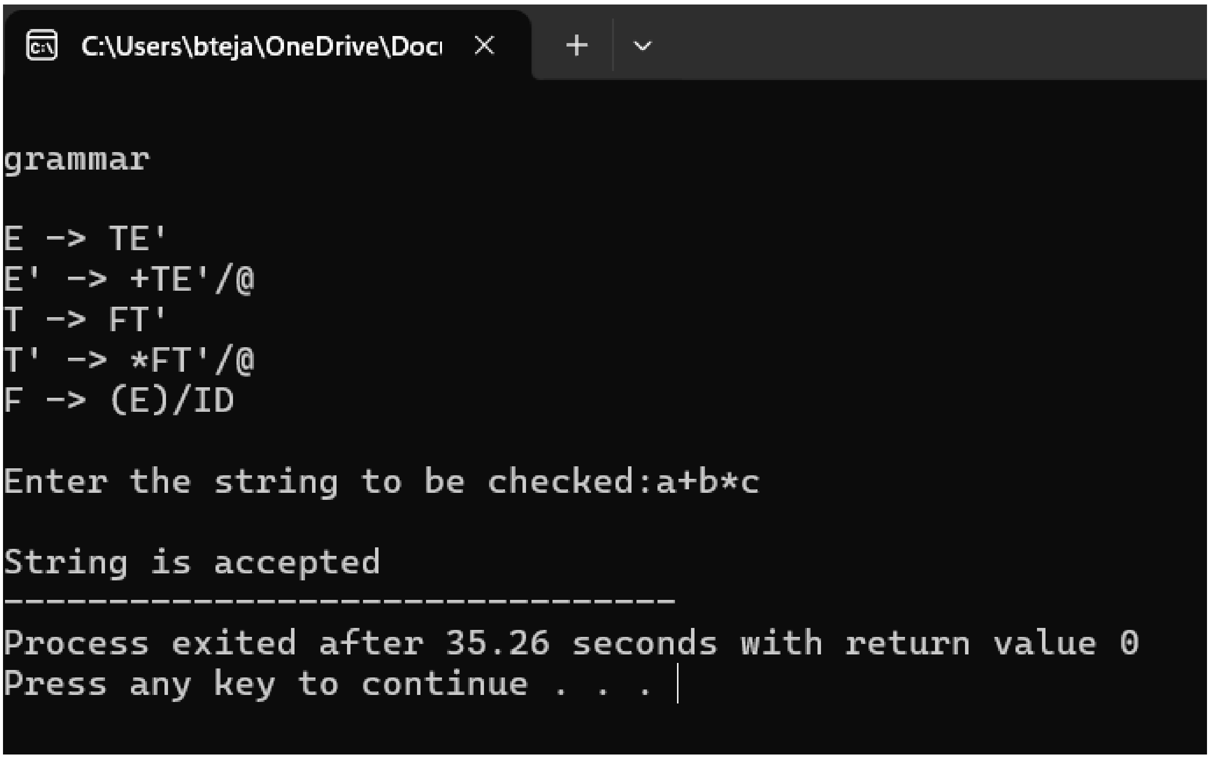
```
1.insert
2.display
3.search
4.modify
5.exit
1
enter the labela+b
enter the address1

1.insert
2.display
3.search
4.modify
5.exit
2
a+b      1

1.insert
2.display
3.search
4.modify
5.exit
```

32. All languages have Grammar. When people frame a sentence we usually say whether the sentence is framed as per the rules of the Grammar or Not. Similarly use the same ideology , implement to check whether the given input string is satisfying the grammar or not

PROGRAM:

```
#include <stdio.h>

#include <string.h>


char input[100];

int i;


int E();
int EP();
int T();
int TP();
int F();

int main(void) {
    printf("\ngrammar\n");
    printf("\nE -> TE'\nE' -> +TE'/@\nT -> FT'\nT' -> *FT'/@\nF -> (E)/ID\n");
    printf("\nEnter the string to be checked:");
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = '\0'; // Removing trailing newline

    i = 0; // Initialize index
    if (E()) {
        if (input[i] == '\0')
            printf("\nString is accepted");
        else
            printf("\nString is not accepted");
    } else
        printf("\nString not accepted");

    return 0;
}

int E() {
    if (T()) {
        if (EP())
            return 1;
        else
            return 0;
    } else
        return 0;
}

int EP() {
    if (input[i] == '+') {
        i++;
        if (T()) {
            if (EP())
                return 1;
            else
                return 0;
        } else
            return 0;
    } else
        return 1;
}

int T() {
    if (F()) {
        if (TP())
            return 1;
        else
            return 0;
    } else
        return 0;
}

int TP() {
    if (input[i] == '*') {
        i++;
        if (F()) {
            if (TP())
                return 1;
            else
                return 0;
        } else
            return 0;
    } else
        return 1;
}

int F() {
    if (input[i] == '(') {
        i++;
        if (E()) {
            if (input[i] == ')') {
                i++;
                return 1;
            } else
                return 0;
        } else
            return 0;
    } else if ((input[i] >= 'a' && input[i] <= 'z') || (input[i] >= 'A' && input[i] <= 'Z')) {
        i++;
        return 1;
    } else
        return 0;
}
```

OUTPUT:



```
grammar

E -> TE'
E' -> +TE'/@
T -> FT'
T' -> *FT'/@
F -> (E)/ID

Enter the string to be checked:a+b*c

String is accepted
--------------------------------
Process exited after 35.26 seconds with return value 0
Press any key to continue . . .
```

33.Write a C program to construct recursive descent parsing

PROGRAM:

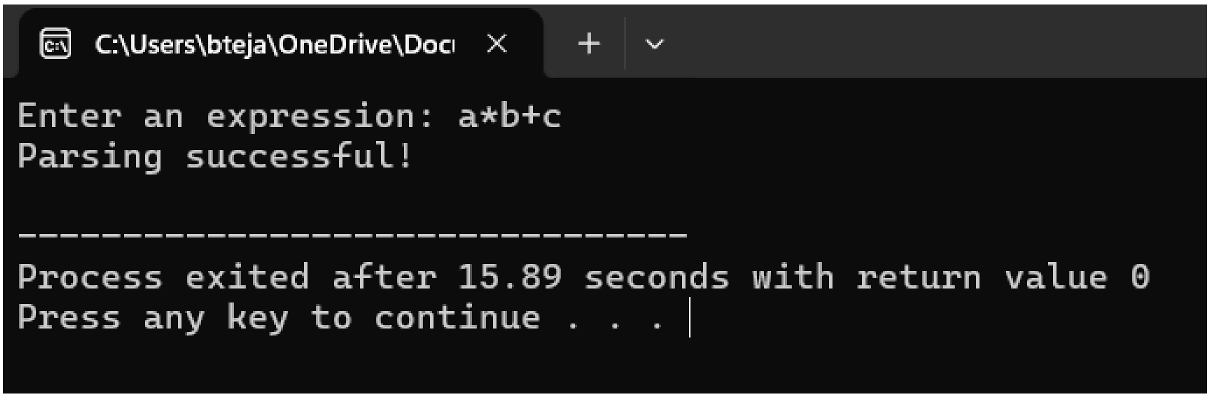
```
#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>


// Token types

typedef enum {

   TOKEN_ID,

   TOKEN_PLUS,

   TOKEN_STAR,

   TOKEN_LPAREN,

   TOKEN_RPAREN,

   TOKEN_EOF,

   TOKEN_UNKNOWN

} TokenType;
```

```
// Token structure
typedef struct {
    TokenType type;
    char value[100];
} Token;

// Lexer variables
const char *input;
int pos = 0;
Token currentToken;

// Function declarations
Token getNextToken();
void parseE();
void parseEPrime();
void parseT();
void parseTPrime();
void parseF();
void error(const char *message);

// Lexer function to get the next token
Token getNextToken() {
    while (input[pos] != '\0') {
        if (isspace(input[pos])) {
            pos++;
            continue;
        }

        if (isalpha(input[pos])) {
            Token token = { TOKEN_ID, {0} };
            int length = 0;
            while (isalnum(input[pos])) {
                token.value[length++] = input[pos++];
            }
            return token;
        }

        switch (input[pos]) {
            case '+':
                pos++;
                return (Token){TOKEN_PLUS, "+"};
            case '*':
                pos++;
                return (Token){TOKEN_STAR, "*"};
            case '(':
                pos++;
                return (Token){TOKEN_LPAREN, "("};
            case ')':
                pos++;
                return (Token){TOKEN_RPAREN, ")"};
            default:
                pos++;
                return (Token){TOKEN_UNKNOWN, {input[pos - 1]}};
        }
    }
    return (Token){TOKEN_EOF, ""};
}

// Error reporting function
```

```
void error(const char *message) {
    printf("Error: %s\n", message);
    exit(1);
}

// Parsing functions
void parseE() {
    parseT();
    parseEPrime();
}

void parseEPrime() {
    if (currentToken.type == TOKEN_PLUS) {
        currentToken = getNextToken();
        parseT();
        parseEPrime();
    }
}

void parseT() {
    parseF();
    parseTPrime();
}

void parseTPrime() {
    if (currentToken.type == TOKEN_STAR) {
        currentToken = getNextToken();
        parseF();
        parseTPrime();
    }
```

```
}

void parseF() {
    if (currentToken.type == TOKEN_ID) {
        currentToken = getNextToken();
    } else if (currentToken.type == TOKEN_LPAREN) {
        currentToken = getNextToken();
        parseE();
        if (currentToken.type != TOKEN_RPAREN) {
            error("Expected ')'");
        }
        currentToken = getNextToken();
    } else {
        error("Expected identifier or '('");
    }
}

// Main function to run the parser
int main() {
    char userInput[256];
    printf("Enter an expression: ");
    fgets(userInput, sizeof(userInput), stdin);

    // Remove newline character if present
    size_t len = strlen(userInput);
    if (len > 0 && userInput[len - 1] == '\n') {
        userInput[len - 1] = '\0';
    }

    input = userInput;
    pos = 0;
    currentToken = getNextToken();

    parseE();

    if (currentToken.type != TOKEN_EOF) {
        error("Unexpected token at the end");
    }

    printf("Parsing successful!\n");
    return 0;
}
```

OUTPUT:



```
Enter an expression: a*b+c
Parsing successful!

--------------------------------
Process exited after 15.89 seconds with return value 0
Press any key to continue . . .
```

34. In a class of Grade 3, Mathematics Teacher asked for the Acronym PEMDAS?. All of them are thinking for a while. A smart kid of the class Kishore of the class says it is Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction. Can you write a C Program to help the students to understand about the operator precedence parsing for an expression containing more than one operator, the order of evaluation depends on the order of operations

PROGRAM:

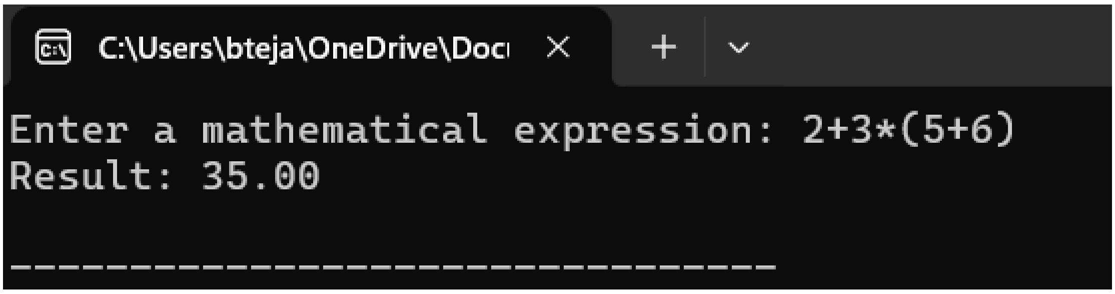
```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
```

```
#include<string.h>
#define MAX 100

// Stack to hold operators
char operators[MAX];
int top_operators = -1;

// Stack to hold values
double values[MAX];
int top_values = -1;

// Function prototypes
void push_operator(char);
char pop_operator();
void push_value(double);
double pop_value();
double apply_operator(char, double, double);
int precedence(char);
void evaluate_top();

void push_operator(char op) {
    operators[++top_operators] = op;
}

char pop_operator() {
    if (top_operators == -1) {
        printf("Error: Operator stack underflow\n");
        exit(1);
    }
    return operators[top_operators--];
```

```
}

void push_value(double val) {
    values[++top_values] = val;
}

double pop_value() {
    if (top_values == -1) {
        printf("Error: Value stack underflow\n");
        exit(1);
    }
    return values[top_values--];
}

double apply_operator(char op, double a, double b) {
    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;
        case '^': return pow(a, b);
        default:
            printf("Error: Unknown operator '%c'\n", op);
            exit(1);
    }
```

```
}

int precedence(char op) {
    switch (op) {
        case '+':
        case '-': return 1;
        case '*':
        case '/': return 2;
        case '^': return 3;
        case '(': return 0;
        default:
            printf("Error: Unknown operator '%c'\n", op);
            exit(1);
    }
}

void evaluate_top() {
    double b = pop_value();
    double a = pop_value();
    char op = pop_operator();
    double result = apply_operator(op, a, b);
    push_value(result);
}

void evaluate_expression(const char *expression) {
    for (int i = 0; expression[i] != '\0'; i++) {
        char c = expression[i];

        if (isspace(c)) {
            continue;
        }

        if (isdigit(c) || c == '.') {
            char number[MAX];
            int number_index = 0;

            while (isdigit(expression[i]) || expression[i] == '.') {
                number[number_index++] = expression[i++];
            }
            number[number_index] = '\0';
            i--;

            push_value(atof(number));
        } else if (c == '(') {
            push_operator(c);
        } else if (c == ')') {
            while (operators[top_operators] != '(') {
                evaluate_top();
            }
            pop_operator(); // pop the '('
        } else if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^') {
            while (top_operators != -1 && precedence(operators[top_operators]) >= precedence(c)) {
                evaluate_top();
            }
            push_operator(c);
        } else {
            printf("Error: Invalid character '%c'\n", c);
            exit(1);
        }
    }
    while (top_operators != -1) {
        evaluate_top();
    }

    printf("Result: %.2f\n", pop_value());
}

int main() {
    char expression[MAX];
    printf("Enter a mathematical expression: ");
    fgets(expression, sizeof(expression), stdin);

    // Remove newline character from the input if it exists
    size_t len = strlen(expression);
    if (len > 0 && expression[len - 1] == '\n') {
        expression[len - 1] = '\0';
    }

    evaluate_expression(expression);

    return 0;
}
```

OUTPUT:



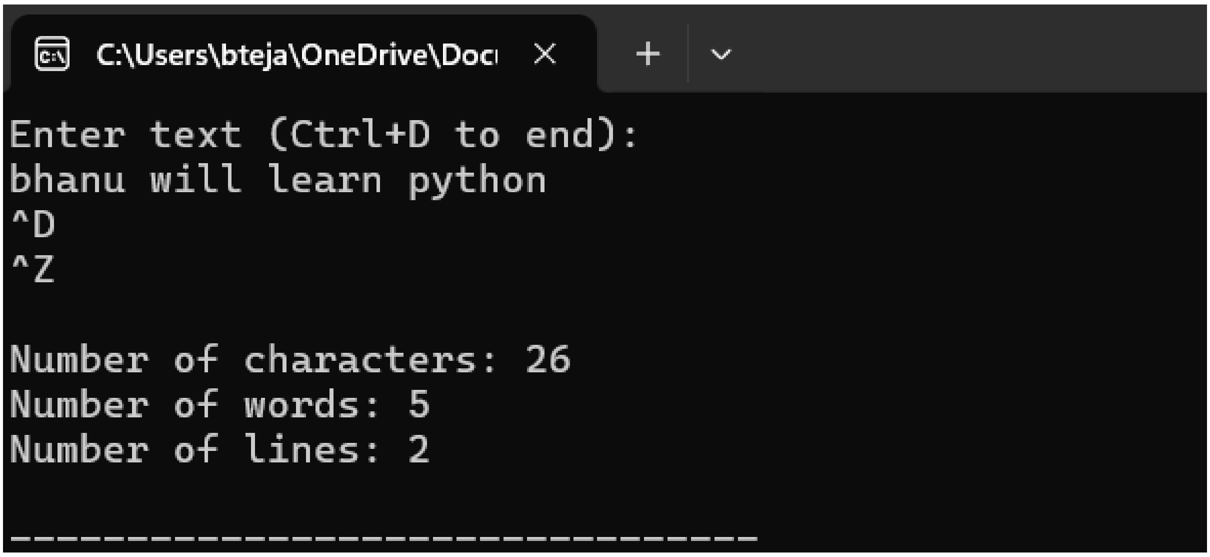
```
Enter a mathematical expression: 2+3*(5+6)
Result: 35.00

--------------------------------
```

35. Write a C program for implementing a Lexical Analyzer to Count the number of characters, words, and lines

PROGRAM:

```
#include <stdio.h>
#include <ctype.h>

#define MAX_LINE_LENGTH 1000

int main() {
    char line[MAX_LINE_LENGTH];
    int char_count = 0, word_count = 0, line_count = 0;
    int in_word = 0;  // Flag to track if we're currently in a word

    printf("Enter text (Ctrl+D to end):\n");

    // Read input until EOF
    while (fgets(line, MAX_LINE_LENGTH, stdin) != NULL) {
        line_count++; // Count each line

        for (int i = 0; line[i] != '\0'; i++) {
            char_count++; // Count each character

            // Check for word boundaries
            if (isspace(line[i])) {
                in_word = 0; // Not in a word
            } else if (!in_word) {
                in_word = 1; // Start of a new word
                word_count++; // Count each word
            }
        }
    }
    // Output results
    printf("\n");
    printf("Number of characters: %d\n", char_count);
    printf("Number of words: %d\n", word_count);
    printf("Number of lines: %d\n", line_count);

    return 0;
}
```

OUTPUT:



```
Enter text (Ctrl+D to end):
bhanu will learn python
^D
^Z

Number of characters: 26
Number of words: 5
Number of lines: 2

--------------------------------
```