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

/* Global declarations */

/* Variables */
int charClass;
char lexeme[100];
char nextChar;
int lexLen;
int token;
int nextToken;
FILE *in_fp;

/* Function declarations */
void addChar();
void getChar();
void getNonBlank();
int lex();
int lookup(char ch);

/* Recursive functions */
void expr();
void term();
void factor();
void error();

/* Character classes */
#define LETTER 0
#define DIGIT 1
#define UNKNOWN 99

/* Token codes */
#define INT_LIT 10
#define IDENT 11
#define ASSIGN_OP 20
#define ADD_OP 21
#define SUB_OP 22
#define MULT_OP 23
#define DIV_OP 24
#define LEFT_PAREN 25
#define RIGHT_PAREN 26
/*****************************************************/
/* main driver */
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    if ((in_fp = fopen(argv[1], "r")) == NULL) {
        printf("ERROR - cannot open %s\n", argv[1]);
        return 1;
    }

    getChar();
    do {
        lex();
        expr();
    } while (nextToken != EOF);

    fclose(in_fp);
    return 0;
}

void expr() {
    printf("Enter <expr>\n");
    term();
    while (nextToken == ADD_OP || nextToken == SUB_OP) {
        lex();
        term();
    }
    printf("Exit <expr>\n");
}

void term() {
    printf("Enter <term>\n");
    factor();
    while (nextToken == MULT_OP || nextToken == DIV_OP) {
        lex();
        factor();
    }
    printf("Exit <term>\n");
}

void factor() {
    printf("Enter <factor>\n");
    if (nextToken == INT_LIT || nextToken == IDENT)
        lex();
    else if (nextToken == LEFT_PAREN) {
        lex();
        expr();
        if (nextToken == RIGHT_PAREN)
            lex();
        else
            error();
    } else
        error();
    printf("Exit <factor>\n");
}

void error() {
    printf("SYNTAX ERROR\n");
}

/*****************************************************/
/* lookup - a function to lookup operators and parentheses
and return the token */
int lookup(char ch)
{
        switch (ch)
        {
        case '(':
              addChar();
              nextToken = LEFT_PAREN;
              break;
        case ')':
              addChar();
              nextToken = RIGHT_PAREN;
              break;
        case '+':
              addChar();
              nextToken = ADD_OP;
              break;
        case '-':
              addChar();
              nextToken = SUB_OP;
              break;
        case '*':
              addChar();
              nextToken = MULT_OP;
              break;
        case '/':
              addChar();
              nextToken = DIV_OP;
              break;
        default:
              addChar();
              nextToken = UNKNOWN;
              break;
        }
        return nextToken;
}
/*****************************************************/
/* addChar - a function to add nextChar to lexeme */
void addChar()
{
        if (lexLen <= 98)
        {
              lexeme[lexLen++] = nextChar;
              lexeme[lexLen] = '\0';
        }
        else
              printf("Error - lexeme is too long \n");
}
/*****************************************************/
/* getChar - a function to get the next character of
input and determine its character class */
void getChar()
{
        if ((nextChar = getc(in_fp)) != EOF)
        {
              if (isalpha(nextChar))
                    charClass = LETTER;
              else if (isdigit(nextChar))
                    charClass = DIGIT;
              else
                    charClass = UNKNOWN;
        }
        else
              charClass = EOF;
}
/*****************************************************/
/* getNonBlank - a function to call getChar until it
returns a non-whitespace character */
void getNonBlank()
{
        while (isspace(nextChar))
              getChar();
}
/*
*****************************************************/
/* lex - a simple lexical analyzer for arithmetic
expressions */
int lex()
{
        lexLen = 0;
        getNonBlank();
        switch (charClass)
        {
        /* Parse identifiers */
        case LETTER:
              addChar();
              getChar();
              while (charClass == LETTER || charClass == DIGIT)
              {
                    addChar();
                    getChar();
              }
              nextToken = IDENT;
              break;
        /* Parse integer literals */
        case DIGIT:
              addChar();
              getChar();
              while (charClass == DIGIT)
              {
                    addChar();
                    getChar();
              }
              nextToken = INT_LIT;
              break;
        /* Parentheses and operators */
        case UNKNOWN:
              lookup(nextChar);
              getChar();
              break;
        /* EOF */
        case EOF:
              nextToken = EOF;
              lexeme[0] = 'E';
              lexeme[1] = 'O';
              lexeme[2] = 'F';
              lexeme[3] = '\0';
              break;
        } /* End of switch */
        printf("Next token is: %d, Next lexeme is %s\n",
               nextToken, lexeme);
        return nextToken;
} /* End of function lex */

```

```
PS C:\Users\amnes\Documents\GitHub\CECS-342\lab2> ./lab2 Test1.txt
Next token is: 25, Next lexeme is (
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 11, Next lexeme is sum
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 21, Next lexeme is +
Exit <factor>
Exit <term>
Next token is: 10, Next lexeme is 47
Enter <term>
Enter <factor>
Next token is: 26, Next lexeme is )
Exit <factor>
Exit <term>
Next token is: 24, Next lexeme is /
Exit <factor>
Next token is: 11, Next lexeme is total
Enter <factor>
Next token is: -1, Next lexeme is EOF
Exit <factor>
Exit <term>
Exit <expr>
```

```
PS C:\Users\amnes\Documents\GitHub\CECS-342\lab2> ./lab2 Test2.txt
Next token is: 10, Next lexeme is 50
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 10, Next lexeme is 47
Exit <factor>
Exit <term>
Exit <expr>
Next token is: 24, Next lexeme is /
Enter <expr>
Enter <term>
SYNTAX ERROR
Exit <factor>
Next token is: 11, Next lexeme is x
Enter <factor>
Next token is: -1, Next lexeme is EOF
Exit <factor>
Exit <term>
Exit <expr>
```

```
PS C:\Users\amnes\Documents\GitHub\CECS-342\lab2> ./lab2 Test3.txt
Next token is: 25, Next lexeme is (
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 11, Next lexeme is sum
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 21, Next lexeme is +
Exit <factor>
Exit <term>
Next token is: 10, Next lexeme is 47
Enter <term>
Enter <factor>
Next token is: 24, Next lexeme is /
Exit <factor>
Next token is: 11, Next lexeme is total
Enter <factor>
Next token is: -1, Next lexeme is EOF
Exit <factor>
Exit <term>
Exit <expr>
SYNTAX ERROR
Exit <factor>
Exit <term>
Exit <expr>
```