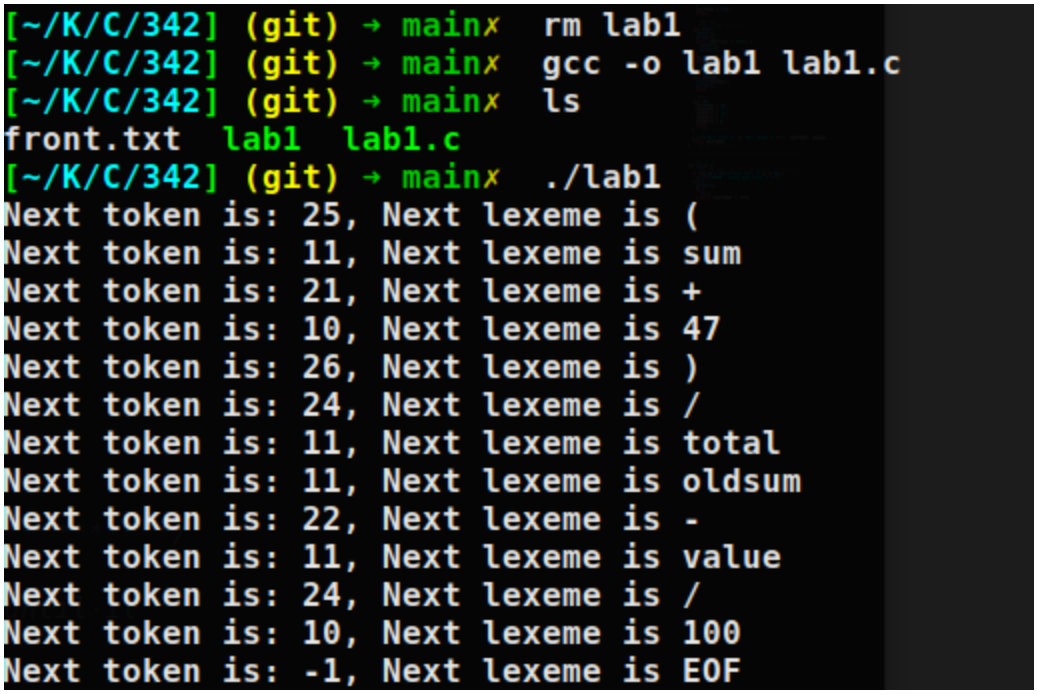
```c
/* A lexical analyzer system for simple
arithmetic expressions */
#include <ctype.h>
#include <stdio.h>

/* Global declarations */

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

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

/* 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

/*****************************************************/

/* lookup - a function to lookup operators and parentheses
and return the token */
```

```
// correct it if you have any idea
int lookup(char ch) {
  switch (ch) {
  case '(':
    addChar();
    nextToken = LEFT_PAREN;
    break;
  case ')':
    addChar();
    nextToken = RIGHT_PAREN;
    break;
  // YOUR CODE
  case '+':
    addChar();
    nextToken = ADD_OP;
    break;
  case '-':
    addChar();
    nextToken = SUB_OP;
    break;
  case '*':
    addChar();
    nextToken = MULT_OP;
    break;
  case '/':
    addChar();
    nextToken = DIV_OP;
    break;
  case '=':
    addChar();
    nextToken = ASSIGN_OP;
    break;
  default:
    if (isalpha(ch)) {
      addChar();
      nextToken = IDENT;
    } else if (isdigit(ch)) {
      addChar();
      nextToken = INT_LIT;
    } else {
      nextToken = UNKNOWN;
    }
```

```
    break;
  }
  return nextToken;
}

/*****************************************************/

/* addChar - a function to add nextChar to lexeme */
void addChar() {
  if (lexLen <= 98) {
    // YOUR CODE
    lexeme[lexLen++] = nextChar;
    lexeme[lexLen] = '\0'; // add null terminator for a valid string
  } 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;
    // YOUR CODE
    else if (isdigit(nextChar))
      charClass = DIGIT;
    else
      // YOUR CODE;
      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;

  // YOUR CODE
  /* Parse integer literals */
  case DIGIT:
    addChar();
    getChar();
    while (charClass == DIGIT) {
      addChar();
      getChar();
    }
    nextToken = INT_LIT;
    break;

  // YOUR CODE
  /* 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 */

/*****************************************************/

/* main driver */
int main() {
  /* Open the input data file and process its contents */
  if ((in_fp = fopen("front.txt", "r")) == NULL)
    printf("ERROR - cannot open front.in \n");
  else {
    getChar();
    do {
      lex();
    } while (nextToken != EOF);
  }
}
```



The program run successfully with correct output (100% completion)