**Lexical Analyzer for Simple Arithmetic Expressions**

Submitted by: Abdulrahman Diaa

CRN: 3880

Date: April 2025

ID:200048408

1. Introduction  
   A compiler translates high-level programming languages into machine code. This process occurs in several well-defined phases, with the lexical analysis phase being the first. In this phase, the source code is read and divided into meaningful components known as tokens. The purpose of this report is to describe the implementation of a lexical analyzer for simple arithmetic expressions written in C.

Description

**1. Header Comments and Includes**

**/\* front.c - a lexical analyzer system for simple arithmetic expressions \*/**

**#include <stdio.h>**

**#include <ctype.h>**

* The comment describes the file as a lexical analyzer for arithmetic expressions
* stdio.h provides input/output functions
* ctype.h provides character classification functions

1. **Global Declarations**

**/\* Global declarations \*/**

**/\* Variables \*/**

**int charClass;**

**char lexeme[100];**

**char nextChar;**

**int lexLen;**

**int nextToken;**

**FILE \*in\_fp;**

* charClass: Stores the classification of the current character (letter, digit, etc.)
* lexeme[100]: Buffer to store the current lexeme (max 99 chars + null terminator)
* nextChar: Stores the next character to process
* lexLen: Tracks the length of the current lexeme
* nextToken: Stores the token code for the current lexeme
* in\_fp: File pointer for input

1. **Function Prototypes**

**void addChar();**

**void getChar();**

**void getNonBlank();**

**int lex();**

**int lookup(char ch);**

* Declares all functions before they're defined

**4. Character Class Definitions**

**#define LETTER 0**

**#define DIGIT 1**

**#define UNKNOWN 99**

* Constants for character classification:
  + LETTER: Alphabetic characters
  + DIGIT: Numeric characters
  + UNKNOWN: All other characters
    1. **Token Code Definitions**

**#define INT\_LIT 10**

**#define IDENT 11**

**#define ADD\_OP 21**

**#define SUB\_OP 22**

**#define MULT\_OP 23**

**#define DIV\_OP 24**

**#define LEFT\_PAREN 25**

**#define RIGHT\_PAREN 26**

**#define EOF\_TOKEN -1**

* Numeric codes for different token types:
  + Integer literals, identifiers, operators, parentheses, and EOF
  + **6. Main Function**

**int main() {**

**if ((in\_fp = fopen("front.in", "r")) == NULL) {**

**printf("ERROR - cannot open front.in \n");**

**return 1;**

**} else {**

**getChar();**

**do {**

**lex();**

**} while (nextToken != EOF\_TOKEN);**

**}**

**return 0;**

**}**

* Opens input file "front.in"
* If open fails, prints error and exits
* Otherwise:
  + Gets first character
  + Repeatedly calls**lex() until EOF token is encountered**

**7. addChar Function**

**void addChar() {**

**if (lexLen <= 98) {**

**lexeme[lexLen] = nextChar;**

**lexeme[lexLen + 1] = '\0';**

**lexLen++;**

**} else {**

**printf("Error - lexeme is too long \n");**

**}**

**}**

* Adds nextChar to lexeme buffer
* Maintains null-terminated string
* Checks for buffer overflow

**8. getChar Function**

**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\_TOKEN;**

**}**

**}**

* Reads next character from file
* Classifies it as LETTER, DIGIT, or UNKNOWN
* Sets EOF\_TOKEN if end of file reached

**9. getNonBlank Function**

**void getNonBlank() {**

**while (isspace(nextChar)) {**

**getChar();**

**}**

**}**

* Skips whitespace characters by reading until non-whitespace found

**10. lookup Function**

**int lookup(char ch) {**

**switch (ch) {**

**case '(':**

**addChar();**

**nextToken = LEFT\_PAREN;**

**break;**

**/\* Other cases similarly \*/**

**default:**

**addChar();**

**nextToken = EOF\_TOKEN;**

**break;**

**}**

**return nextToken;**

**}**

* Identifies operators and parentheses
* Returns appropriate token code
* Default case handles **unknown characters as EOF**

**11. lex Function (Main Lexical Analyzer)**

**int lex() {**

**lexLen = 0;**

**getNonBlank();**

**switch (charClass) {**

**case LETTER: /\* Identifier processing \*/**

**addChar();**

**getChar();**

**while (charClass == LETTER || charClass == DIGIT) {**

**addChar();**

**getChar();**

**}**

**nextToken = IDENT;**

**break;**

**case DIGIT: /\* Integer literal processing \*/**

**addChar();**

**getChar();**

**while (charClass == DIGIT) {**

**addChar();**

**getChar();**

**}**

**nextToken = INT\_LIT;**

**break;**

**case UNKNOWN: /\* Operator/parenthesis \*/**

**lookup(nextChar);**

**getChar();**

**break;**

**case EOF\_TOKEN: /\* End of file \*/**

**nextToken = EOF\_TOKEN;**

**lexeme[0] = 'E';**

**lexeme[1] = 'O';**

**lexeme[2] = 'F';**

**lexeme[3] = '\0';**

**break;**

**}**

**printf("Next token is: %d, Next lexeme is %s\n", nextToken, lexeme);**

**return nextToken;**

**}**

* Main lexical analysis function
* Handles:
  + Identifiers (letters followed by letters/digits)
  + Integer literals (digits)
  + Operators/parentheses
  + EOF
* Prints token information
* Returns token code