A black background with grey leaves

AI-generated content may be incorrect.

Lexical Analyzer

Build Scanner

A grey logo on a black background

AI-generated content may be incorrect.

**Prepared By**

Name : Marwan Gamal

ID : 200045173

**Under Supervision**

Name of Doctor

Name of T. A.

1. **Introduction**
   1. **Phases of Compiler :**

 **Lexical Analysis**: Convert source code into a sequence of tokens.

 **Syntax Analysis**: Parse the token stream to detect grammatical structure.

 **Semantic Analysis**: Check for semantic consistency.

 **Code Generation**: Translate into target machine code.

1.  **Lexical Analyzer : Read the source code and convert it into a stream of tokens**
2. **Software Tools**
   1. **Computer Program : VS**
   2. **Programming Language: C**
3. **Implementation of a Lexical Analyzer**

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

**#include <stdio.h> // For input/output functions**

**#include <ctype.h> // For character classification functions like isalpha(), isdigit(), etc.**

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

**/\* Variables \*/**

**int charClass; // Class of current character (LETTER, DIGIT, UNKNOWN, EOF)**

**char lexeme[100]; // Array to store the current lexeme**

**char nextChar; // The current character read from input**

**int lexLen; // Current length of the lexeme**

**int token; // Stores the token type returned by lex()(not used directly)**

**int nextToken; // Stores the current token code**

**FILE \*in\_fp, \*fopen(); // File pointer for input file**

**/\* Function declarations \*/**

**void addChar(); // Adds current character to the lexeme**

**void getChar(); // Gets the next character and sets its class**

**void getNonBlank(); // Skips whitespace**

**int lex(); // Main lexical analyzer function**

**/\* Character classes \*/**

**#define LETTER 0 // Represents alphabetic character**

**#define DIGIT 1 // Represents numeric character**

**#define UNKNOWN 99 // Represents any other character**

**/\* Token codes \*/**

**#define INT\_LIT 10 // Integer literal**

**#define IDENT 11 // Identifier**

**#define ASSIGN\_OP 20 // Assignment operator (=)**

**#define ADD\_OP 21 // Addition operator (+)**

**#define SUB\_OP 22 // Subtraction operator (-)**

**#define MULT\_OP 23 // Multiplication operator (\*)**

**#define DIV\_OP 24 // Division operator (/)**

**#define LEFT\_PAREN 25 // Left parenthesis (**

**#define RIGHT\_PAREN 26 // Right parenthesis )**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* main driver \*/**

**main() {**

**// Open the input file "front.in" for reading**

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

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

**else {**

**getChar(); // Read the first character**

**do {**

**lex(); // Analyze and tokenize each lexeme**

**} while (nextToken != EOF); // Continue until end of file**

**}**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* lookup : function for identifying singlecharacter operators and parentheses and assigning them the correct token type. \*/**

**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 = EOF; // Unknown character**

**break;**

**}**

**return nextToken;**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* addChar : adds nextChar to the lexeme buffer \*/**

**void addChar() {**

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

**lexeme[lexLen++] = nextChar; // Add character and increment length**

**lexeme[lexLen] = 0; // Null terminate string**

**} else**

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

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* getChar : reads next character from input and sets 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;**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* skips whitespace characters \*/**

**void getNonBlank() {**

**while (isspace(nextChar))**

**getChar();**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* the lexical analyzer function \*/**

**int lex() {**

**lexLen = 0; // Reset lexeme length**

**getNonBlank(); // Skip whitespace**

**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;**

**/\* Operators and parentheses \*/**

**case UNKNOWN:**

**lookup(nextChar); // Look up the token type**

**getChar();**

**break;**

**/\* End of file \*/**

**case EOF:**

**nextToken = EOF;**

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

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

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

**lexeme[3] = 0;**

**break;**

**}**

**// Print the token and lexeme**

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

**return nextToken;}**

1. **References : CONCEPTS OF PROGRAMMING LANGUAGES BOOK**