

1. **Introduction**

This program implements a **lexical analyzer** (or **scanner**) in the C programming language for parsing simple arithmetic expressions. The primary function of the lexical analyzer is to read characters from an input file (front.in) and group them into meaningful sequences called **tokens**, which are the building blocks of a programming language's syntax.

* 1. **Phases of Compiler**

(sum + 47) / total

1. **Lexical Analyzer**

**int lex()**

Next token is: 10, Next lexeme is 123

Next token is: 21, Next lexeme is +

Next token is: 11, Next lexeme is x

(sum + 47) / total

1. **Software Tools**

**Functions :addChar(),getChar(),getNonBlank(),lex()**

**gcc front.c -o front**

* 1. **Computer Program**

(front.in)

1. **Next token is: 25 Next lexeme is (**
2. **Next token is: 11 Next lexeme is sum**
3. **Next token is: 21 Next lexeme is +**
   1. **Programming Language :c program**
4. **Implementation of a Lexical Analyzer**

**Next token is: 25, Next lexeme is (**

**Next token is: 11, Next lexeme is sum**

**Next token is: 21, Next lexeme is +**

1. **References**

**Concept of programing language book**

**The code**

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

**#include <stdio.h>**

**#include <ctype.h>**

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

**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);**

**/\* 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**

**/\* 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;**

**}**

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

**return nextToken;**

**}**

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

**int main() {**

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

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

**else {**

**getChar();**

**do {**

**lex();**

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

**}**

**}**

**/\* 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 = EOF;**

**break;**

**}**

**return nextToken;**

**}**