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 : Seif El-Din Sherif

ID : 200045176

**Under Supervision**

Doctor : Nehal Abd-elsalam

T. A. : Fares El-Tomy

**1- Phases of a compiler :**

The compiler is a program that translates high level language into low-level language ML (machine language).

A compiler has three phases:

1. Lexical analysis : in this phase the compiler converts the source code into lexical units.
2. Syntax analysis : in this phase the compiler converts the lexical units into parse trees.
3. Semantics analysis : in this phase the compiler generates intermediate code.
4. Code generation : in this phase the machine code is generated.

**2 – lexical analyzer :**

The lexical analyzer converts the source code into meaningful tokens.

**3.1 – computer program :**

**3.2 – programming language :** C

**4 – lexical analyzer implementation :**

Code :

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

#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, \*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

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

/\* main driver \*/

main() {

/\* Open the input data file and process its contents \*/

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;

}

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

/\* 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 \*/

oid getNonBlank() {

while (isspace(nextChar))

getChar();

}

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

/\* lex - a simple lexical analyzer for arithmetic expressions \*/

int lex() {

lexLen = 0;

getNonBlank();

switch (charClass) {

case LETTER:

addChar();

getChar();

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

addChar();

getChar();

}

nextToken = IDENT;

break;

case DIGIT:

addChar();

getChar();

while (charClass == DIGIT) {

addChar();

getChar();

}

nextToken = INT\_LIT;

break;

case UNKNOWN:

lookup(nextChar);

getChar();

break;

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;

}

Code explanation :

This code converts lexemes into tokens,

It begins with simple libraries (<stdio.h>) for input and output functions.

Then global variables used for storing and analyzing the current lexeme to convert it into tokens.

#define LETTER 0 - (used for identifirs).

#define DIGIT 1 (used for digits).

#define UNKNOWN 99 ( used for symbols).

Then the main function used for opening the input file and reads it until its finished.

Then the lex function which identifies the lexeme whether it’s a number, character or symbol.