Lexical Analyzer for Arithmetic Expressions in C

Aser Essam  
  
This is a simple lexical analyzer program I worked on for understanding how compilers break down arithmetic expressions.   
The code is written in C and focuses on recognizing tokens like integers, identifiers, and basic arithmetic operators.

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

#include <stdio.h>

#include <ctype.h>

/\* Global declarations \*/

int charClass;

char lexeme[100]; // stores the current lexeme

char nextChar; // current character being read

int lexLen; // length of the current lexeme

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 // integer literal

#define IDENT 11 // identifier (like variable names)

#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 function \*/

main() {

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

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

else {

getChar();

do {

lex();

} while (nextToken != EOF);

}

}

/\* This function matches single-character tokens like +, -, etc. \*/

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;

}

/\* Adds the current character to the lexeme array \*/

void addChar() {

if (lexLen <= 98) {

lexeme[lexLen++] = nextChar;

lexeme[lexLen] = 0;

} else {

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

}

}

/\* Reads the next character and updates charClass accordingly \*/

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 \*/

void getNonBlank() {

while (isspace(nextChar))

getChar();

}

/\* Main lexical analyzer function \*/

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;

}