// front.cpp - Lexical Analyzer in C++

#include <iostream>

#include <fstream>

#include <cctype>

#include <string>

using namespace std;

// Character classes

#define LETTER 0

#define DIGIT 1

#define UNKNOWN 99

#define END\_OF\_FILE -1

// 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

// Global declarations

int charClass;

string lexeme;

char nextChar;

int nextToken;

ifstream inFile;

// Function declarations

void addChar();

void getChar();

void getNonBlank();

int lookup(char ch);

int lex();

void addChar() {

if (lexeme.length() < 99) {

lexeme += nextChar;

} else {

cout << "Error - lexeme is too long\n"; }}

void getChar() {

if (inFile.get(nextChar)) {

if (isalpha(nextChar))

charClass = LETTER;

else if (isdigit(nextChar))

charClass = DIGIT;

else

charClass = UNKNOWN;

} else {

charClass = END\_OF\_FILE; }}

void getNonBlank() {

while (isspace(nextChar)) {

getChar(); }}

int lookup(char ch) {

switch (ch) {

case '(': addChar(); return LEFT\_PAREN;

case ')': addChar(); return RIGHT\_PAREN;

case '+': addChar(); return ADD\_OP;

case '-': addChar(); return SUB\_OP;

case '\*': addChar(); return MULT\_OP;

case '/': addChar(); return DIV\_OP;

case '=': addChar(); return ASSIGN\_OP;

default: addChar(); return END\_OF\_FILE; }}

int lex() {

lexeme = "";

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:

nextToken = lookup(nextChar);

getChar();

break;

case END\_OF\_FILE:

lexeme = "EOF";

nextToken = END\_OF\_FILE;

break;

}

cout << "Next token is: " << nextToken << ", Next lexeme is " << lexeme << endl;

return nextToken;

}

int main() {

inFile.open("front.in");

if (!inFile) {

cerr << "ERROR - cannot open front.in\n";

return 1; }

getChar();

do {

lex();

} while (nextToken != END\_OF\_FILE);

inFile.close();

return 0;

}

*Explain the code:*

// front.cpp - Lexical Analyzer in C++

A comment explaining the file name and function.

#include <iostream>

<iostream> is a library used to display and receive data from the user (cout,cin).

#include <fstream>

<ifstream> Library for opening and reading files .

#include <cctype>

A library containing functions for processing characters such as isalpha(), isdigit(), and isspace().

#include <string>

statement imports the string library, which provides you with tools for working with strings.

using namespace std;

We write this way to not keep writing (std::)

#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

)#define( statement is used to define constants so that values ​​or expressions in the code are replaced during compilation.

int charClass;

)int( is a data type that indicates integer numbers.

)charClass( variable

string lexeme;

is a data type that indicates the word or symbol we are analyzing..

char nextChar;

is a data type that indicates single characters.

ifstream inFile;

ifstream: Used to read data from files.

inFile: The name of the object from which the data will be read.

void addChar() ;

A function to add the current character nextChar to a lexeme.

void getChar() ;

A function to read the next character from a file and determine its type.

void getNonBlank() ;

A function to skip any spaces or any blank

int lookup(char ch) ;

A function sees what sign and returns what it was,and It returns a constant value representing the appropriate token for this character.

int lex() ;

It reads the current character, determines its type (letter, number, symbol, or end of file), forms a word (lexeme) from it, and returns its type as "Token".

void addChar() {

we said that is a function to add the current character nextChar to a lexeme.

if (lexeme.length() < 99) {

Here we set a condition "If the length of the lexicon we are currently building is less than 99 characters

lexeme += nextChar;

If the if condition is true here, we add the next character (nextChar) to the end of the lexeme.

} else { cout << "Error - lexeme is too long\n";}}

If the if condition is not met, we print this message.

void getChar() {

we said that is a function to read the next character from a file and determine its type.

if (inFile.get(nextChar)) {

Here we set a condition we read the next character from the file (inFile) and store it in nextChar.

If the character reading succeeds, the (if) statement is executed.

if (isalpha(nextChar))

Here we set a condition If the letter we read is the letter (A-Z or a-z) the (if) statement is executed.

charClass = LETTER;

if the condition if [isalpha(nextChar) ] true, We store that the character type is letter.

else if (isdigit(nextChar))

If the if condition is not met [isalpha(nextChar) ], We check if the current character nextChar is a digit or not.

charClass = DIGIT;

if the condition if [else if (isdigit(nextChar))] true, We store that the character type is digit.

}else{charClass = UNKNOWN;}

If it is not a digit or a letter so it is unknown

} else {

charClass = END\_OF\_FILE;}

If there are no characters to read meaning we have reached the end of the file, we store the character type as END\_OF\_FILE.

void getNonBlank() {

we said that is a function to skip any spaces or any blank.

while (isspace(nextChar))

if nextChar is a whitespace , this condition is met.

getChar();}

if the condition met. We use function to read the next character in the file.

int lookup(char ch) {

we said that is a function sees what sign and returns what it was, and It returns a constant value representing the appropriate token for this character.

switch (ch) {