#include <iostream>

// this library for cout and cin statments.

#include <string>

//this library for string.

#include <cctype>

//this library for handling with files.

#include <cstring>

//this library provides functions for handling c-sytyle string.

using namespace std;

//this is statement provides do not write std:: every line.

#define LETTER 0

//represents alphabeic characters (A-Z, a-z)

#define DIGIT 1

//represents numric digits (0-9).

#define UNKNOWN 99

//represents unrecogonized symbols.

// Token codes

#define EOF\_TOKEN -1

//this is line for represent the end of file marker.

#define INT\_LIT 10

// likely represent literal token.

#define IDENT 11

//this is line for represents an identifier token.

#define ASSIGN\_OP 20

//this is line for represents the assigmnet opreator =.

#define ADD\_OP 21

//this is line for represents the addition opreator +.

#define SUB\_OP 22

//this is line for represents the subtraction opreator -.

#define MULT\_OP 23

//this is line for represent the multplication opreator \*.

#define DIV\_OP 24

//this is line for represent the divison opreator /.

#define LEFT\_PAREN 25

//this is line for represents the left paranthesis (.

#define RIGHT\_PAREN 26

//this is line for represent the right paranthesis ).

class Lexer {

//this is the class that contain all processes that related to lexeical analyzer.

private:

//that allow for all attributes used in class only.

int charClass;

//this variable for identify integer variable to store classification of characters.

string lexeme;

// for identify of a text string variable to store lexemes.

char nextChar;

//to store next character which is will be processed.

int token;

// to store the current token.

int nextToken;

// to store the next token.

string input;

//used to store the statement that user are input to be analysing.

int index;

//this variable used to follow the position of the immediatelycharacter that the program is analysing.

public:

//it is flixable that make all attributs used in and out the class.

Lexer(const string& userInput) {

// this is a constructor that take a const reference to a string (filename) like input.

input = userInput;

//it assigns the value of userinput to the member variable input of the lexer class.

index = 0;

//intilizes an index to keep track of the current position while processing the iput text.

getChar();

}

//this is closing parantheses for constructor lexer.

void addChar() {

// If we have not hit the max length yet, add the character to our current word

if (lexeme.length() < 99) {

lexeme += nextChar;

// just tack it on the end

}

//this is closing parantheses for if condition.

else {

// this word is getting too big time to complain

cout << "Error - lexeme is too long" << endl;

}

//this is closing parantheses for else statement.

}

//this is closing parantheses for function addchar.

void getChar() {

//As long as there are still characters remaining to read.

if (index < input.length()) {

// checks if the current index is less than the lengh of the input string.

nextChar = input[index++];

// assigns the character at the current index in input to nextchar.

if (isalpha(nextChar))

// checks if nextchar is an alphapitic letter using isalpha().

charClass = LETTER;

// if a nextchar is a letter.

else if (isdigit(nextChar))

// if not a letter cause may be that it was be a digit.

charClass = DIGIT;

// if nextchar is a digit.

else

charClass = UNKNOWN;

// if nextchar nethier a letter nor a digit.

}

else {

charClass = EOF\_TOKEN;

// if the if condition fails.

}

//this is closing parantheses for else statement.

}

//this is closing parantheses for function getchar.

void getNonBlank() {

// skip past any boring spaces or tabs

while (isspace(nextChar))

getChar();

// just keep grabbing characters until we hit something interesting

}

//this is closing parantheses for function getNonBlank.

int lookup(char ch) {

// check what symbol we are looking at and assign the right token type.

switch (ch) {

case '(': addChar(); nextToken = LEFT\_PAREN; break;

// this is opening parenthesis.

case ')': addChar(); nextToken = RIGHT\_PAREN; break;

// this is represents closing parenthesis.

case '+': addChar(); nextToken = ADD\_OP; break;

// this is represents addition

case '-': addChar(); nextToken = SUB\_OP; break;

// this is represents subtraction

case '\*': addChar(); nextToken = MULT\_OP; break;

// this is represents multiplication

case '/': addChar(); nextToken = DIV\_OP; break;

// this is represents division

case '=': addChar(); nextToken = ASSIGN\_OP; break;

// this is represents equals sign

default: addChar(); nextToken = EOF\_TOKEN; break;

// anything else means we are done

}

//this is closing parantheses for switch.

return nextToken;

// tell the caller what we found

}

//this is closing parantheses for function lookup.

int lex() {

//this functin we calling anthoer functions in it.

lexeme = "";

// start fresh with an empty word

// first, skip any whitespace so we can get to the good stuff

getNonBlank();

// now figure out what we are dealing with

switch (charClass) {

case LETTER:

// starts with a letter - probably a variable name

addChar();

// take this first letter

getChar();

// move to next character

// keep going as long as we are getting letters or numbers

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

addChar();

// add to our growing word

getChar();

// move along

}

//this is closing parantheses for while.

nextToken = IDENT;

// we have got ourselves an identifier

break;

case DIGIT:

// starts with a number - let is read the whole number

addChar();

// take this first digit

getChar();

// on to the next

// keep eating up digits until we hit something else

while (charClass == DIGIT) {

addChar();

getChar();

}

//this is closing parantheses for while statement.

nextToken = INT\_LIT;

// we have collected a complete number

break;

case UNKNOWN:

// some symbol or operator

lookup(nextChar);

// figure out what it is exactly

getChar();

// move forward

break;

case EOF\_TOKEN: // we have reached the end

nextToken = EOF\_TOKEN;

lexeme = "EOF";

// just for clarity in output

break;

}

//this is closing parantheses for switch.

// show what we found (useful for debugging)

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

return nextToken; // pass back what type of token we found

}

//this is closing parantheses for function lex.

void processTokens() {

// keep processing tokens one by one until we run out

do {

lex();

} while (nextToken != EOF\_TOKEN); // stop when we reach the end

}

//this is closing parantheses for function processToken.

};

//this is closing parantheses for class lexer.

int main() {

string userInput = "D = A / F + 2";

// this is the statement lexeme and the program will process it to get the tokens.

cout << "processing statment: " << userInput << endl;

Lexer lexer(userInput);

// start breaking it down into tokens

lexer.processTokens();

//we here calling the function that processing the lexeme statement from class lexeme.

return 0;

// all done!

}

// this is closing parantheses for function main.