Concepts of Computer Programming

Lexical Analyzer

Name: Omar Ashraf Ezzat  
ID: 200036952

Programing language used: C++

Code:

#include <iostream>

#include <string>

#include <cctype>

using namespace std;

#define LETTER 0

#define DIGIT 1

#define UNKNOWN 99

#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 L\_PAREN 25

#define R\_PAREN 26

#define EXP\_OP 27

#define SEMICOLON 28

#define FOR 29

#define L\_BRACE 30

#define R\_BRACE 31

#define End\_OF\_File -1

int charClass;

string lexeme;

char nextChar;

int nextToken;

string input;

size\_t pos = 0;

void addChar();

void getChar();

void getNonBlank();

int lex();

int lookup(char ch);

int main() {

    cout << "Enter an expression:   ";

    getline(cin, input);

    getChar();

    do {

        lex();

    } while (nextToken != End\_OF\_File);

    return 0;

}

int lookup(char ch) {

    switch (ch) {

        case '(':

            addChar();

            nextToken = L\_PAREN;

            break;

        case ')':

            addChar();

            nextToken = R\_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;

        case '=':

            addChar();

            nextToken = ASSIGN\_OP;

            break;

        case ';':

            addChar();

            nextToken = SEMICOLON;

            break;

        case '{':

            addChar();

            nextToken = L\_BRACE;

            break;

        case '}':

            addChar();

            nextToken = R\_BRACE;

            break;

        default:

            addChar();

            nextToken = End\_OF\_File;

            break;

    }

    return nextToken;

}

void addChar() {

    lexeme += nextChar;

}

void getChar() {

    if (pos < input.length()) {

        nextChar = input[pos++];

        if (isalpha(nextChar))

            charClass = LETTER;

        else if (isdigit(nextChar))

            charClass = DIGIT;

        else

            charClass = UNKNOWN;

    } else {

        charClass = End\_OF\_File;

    }

}

void getNonBlank() {

    while (isspace(nextChar) && charClass != End\_OF\_File)

        getChar();

}

int lex() {

    lexeme = "";

    getNonBlank();

    switch (charClass) {

    case LETTER:

    addChar();

    getChar();

    while ((charClass == LETTER || charClass == DIGIT) && charClass != End\_OF\_File) {

        addChar();

        getChar();

    }

    if (lexeme == "for") {

        nextToken = FOR;

    } else {

        nextToken = IDENT;

    }

    break;

        case DIGIT:

            addChar();

            getChar();

            while (charClass == DIGIT && charClass != End\_OF\_File) {

                addChar();

                getChar();

            }

            nextToken = INT\_LIT;

            break;

        case UNKNOWN:

            lookup(nextChar);

            getChar();

            break;

        case End\_OF\_File:

            nextToken = End\_OF\_File;

            lexeme = "End\_OF\_File";

            break;

    }

    if (nextToken != EOF) {

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

    }

    return nextToken;

}

Detailed Explanation on each line of the code:

A black background with text

AI-generated content may be incorrect.

In these lines of code we used the following libraries:

<iostream> for input and output operations

<string> for usage of string objects and it would make it easier

<cctype> for character classification function like

{

isalpha() { checks if the character is a letter}

isdidigt() { checks if the character is a digit}

}

A screenshot of a computer screen

AI-generated content may be incorrect.

In these line of code we use #define to define the character categories for example:

#define LETTER = alphabetic characters

#define DIGIT = numeric characters

#define UNKNOWN = symbols

#define Assign\_op = “=”

#define End\_OF\_File as -1 to represent (end of input)

Etc….

A screen shot of a computer code

AI-generated content may be incorrect.

charClass 🡪 Stores the **character type** (LETTER, DIGIT, or UNKNOWN).

lexeme 🡪 Stores the **current token's characters**.

nextChar 🡪 Holds the **current character** being analyzed.

nextToken 🡪 Holds the **token type** of the current lexeme.

input 🡪 Stores the **user input string**.

pos 🡪 Keeps track of the **current position** in input.

A screen shot of a computer code

AI-generated content may be incorrect.

These are Functions to help with the code ( lexems)

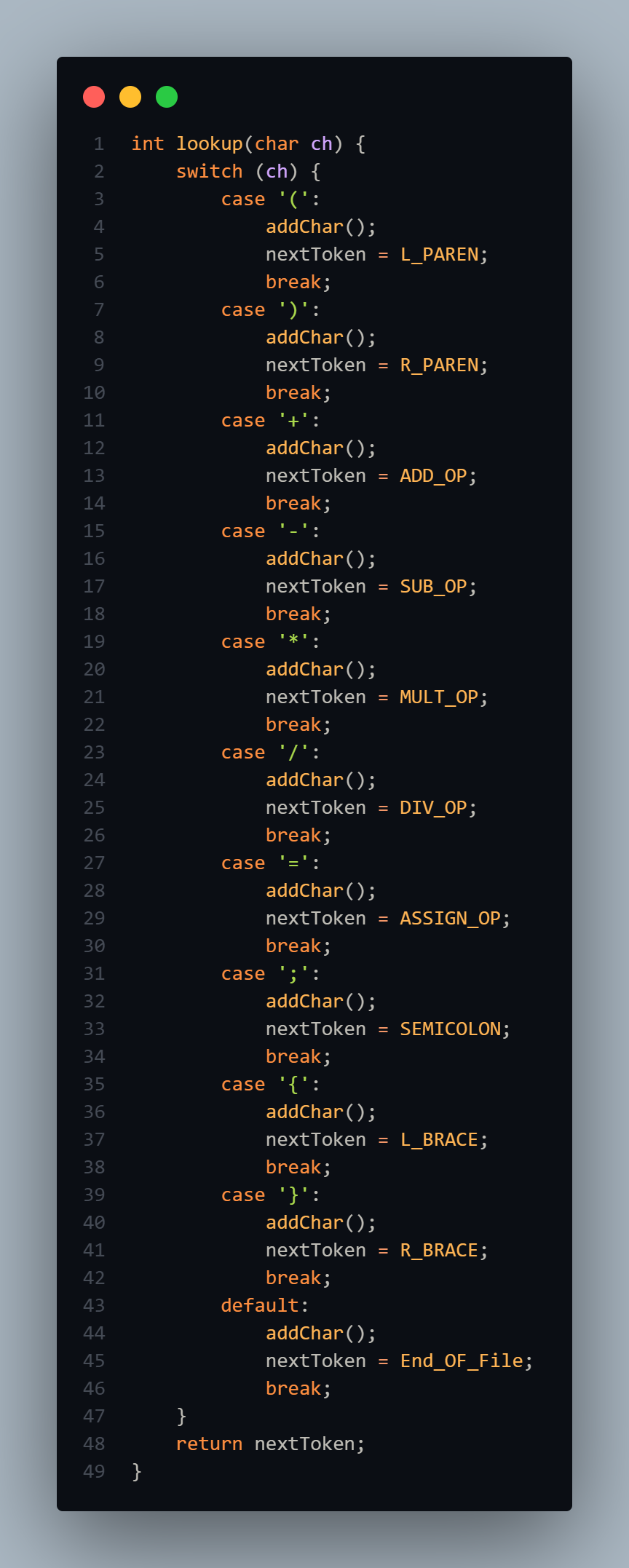
A screen shot of a computer program

AI-generated content may be incorrect.

In this we make the user put the input we use the getline(cin,input) so the program reads entire input statement into the input string

Calling getchar() function to initialize reading and then we run lex() in a do while loop we keep running it till, we get the end of the loop which is End\_Of\_File and that is to ensure that every token in the input is read and processed

And then return 0 is to terminate the program after each token is processed



In the previous page (code) we use lookup() to classify and identify the special characters that are the function itself but, in our case, we use it to determine if nextchar() is known operator or special symbol.

I used switch technique to determine each symbol character, for example if:

Nextchar() is ‘ ( ‘ , ‘ ) ’ , ‘ + ‘ etc… we will add them to the lexeme by addchar() and nexttoken() will be set according to the #define at the beginning of the code like if the lexeme is ‘ = ‘ then the nexttoken() will be assign\_op and etc…

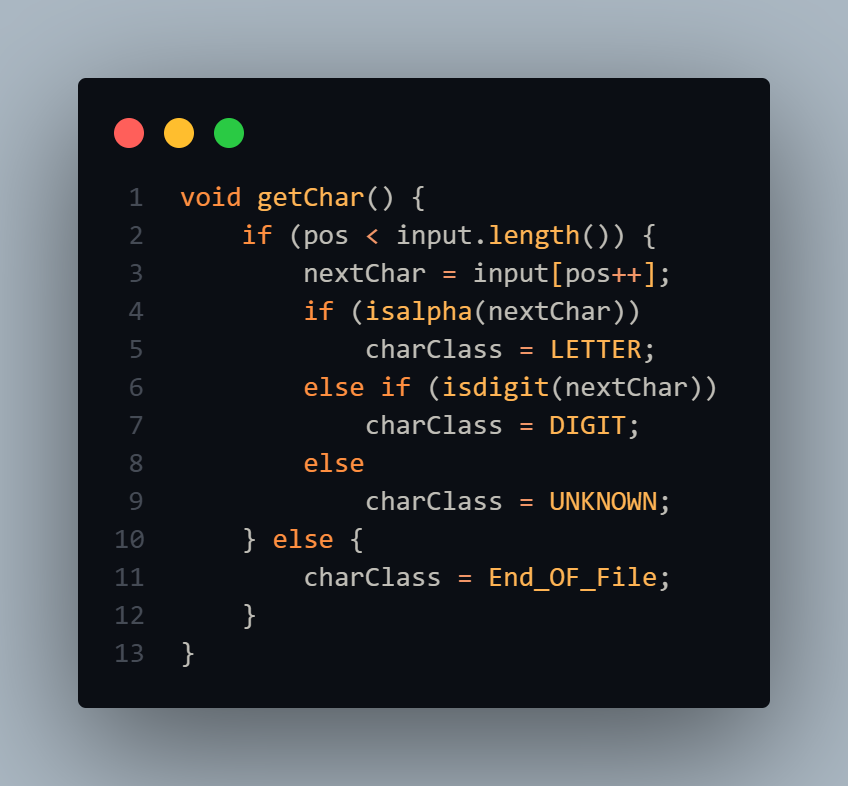
Then we use default that if if the character (ch) is not recognized it will be classified into End\_Of\_File

And then we will return the nexttoken()



In this line of code it is the addchar() which appends

Nextchar() to lexemes forming the tokens



In this code it sees if pos is within the inputs of the code if true it will read nextchar

Then it will classify if nextchar is Letter ( by using function isalpha to determine if the characters are letters)

Or digits ( by using function isdigit to determine if the characters are numbers)

Or it is UKNOW but if it reaches the end of the input it will classify into End\_Of\_File



In this code it Skips spaces and newlines until a non-whitespace character is found.



At the first part of the code before the switch case we clear the lexemes and we remove any leading spaces then we use switch case first at case LETTER it reads the letters and digits to form an identifier but checks if the lexeme is ‘for’ then it will classify as for token

Then we use case DIGITS it will read the digits (numbers) and it will form int\_literal token

Then we use case UKNOWN I used lookup() to to calssify the symbols to put them in their tokens

Then used case End\_Of\_File the mark the end of the input



Then lastly we do the condition that if the nexttoken is not End\_Of\_File we will print the detected token and its lexemes.