**NAME:Sara Khaled eid**

**ID:200039191**

**The Code is written in c++ :-**

#include <iostream>

#include <sstream>

#include <cctype>

#include <cstring>

using namespace std;

int charClass;

char lexeme[100];

char nextChar;

int lexLen;

int token;

int nextToken;

istringstream in\_fp;

void addChar();

void getChar();

void getNonBlank();

int lex();

int lookup(char ch);

#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 LEFT\_PAREN 25

#define RIGHT\_PAREN 26

#define EOF\_TOKEN -1

int main() {

cout << "Enter an expression: ";

string input;

getline(cin, input);

in\_fp.clear();

in\_fp.str(input);

getChar();

do {

lex();

} while (nextToken != EOF\_TOKEN);

return 0;

}

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\_TOKEN; break;

}

return nextToken;

}

void addChar() {

if (lexLen < 98) {

lexeme[lexLen++] = nextChar;

lexeme[lexLen] = '\0';

} else {

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

}

}

void getChar() {

if (in\_fp.get(nextChar)) {

if (isalpha(nextChar))

charClass = LETTER;

else if (isdigit(nextChar))

charClass = DIGIT;

else

charClass = UNKNOWN;

} else {

charClass = EOF\_TOKEN;

nextChar = '\0';

}

}

void getNonBlank() {

while (isspace(nextChar))

getChar();

}

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\_TOKEN:

nextToken = EOF\_TOKEN;

strcpy(lexeme, "EOF");

break;

}

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

return nextToken;

}

ا**لشرح:-**  #include <iostream> :This line is a library used in C++ for input and output operations.

#include <sstream>:This line is a library used to handle text as data that can be read and processed**.**

#include <cctype>:This library allows me to process characters individually, not the whole word at once.

#include <cstring>:This line is a library that helps us handle text stored as character arrays.

using namespace std; : This line using namespace std; allows us to use commands like cout and cin without writing std:: before them every time.

int charClass;: This line defines a variable named charClass as a global variable of type int.

char lexeme[100]; :This line defines an array named lexeme of type char array with a size of 100 characters.

char nextChar;: This line defines a variable named nextChar of type char, used to store the current character.

**int lexLen;**  
This line defines a variable named lexLen of type int, used to store the length of the word.

**int token;**  
This line declares an int variable named token. It will be used to store the token code.

**int nextToken;**  
The line is an int variable named nextToken, which is used to store the token code of the lexeme.

**istringstream in\_fp;**  
The line is a variable named in\_fp of type istringstream, which is used to read data as a string.

**void addChar();**  
This line is a function named addChar of type void, meaning it does not return any value.

**void getChar();**  
This line is a function named getChar of type void, meaning it does not return any value. Its purpose is to read the next character from the input and determine its type.

**void getNonBlank();**  
This line is a function named getNonBlank of type void, meaning it does not return any value.

**int lex();**  
This line is a function named lex of type int. Its purpose is to perform lexical analysis on the input.

**int lookup(char ch);**  
This line is a function named lookup of type int. It takes one parameter of type char named ch.

**#define LETTER 0;**  
This line is a #define directive that defines a constant named LETTER with a value of 0. It represents the category of alphabetic characters with the value 0.

**#define DIGIT 1;**  
This line defines a constant named DIGIT with a value of 1. It represents the category of digits (0-9) with the value 1, making the code clearer by using DIGIT instead of the number 1.

**#define UNKNOWN 99;**  
This line defines a constant named UNKNOWN with a value of 99. It represents any unknown character or symbol with the value 99.

**#define INT\_LIT 10;**  
This line defines a constant named INT\_LIT with a value of 10. It represents integer literals with the value 10.

**#define IDENT 11;**  
This line defines a constant named IDENT with a value of 11. It represents identifiers, such as variable names, with the value 11.

**#define ASSIGN\_OP 20;**  
This line defines a constant named ASSIGN\_OP with a value of 20. It represents the = symbol with the value 20.

**#define ADD\_OP 21;**  
This line defines a constant named ADD\_OP with a value of 21. It represents the addition operator (+) with the value 21.

**#define SUB\_OP 22;**  
This line defines a constant named SUB\_OP with a value of 22. It represents the subtraction operator (-) with the value 22.

**#define MULT\_OP 23;**  
This line defines a constant named MULT\_OP with a value of 23. It represents the multiplication operator (\*) with the value 23.

**#define DIV\_OP 24;**  
This line defines a constant named DIV\_OP with a value of 24. It represents the division operator (/) with the value 24.

**#define LEFT\_PAREN 25;**  
This line defines a constant named LEFT\_PAREN with a value of 25. It represents the left parenthesis ( with the value 25.

**#define RIGHT\_PAREN 26;**  
This line defines a constant named RIGHT\_PAREN with a value of 26. It represents the right parenthesis ) with the value 26.

**#define EOF\_TOKEN -1;**  
This line defines a constant named EOF\_TOKEN with a value of -1. It represents the end of input with the value -1.

**int main() {**  
This line defines the main function, which is the entry point of the program.

**do { lex(); } while (nextToken != EOF\_TOKEN);**  
Here, the code repeatedly calls the lex() function, each time reading a part of the expression and determining its type. The do-while loop continues until it reaches the EOF\_TOKEN, which means we've reached the end of the expression.

**return 0;**  
This line returns 0, indicating that the program executed successfully.

**cout << "Enter an expression: ";**  
The line prints a message to the user asking them to enter an expression.

**string input;**  
The line defines a variable input of type string to store the expression that the user will enter.

**getline(cin, input);**  
The line reads the entire line entered by the user and stores it in the input variable.

**in\_fp.clear();**  
The line clears any errors that might have occurred in in\_fp before.

**in\_fp.str(input);**  
The line puts the value of the input variable into the istringstream so that we can use it to read the data.

**getChar();**  
The line, when an expression is entered, takes the first character and stores it in nextChar so the code can start processing from that character.

**int lookup(char ch);**  
The line is a function named lookup that determines the type of the character so that the code can process it correctly.

**switch (ch)**  
The line switch (ch) tests the value of the variable ch, and determines what happens for each character.

**case '(': addChar(); nextToken = LEFT\_PAREN; break;**  
This line checks if the character entered is ( (left parenthesis). If it is, it adds ( to the string, sets the token type as LEFT\_PAREN, and then stops checking further cases.

**case ')': addChar(); nextToken = RIGHT\_PAREN; break;**  
This line checks if the entered character is ) (right parenthesis). If it is, it adds ) to the string, sets the token type as RIGHT\_PAREN, and then stops checking further cases.

**case '+': addChar(); nextToken = ADD\_OP; break;**  
This line checks if the entered character is + (addition), and in that case, it adds it to the string and sets the token type as "addition operation."

**case '-': addChar(); nextToken = SUB\_OP; break;**  
The line checks if the entered character is - (subtraction), and in that case, it adds it to the string and sets the token type as "subtraction operation."

**case '\*': addChar(); nextToken = MULT\_OP; break;**  
The line checks if the entered character is \* (multiplication), and in that case, it adds it to the string and sets the token type as "multiplication operation."

**case '/': addChar(); nextToken = DIV\_OP; break;**  
The line checks if the entered character is / (division), and in that case, it adds it to the string and sets the token type as "division operation."

**default: addChar(); nextToken = EOF\_TOKEN; break;**  
The line checks if the entered character is not one of the known operators or parentheses, and in that case, it adds it to the string and sets the token type as "end of file" (EOF\_TOKEN).

**return nextToken;**  
This line returns the current token value so the code can use it for further processing.

**void addChar();**  
This function is responsible for adding the current character and placing it into the word we're building (lexeme).

**if (lexLen < 98) {**  
In this case, we check if the length of the word we are building is less than 98 characters, because if we reach that limit, we won't be able to add more to the word.

**lexeme[lexLen++] = nextChar;**  
In this line, I begin to add the character.

**lexeme[lexLen] = '\0';**  
Here, we add \0, and this means that the word is finished.

**else { cout << "Error - lexeme is too long" << endl; }**  
If the word is longer than 98 characters, we print "Error - lexeme is too long."

**void getChar();**  
This function's job is to read the input and determine what it will be.

**if (in\_fp.get(nextChar)) {**  
Here, we read the next character from the input. in\_fp is an object that holds the input data. nextChar is a variable that will hold the character we read from the input.

**if (isalpha(nextChar)) charClass = LETTER;**  
Here, we check if the character we read from the input is a letter, and we set the variable charClass to LETTER.

**else if (isdigit(nextChar)) charClass = DIGIT;**  
Here, we check if the character we read from the input is a digit, and we set the variable charClass to DIGIT.

**else charClass = UNKNOWN;**  
Here, if the input is neither a letter nor a digit, we set the variable charClass to UNKNOWN.

**else { charClass = EOF\_TOKEN; nextChar = '\0'; }**  
Here, if the input has ended and there's nothing left, we set charClass to EOF\_TOKEN and set nextChar to \0 to indicate that there is no more input.

**void getNonBlank() {**

**while (isspace(nextChar))**

**getChar();**

**}**

"This function getNonBlank() is responsible for ensuring that we ignore spaces in the input text.

It uses isspace(nextChar) to check if the character we read is a space or not.

If it's a space, it calls getChar() to read the next character until it finds a non-space character."

**int lex() {**

"Here, we have a function called lex() of type int that determines the type of each part of the input, whether it's a letter, a digit, or something else."

**lexLen = 0;**

**getNonBlank();**

Here, we set the variable lexLen to zero because we are starting with the first character of the new word. Then, we call the function getNonBlank() to begin ignoring any spaces between characters in the input text.

**switch (charClass) {**

"Here, we use switch to decide how to proceed based on the type of input we read.

charClass determines the type of the input."

**case LETTER:**

**addChar();**

**getChar();**

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

**addChar();**

**getChar();**

**}**

**nextToken = IDENT;**

**break;**

**"**Here, we check if the input is a letter. If it is, we execute the following:

addChar(): We add the current character to the word we are building.

getChar(): We read the next character.

We repeat this process until we encounter a character that is neither a letter nor a digit.

Once we finish reading the word, we set the token type to IDENT

**case DIGIT:**

**addChar();**

**getChar();**

**while (charClass == DIGIT) {**

**addChar();**

**getChar();**

**}**

**nextToken = INT\_LIT;**

**break;**

"Here, if the character we read is a digit:

addChar(): We add the current digit to the number (or word) we are building.

getChar(): We read the next digit. We repeat this process until we encounter a character that is not a digit. Once we finish reading the number, we set the token type to INT\_LIT."

**case UNKNOWN:**

**lookup(nextChar);**

**getChar();**

**break;**

Here, if the input we read is neither a letter nor a digit:

We execute the function lookup(nextChar).

The lookup() function deals with special symbols or parentheses ( and ).

We read the next character using getChar() to continue analyzing the input."

**case EOF\_TOKEN:**

**nextToken = EOF\_TOKEN;**

**strcpy(lexeme, "EOF");**

**break;**

"Here, if the input has ended:

We set nextToken to EOF\_TOKEN to indicate that we have reached the end of the input.

We use strcpy(lexeme, "EOF") to write the word 'EOF' in lexeme."

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

**return nextToken;**

**}**

"Here, we print the current token type and lexeme to the screen to see what has been recognized.

Then, the function returns the identified token type so that other functions can use it."