A black background with grey leaves

AI-generated content may be incorrect.

Lexical Analyzer

Build Scanner

A grey logo on a black background

AI-generated content may be incorrect.

**Prepared By**

Student Name Student ID

**200042740**

**Saeed Mohamed saeed**

**2. Lexical Analyzer**

**#include <iostream>**

**#include <cctype>**

**#include <string>**

**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 LEFT\_PAREN 25**

**#define RIGHT\_PAREN 26**

**int charClass;**

**string lexeme;**

**char nextChar;**

**int nextToken;**

**string input = "saeed= 12 k + 7p \* (y / m - z)";**

**int index = 0;**

**void addChar();**

**void getChar();**

**void getNonBlank();**

**int lex();**

**int lookup(char ch);**

**int main() {**

**getChar();**

**do {**

**lex();**

**} while (nextToken != EOF);**

**return 0;**

**}**

**void addChar() {**

**lexeme += nextChar;**

**}**

**void getChar() {**

**if (index < input.length()) {**

**nextChar = input[index++];**

**if (isalpha(nextChar))**

**charClass = LETTER;**

**else if (isdigit(nextChar))**

**charClass = DIGIT;**

**else**

**charClass = UNKNOWN;**

**} else {**

**charClass = EOF;**

**}**

**}**

**void getNonBlank() {**

**while (isspace(nextChar))**

**getChar();**

**}**

**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:**

**lookup(nextChar);**

**getChar();**

**break;**

**case EOF:**

**nextToken = EOF;**

**lexeme = "EOF";**

**break;**

**}**

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

**return nextToken;**

**}**

**int lookup(char ch) {**

**switch (ch) {**

**case '=':**

**addChar();**

**nextToken = ASSIGN\_OP;**

**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 = LEFT\_PAREN;**

**break;**

**case ')':**

**addChar();**

**nextToken = RIGHT\_PAREN;**

**break;**

**default:**

**addChar();**

**nextToken = EOF;**

**break**

**::contentReference[oaicite:48]{index=48}**

**Header Files:**

**#include <iostream> // For input/output (e.g., cout)**

**#include <cctype> // For character classification functions like isalpha, isdigit, isspace**

**#include <string> // For using the string data type**

**Namespace:**

**using namespace std; // To avoid writing std:: before cout, string, etc.**

**Character Classes:**

**#define LETTER 0**

**#define DIGIT 1**

**#define UNKNOWN 99**

**These define categories for characters: letters, digits, and unknown symbols.**

**Token Types:**

**define INT\_LIT 10 // Integer literal**

**#define IDENT 11 // Identifier (e.g., variable names)**

**#define ASSIGN\_OP 20 // '=' operator**

**#define ADD\_OP 21 // '+' operator**

**#define SUB\_OP 22 // '-' operator**

**#define MULT\_OP 23 // '\*' operator**

**#define DIV\_OP 24 // '/' operator**

**#define LEFT\_PAREN 25 // '('**

**#define RIGHT\_PAREN 26 // ')'**

**These constants are token codes for different types of tokens.**

**Global Variables:**

**int charClass; // Holds the character class of the current character**

**string lexeme; // Stores characters forming the current token**

**char nextChar; // Current character being processed**

**int nextToken; // Holds the current token code**

**string input = "saeed= 12 k + 7p \* (y / m - z)"; // Input string to be analyzed**

**int index = 0; // Current index in the input string**

**Utility Functions:**

**void addChar() {**

**lexeme += nextChar;**

**}**

**Adds the current character to the lexeme.**

**void getChar() {**

**if (index < input.length()) {**

**nextChar = input[index++];**

**if (isalpha(nextChar))**

**charClass = LETTER;**

**else if (isdigit(nextChar))**

**charClass = DIGIT;**

**else**

**charClass = UNKNOWN;**

**} else {**

**charClass = EOF;**

**}**

**}**

**Reads the next character from the input string and sets its class (letter, digit, or unknown).**

**void getNonBlank() {**

**while (isspace(nextChar))**

**getChar();**

**}**

**Skips whitespace characters.**

**int lookup(char ch) {**

**switch (ch) {**

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

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

**default: addChar(); nextToken = EOF; break;**

**}**

**return nextToken;**

**}**

**Recognizes single-character tokens like operators and parentheses.**

**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:**

**lookup(nextChar); getChar();**

**break;**

**case EOF:**

**nextToken = EOF;**

**lexeme = "EOF";**

**break;**

**}**

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

**return nextToken;**

**}**

**This is the main lexical analyzer function. It reads the input string, classifies parts into tokens, and prints each token and its lexeme.**

**int main() {**

**getChar();**

**do {**

**lex();**

**} while (nextToken != EOF);**

**return 0;**

**}**

**Starts by reading the first character and then keeps calling lex() until the end of input is reached.**

**Output Explanation**

**saeed= 12 k + 7p \* (y / m - z)**

**Next token is: 11, Next lexeme is: saeed**

**Next token is: 20, Next lexeme is: =**

**Next token is: 10, Next lexeme is: 12**

**Next token is: 11, Next lexeme is: k**

**Next token is: 21, Next lexeme is: +**

**Next token is: 11, Next lexeme is: 7p**

**Next token is: 23, Next lexeme is: \***

**Next token is: 25, Next lexeme is: (**

**Next token is: 11, Next lexeme is: y**

**Next token is: 24, Next lexeme is: /**

**Next token is: 11, Next lexeme is: m**

**Next token is: 22, Next lexeme is: -**

**Next token is: 11, Next lexeme is: z**

**Next token is: 26, Next lexeme is: )**

**Note: "7p" is read as a single identifier instead of a number followed by a variable.**

**3. Software Tools**

**3.1. Computer Program : visual studio**

**3.2. Programming Language: c++**

**4. Implementation of a Lexical Analyzer**

**Lexeme and Token Table**

|  |  |  |
| --- | --- | --- |
| Lexeme | Token Code | Token Type |
| saeed | **11** | **Identifier** |
| = | **20** | **Assignment Operator** |
| 12 | **10** | **Integer Literal** |
| k | **11** | **Identifier** |
| + | **21** | **Addition Operator** |
| 7p | **11** | **Identifier (wrongly combined)** |
| \* | **23** | **Multiplication Operator** |
| ( | **25** | **Left Parenthesis** |
| y | **11** | **Identifier** |
| / | **24** | **Division Operator** |
| m | **11** | **Identifier** |
| - | **22** | **Subtraction Operator** |
| z | **11** | **Identifier** |
| ) | **26** | **Right Parenthesis** |

**5. References :**

**concept of programming languages**