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:Ruwayda mohamed

ID:200047927

**Under Supervision**

Name of Doctor:Nehal mohamed

Name of T. A.

1. **Introduction:**

A **lexical analyzer** also known as a scanner or lexer is the first phase of a compiler. It reads the source code and converts it into a stream of tokens, which are then used by the parser for syntax analysis.

* 1. **Phases of Compiler**

1. **Lexical Analyzer**

The lexical analyzer is the first phase of a compiler.  
Its job is to read the source code character by character and group them into **lexemes**, producing a sequence of **tokens** such as identifiers, numbers, and operators.  
These tokens are passed to the next phase of the compiler syntax analysis.

1. **Software Tools**
   1. **Computer Program : notepad , visual studio**
   2. **Programming Language:** C++
2. **Implementation of a Lexical Analyzer:** Identifier, Add Operator, Int Literal, getChar(),addChar(),lookup

**#include <iostream> // library for cout**

**#include <fstream> // library to read files**

**#include <cctype>// library to check characters**

**#include <string> // library for the string type**

**using namespace std; // it allows me to skipn writing std**

**/\* Character classes \*/**

**#define LETTER 0**

**#define DIGIT 1**

**#define UNKNOWN 99**

**#define EOF\_FLAG -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 variables \*/**

**int charClass;// stores the type of the current character**

**string lexeme;// holds the sequence of characters forming a word or number**

**char nextChar;// stores the next character read from the input file.**

**int token; // current token code identified by the lexical analyzer**

**int nextToken;// the next token to be processed**

**ifstream in\_fp;**

**/\* Function declarations \*/**

**void addChar();// adds the current character nextChar to the lexeme string**

**void getChar();// reads the next character from the input file and sets its character class**

|  |
| --- |
|  |

|  |
| --- |
|  |

**void getNonBlank();// a non-space character is found**

**int lex();**

**int lookup(char ch);**

**/\* addChar - adds nextChar to lexeme \*/**

**void addChar() {**

**if (lexeme.length() < 98) { / if lexeme is not long**

**lexeme += nextChar; // add the current character to the lexeme string**

**}**

**else { // If lexeme is too long, show error message**

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

**}**

**}**

**/\* getChar - reads the next character and sets charClass \*/**

**void getChar() {**

**if (in\_fp.get(nextChar)) { // read next character from file**

**if (isalpha(nextChar)) // If letter set the charclass= letter**

**charClass = LETTER; // if digit set the charclass = unknown**

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

**charClass = DIGIT;**

**else**

**charClass = UNKNOWN;**

**}**

**else {**

**charClass = EOF\_FLAG;**

**}**

**}**

**/\* getNonBlank - skips whitespace \*/**

**void getNonBlank() {**

**while (isspace(nextChar)) { // Check if current character is a space or tab**

**getChar();// until a non-space character is found**

**}**

**}**

**/\* lookup - recognizes operators and parentheses \*/**

**int lookup(char ch) {**

**switch (ch) { / uses switch to check character type**

**case '(':**

**addChar();// to store the character**

**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\_FLAG;**

**break;**

**}**

**return nextToken;**

**}**

**/\* lex - lexical analyzer for arithmetic expressions \*/**

**int lex() {**

**lexeme = "";// Clear previous lexeme before starting**

**getNonBlank();// skip**

**switch (charClass) { // Check the character class**

**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);// uses to detect operator or parenthesis**

**getChar();**

**break;**

**case EOF\_FLAG:**

**nextToken = EOF\_FLAG;**

**lexeme = "EOF";**

**break;**

**}**

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

**return nextToken;**

**}**

**/\* main driver \*/**

**int main() {**

**in\_fp.open("front.in");**

**if (!in\_fp) {**

**cout << "ERROR - cannot open front.in" << endl;**

**return 1;**

**}**

**else {**

**getChar();**

**do {**

**lex();**

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

**}**

**in\_fp.close();**

**return 0;**

**}**

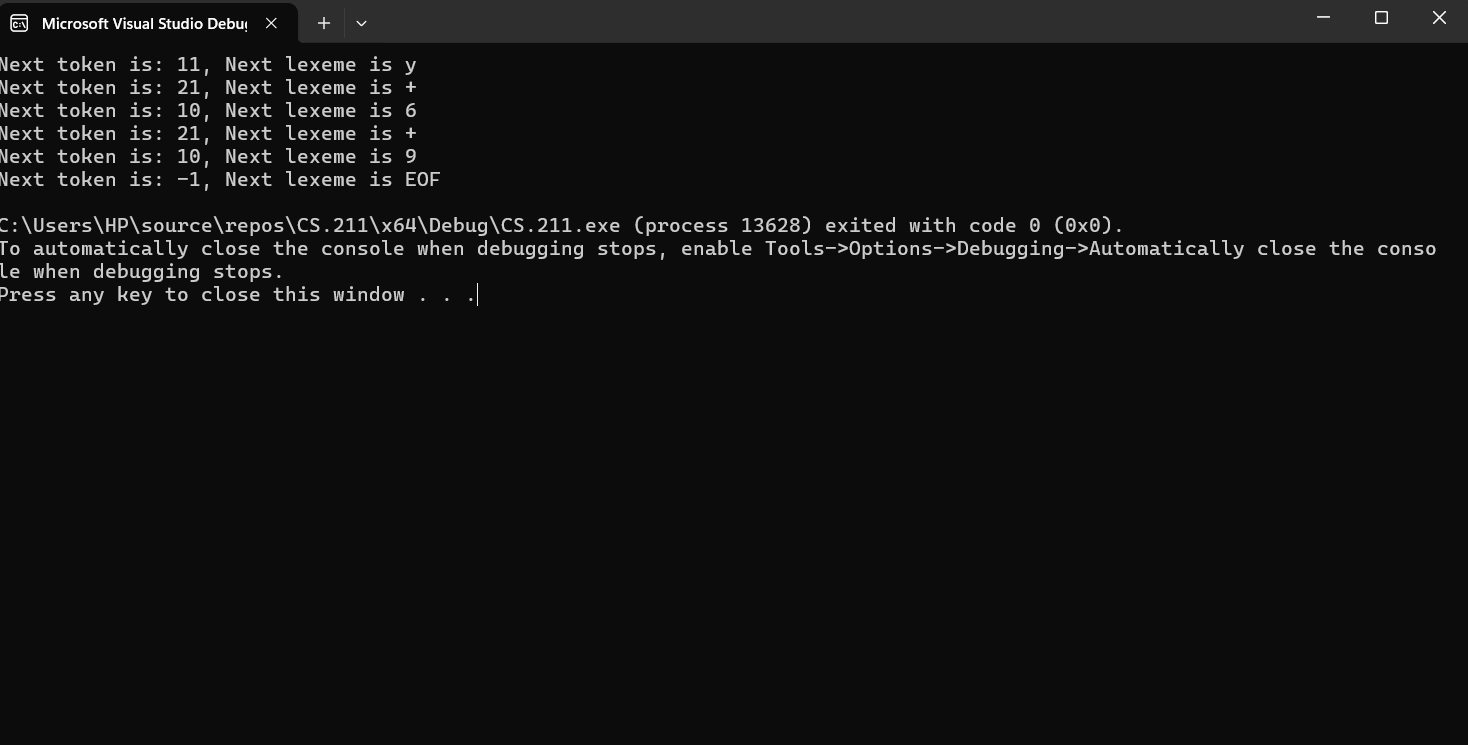
1. **References : course book**

**Important Note: -**

Technical reports include a mixture of text, tables, and figures. Consider how you can present the information best for your reader. Would a table or figure help to convey your ideas more effectively than a paragraph describing the same data?

Figures and tables should: -

* Be numbered
* Be referred to in-text, e.g. *In Table 1*…, and
* Include a simple descriptive label - above a table and below a figure.



|  |  |
| --- | --- |
| Token | Lexeme |
| IDENTIFIER | Y |
| ADD\_OP | + |
| INT\_LIT | 6 |
| ADD\_OP | + |
| INT\_LITRAL | 9 |