**Name: Niranjan Vinod Patil.**

**Batch: B31.**

**Roll No: SCETTYB305.**

**Course: Compiler Design.**

**----------------------------------------------------------------------------------------------------**

**Assignment No: 01**

**Aim:**

To write a C++ program to implement Lexical analyzer this recognizes keywords, operators & identifier.

**Theory:**

**Lexical analysis –**

Lexical analysis is the process of converting a sequence of characters from source program into a sequence of tokens.

**How it works:**

A program which performs lexical analysis is termed as a lexical analyzer (lexer), tokenizer or scanner.

Lexical analysis consists of two stages of processing which are as follows:

1. Scanning
2. Tokenization

## Token, Pattern and Lexeme

### Token

Token is a valid sequence of characters which are given by lexeme. In a programming language,

• keywords,

• constant,

• identifiers,

• numbers,

• operators and

• punctuations symbols

are possible tokens to be identified.

### Pattern

Pattern describes a rule that must be matched by sequence of characters (lexemes) to form a token. It can be defined by regular expressions or grammar rules.

### Lexeme

Lexeme is a sequence of characters that matches the pattern for a token i.e., instance of a token.

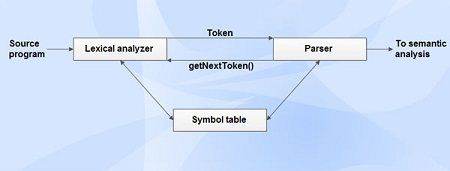
(eg.) c=a+b\*5;

**Lexemes and tokens**

|  |  |
| --- | --- |
| **Lexemes** | **Tokens** |
| C | Identifier |
| = | assignment symbol |
| A | Identifier |
| + | + (addition symbol) |
| B | Identifier |
| \* | \* (multiplication symbol) |
| 5 | 5 (number) |

The sequence of tokens produced by lexical analyzer helps the parser in analyzing the syntax of programming languages.

## Role of Lexical Analyzer

[](http://ecomputernotes.com/images/Interaction-between-lexical-analyzer-and-parser.jpg)

Lexical analyzer performs the following tasks:

1. Reads the source program, scans the input characters, group them into lexemes and produce the token as output.
2. Enters the identified token into the symbol table.
3. Strips out white spaces and comments from source program.
4. Correlates error messages with the source program i.e., displays error message with its occurrence by specifying the line number.
5. Expands the macros if it is found in the source program.

Tasks of lexical analyzer can be divided into two processes:

1. **Scanning:**Performs reading of input characters, removal of white spaces and comments.
2. **Lexical Analysis:**Produce tokens as the output.

## Need of Lexical Analyzer

**Simplicity of design of compiler:** The removal of white spaces and comments enables the syntax analyzer for efficient syntactic constructs.

**Compiler efficiency is improved:**Specialized buffering techniques for reading characters speed up the compiler process.

**Input to lexical analyzer: addition.txt File. This file contains**

void main()

{

int a, b, c;

c = a + b;

return 0;

}

**Output after lexical analysis:**

void is keyword

(is parenthesis

main is identifier

)is parenthesis

{is parenthesis

int is keyword

,is Seperator

a is identifier

,is Seperator

b is identifier

;is Seperator

c is identifier

c is identifier

=is Operator

a is identifier

+is Operator

;is Seperator

b is identifier

return is keyword

;is Seperator

0 is identifier

}is parenthesis

**Function used in program:**

1. **myfile.is\_open() -** An open file is represented within a program by a stream (i.e., an object of one of these classes; in the previous example, this was myfile) and any input or output operation performed on this stream object will be applied to the physical file associated to it.
2. **fstream:** Stream class to both read and write from/to files.
3. **myfile.eof() :** Returns true if a file open for reading has reached the end.
4. **myfile.get()**: The get() function is member of ifstream class. It is used to read character form the file.
5. **isalnum(ch):** checks whether the given character is alphanumeric or not. isalnum() function defined in ctype.h header file.

**Algorithm:**

**Step 1: Start**

**Step 2:** Open that file to recognized identifier, keyword, operator and identifier.

**Step 3:** Check that file is open or not. If file is open then Repeat step 4 and 5 until end of file else print unable to open file.

**Step 4:** Get each character from file and compare with operator, separator and parenthesis list. If it is matched then print character is operator or separator or parenthesis.

**Step 5:** if character is alphanumeric then until some operator, comma, semicolon all character is stored in buffer and buffer compared with keyword list. If it is matched then print character is keyword else character is identifier.

**Step 6:** Stop

**Pseudo code:**

int isKeyword(char buffer[]){

charkeywords[32][10] = {"auto","break","case","char","const","continue","default",

"do","double","else","enum","extern","float","for","goto",

"if","int","long","register","return","short","signed",

"sizeof","static","struct","switch","typedef","union",

"unsigned","void","volatile","while"};

int i, flag = 0;

for(i = 0; i < 32; ++i){

if(strcmp(keywords[i], buffer) == 0){

flag = 1;

break;

}

}

return flag;

}

int main ()

{

char ch, buffer[15];

char arithmetic\_operators[] = {"+-\*/%="};

char parenthesis[] = {"(){}"};

char separators[] = {",;"};

int i,j=0;

ifstream myfile ("addition.txt");

if (myfile.is\_open())

{

while (!myfile.eof() )

{

ch = myfile.get();

for(i = 0;i < 6; i++)

{

if(ch == arithmetic\_operators[i])

{

cout <<ch<<"is Operator"<<'\n';

}

}

for(int i = 0; i<3; i++)

{

if(ch == separators[i])

{

cout <<ch<<"is Seperator"<<'\n';

}

}

for(int i = 0; i<5; i++)

{

if(ch == parenthesis[i])

{

cout <<ch<<"is parenthesis"<<'\n';

}

}

if(isalnum(ch)){

buffer[j++] = ch;

}

else if((ch == ' ' || ch == '\n' || ch ==',' || ch == ';'|| ch == '(' || ch == ')' || ch == '{' || ch =='}') && (j != 0))

{

buffer[j] = '\0';

j = 0;

if(isKeyword(buffer) == 1)

cout<<buffer<<" is keyword\n";

else

cout<<buffer<<" is identifier\n";

}

}

}

else

{

cout << "Unable to open file";

}

return 0;

}