

ABHINAV RANJAN  
RA1911003010003  
CSE A1 SECTION  
SRMIST , KTR

## **COMPILER DESIGN LAB**

### ***EXP 1 - LEXICAL ANALYZER***

#### **AIM :**

To write a C++ program to show the implementation of a lexical analyzer

#### **REQUIREMENTS :**

1. Knowledge of the working of a lexical analyzer
2. Knowledge of the concept of tokens - identifiers , operators , keywords , digits , alphanumeric , etc
3. Online compiler GDB or any compiler like Dev C++

#### **THEORY :**

##### **LEXICAL ANALYSIS :**

The Lexical analyzer phase is the first phase of the compilation process. It takes source code as input. It reads the source program one character at a time and converts it into meaningful lexemes. Lexical analyzer represents these lexemes in the form of tokens.

## TOKENS :

Lexemes are said to be a sequence of characters (alphanumeric) in a token. There are some predefined rules for every lexeme to be identified as a valid token. These rules are defined by grammar rules, by means of a pattern. A pattern explains what can be a token, and these patterns are defined by means of regular expressions.

In programming language, keywords, constants, identifiers, strings, numbers, operators and punctuation symbols can be considered as tokens.

## ALGORITHM :

**STEP 1 :** To take an input string and use stringstream to read the given input word by word

**STEP 2 :** The tokens are to be identified so a while loop is created in which there are conditions to identify different types of input such as keywords , operators , identifiers ,etc.

**STEP 3 :** Once the given input is scanned character by character and sorted into different categories , the output window has to display the details.

**STEP 4 :** The output console shows the details of each input character - whether they are operators or identifiers , and hence the work of the lexical analyzer is done.

## SOURCE CODE :

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    string s;
    cout<< "Enter an input " <<endl;
    getline(cin,s);
    cout<<"The lexical analysis of the given input is :"<<endl;
    stringstream str(s);
    string ch;
    while(str>>ch)
    {
        if(ch == "if" || ch == "else" ||
        ch == "while" || ch == "do" ||
        ch == "break" || ch == "continue"
        || ch == "int" || ch == "double"
        || ch == "float" || ch == "return"
        || ch == "char" || ch == "case"
        || ch == "long" || ch == "short"
        || ch == "typedef" || ch == "switch"
        || ch == "unsigned" || ch == "void"
        || ch == "static" || ch == "struct"
        || ch == "sizeof" || ch == "long"
        || ch == "volatile" || ch == "typedef"
        || ch == "enum" || ch == "const"
        || ch == "union" || ch == "extern"
        || ch == "bool")
        {
            cout<<ch<<" is a keyword"<<endl;
        }
        else if(ch[0] == ' ' || ch[0] == '+' || ch[0] == '-' || ch[0] == '*' ||
```

```

    ch[0] == '/' || ch[0] == '>' || ch[0] == '<' || ch[0] == '=' || ch[0] == '(' ||
ch[0] == ')' ||
    ch[0] == '[' || ch[0] == ']' || ch[0] == '{' || ch[0] == '}' ||
    ch[0] == '&' || ch[0] == '|')
{
    cout<<ch<<" is an operator"<<endl;
}
else if(ch[0] == ',' || ch[0] == ';' || ch[0] == '.')
{
    cout<<ch<<" is a separator"<<endl;
}
else if(isdigit(ch[0]))
{
    cout<<ch<<" is a constant"<<endl;
}
else
{
    cout<<ch<<" is an identifier"<<endl;
}
}
return 0;
}

```

## SCREENSHOT OF OUTPUT :

The screenshot shows the Programiz C++ Online Compiler interface. The code editor on the left contains a C++ program that performs lexical analysis on the input string "int y = z + x - 4;". The program identifies keywords, identifiers, operators, separators, and constants. The output window on the right displays the results of the analysis.

**Programiz**  
C++ Online Compiler

**LOOKING TO LEARN PROGRAMMING?**  
Start your programming journey with Programiz **AT NO COST.**

[Learn More](#)

[Learn Python App](#)

**main.cpp**

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int main()
4 {
5     string s;
6     cout<<"Enter an input " <<endl;
7     getline(cin,s);
8     cout<<"The lexical analysis of the given input is : "<<endl;
9     stringstream str(s);
10    string ch;
11    while(str>>ch)
12    {
13        if(ch == "if" || ch == "else" ||
14        ch == "while" || ch == "do" ||
15        ch == "break" || ch == "continue"
16        || ch == "int" || ch == "double"
17        || ch == "float" || ch == "return"
18        || ch == "char" || ch == "case"
19        || ch == "long" || ch == "short"
20        || ch == "typedef" || ch == "switch"
21        || ch == "unsigned" || ch == "void"
22        || ch == "static" || ch == "struct"
23        || ch == "sizeof" || ch == "long"
24        || ch == "volatile" || ch == "typedef"
25        || ch == "enum" || ch == "const"

```

**Output**

```

/tmp/RvOpUj/w01V.o
Enter an input
int y = z + x - 4;
The lexical analysis of the given input is :
int is a keyword
y is an identifier
= is an operator
z is an identifier
+ is an operator
x is an identifier
- is an operator
4; is a constant

```

Output Clear

```
/tmp/RvOpUjWo1V.o
Enter an input
int y = z + x - 4;
The lexical analysis of the given input is :
int is a keyword
y is an identifier
= is an operator
z is an identifier
+ is an operator
x is an identifier
- is an operator
4; is a constant
```

## **OBSERVATION :**

Thus all the given inputs were scanned character by character and were segregated into their respective categories (like operators , identifiers , etc) which were shown in the output.

## **RESULT :**

Thus we have successfully implemented a C++ program for lexical analyzer where each input character has been put in its respective category of tokens.