**SRM INSTITUTE OF SCIENCE & TECHNOLOGY DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

Logo, company name

Description automatically generated

|  |
| --- |
| **TOPIC: - LEXICAL ANALYZER** |

|  |  |
| --- | --- |
| **Department** | **Computer science** |
| **Subject** | **Compiler Design** |
| **Semester** | **6** |
| **Faculty** | **Sentamarai N** |
| **Submitted by** | **KARAN KHANDELWAL(RA1911031010024)**  **AKASH TIWARI(RA1911031010051)**  **DEV SHANKAR**  **TRIPATHI(RA1911031010048)** |

**INTRODUCTION**

**Lexical Analysis is the first phase of the compiler also known as a scanner. It converts the High level input program into a sequence of Tokens. it helps you to convert a sequence of characters into a sequence of tokens. The lexical analyzer breaks this syntax into a series of tokens. It removes any extra space or comment written in the source code.**

**Programs that perform Lexical Analysis in compiler design are called lexical analyzers or lexers. A lexer contains tokenizer or scanner. If the lexical analyzer detects that the token is invalid, it generates an error. The role of Lexical Analyzer in compiler design is to read character streams from the source code, check for legal tokens, and pass the data to the syntax analyzer when it demands. Lexical Analysis can be implemented with the Deterministic finite Automata. The output is a sequence of tokens that is sent to the parser for syntax analysis. For this project python language was used.**

**WHAT IS TOKEN?**

* **Token is a valid sequence of characters that can be treated as single logical entity.**

**Tokens are :**

* **Keywords**
* **Constant**
* **Identifiers**
* **Operators**
* **Numbers**
* **Punctuation**

**Example of tokens:**

**Keywords; Examples-for, while, if etc.**

**Identifier; Examples-Variable name, function name, etc.**

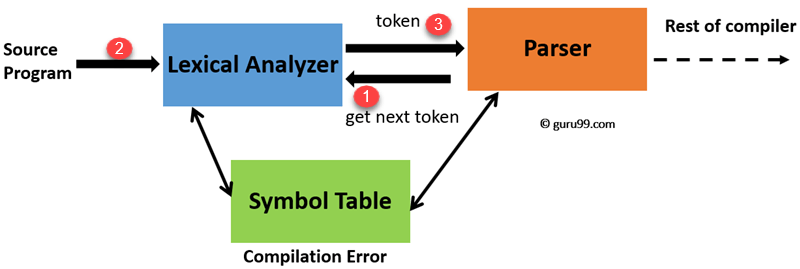
**Operators; Examples '+', '++', '-' etc.**

**Separators; Examples ',' ';' etc**

**LEXICAL ANALYZER ARCHITECTURE: HOW TOKENS ARE RECOGNIZED**

* **The main task of lexical analysis is to read input characters in the code and produce tokens.**
* **Lexical analyzer scans the entire source code of the program. It identifies each token one by one. Scanners are usually implemented to produce tokens only when requested by a parser. Here is how recognition of tokens in compiler design works-**
* **“Get next token” is a command which is sent from the parser to the lexical analyzer.**
* **On receiving this command, the lexical analyzer scans the input until it finds the next token.**
* **It returns the token to Parser.**

**Lexical Analyzer skips whitespaces and comments while creating these tokens. If any error is present, then Lexical analyzer will correlate that error with the source file and line number.**



**ROLES OF THE LEXICAL ANALYZER**

**Lexical analyzer performs below given tasks:**

* **Helps to identify token into the symbol table**
* **Removes white spaces and comments from the source program**
* **Correlates error messages with the source program**
* **Helps you to expands the macros if it is found in the source program**
* **Read input characters from the source program**

**FRONTEND CODE**

from tkinter import \*

import tkinter.messagebox

from backend import \*

root = Tk()

root.title("Lexical Analyzer")

root.geometry("1350x750+0+0")

root.config(bg="black")

Input = StringVar()

txtInput = StringVar()

# Fuctions

def iExit():

    iExit = tkinter.messagebox.askyesno("Lexical Analyzer", "Are you sure???")

    if iExit > 0:

        root.destroy()

    return

def clcdata():

    txtInput.delete(0, END)

def adddata():

    inp = txtInput.get()

    ans = parsefn(inp)

    lbl.config(text="Result : " + ans)

# Frames

MainFrame = Frame(root, bg="black")

MainFrame.grid()

TFrame = Frame(MainFrame, bd=5, padx=54, pady=8, bg="black", relief=RIDGE)

TFrame.pack(side=TOP)

TFrame = Label(TFrame, font=('Arial', 51, 'bold'), text="Lexical Analyzer", bg="black", fg="orange")

TFrame.grid()

BFrame = Frame(MainFrame, bd=2, width=1350, height=70, padx=18, pady=10, bg="black", relief=RIDGE)

BFrame.pack(side=BOTTOM)

DFrame = Frame(MainFrame, bd=2, width=1300, height=400, padx=20, pady=20, bg="black", relief=RIDGE)

DFrame.pack(side=BOTTOM)

DFrameL = LabelFrame(DFrame, bd=2, width=1000, height=600, padx=20, bg="black", relief=RIDGE,

                     font=('Arial', 20, 'bold'), text="Input Info\n", fg="white")

DFrameL.pack(side=LEFT)

DFrameR = LabelFrame(DFrame, bd=2, width=450, height=300, padx=31, pady=3, bg="black", relief=RIDGE,

                     font=('Arial', 20, 'bold'), text="Output\n", fg="white")

DFrameR.pack(side=RIGHT)

# Labels & Entry Box

lblInput = Label(DFrameL, font=('Arial', 18, 'bold'), text="Your Info :", padx=2, pady=2, bg="black", fg="orange")

lblInput.grid(row=0, column=0, sticky=W)

txtInput = Entry(DFrameL, font=('Arial', 18, 'bold'), textvariable=Input, width=39, bg="black", fg="white")

txtInput.grid(row=0, column=1)

# output kabel

lbl = Label(DFrameR, text="")

lbl.pack()

# Buttons

btnadd = Button(BFrame, text="Evaluate", font=('Arial', 20, 'bold'), width=10, height=1, bd=4, bg="orange",

                command=adddata)

btnadd.grid(row=0, column=0)

btnadd.pack()

if \_\_name\_\_ == '\_\_main\_\_':

    root = Tk()

    root.mainloop()

**BACKEND CODE**

lines = ""

keywords = ["void", "main", "int", "float", "bool", "if", "for", "else", "while", "char", "return"]

operators = ["=", "==", "+", "-", "\*", "/", "++", "--", "+=", "-=", "!=", "||", "&&"]

punctuations = [";", "(", ")", "{", "}", "[", "]"]

def is\_int(x):

    try:

        int(x)

        return True

    except:

        return False

def parsefn(b):

    a = ""

    for line in b:

        for i in line.strip().split(" "):

            a += "\n"

            if i in keywords:

                a = a + " " + i + " is a keyword"

            elif i in operators:

                a = a + " " + i + " is an operator"

            elif i in punctuations:

                a = a + " " + i + " is a punctuation"

            elif is\_int(i):

                a = a + " " + i + " is a number"

            else:

                a = a + " " + i + " is an identifier"

    return (a)

**OUTPUT**

