MISR UNIVERSITY
FOR SCIENCE & TECHNOLOGY
College of Information Technology

جـامعـة مـصـر
للعلـوم والتكنـولـوجيـا
كليـة تكنولوجيـا المعلومـات

LEXICAL ANALYZER

Build Scanner

Prepared By

Student Name:

Ahmed Elsayed

Student ID:

200042804

Under Supervision

Name of Doctor

Name of T. A.

- ## Introduction

This document provides an overview of the implementation of a Lexical which is a ,Analyzer
the ,It covers the phases of a compiler .fundamental phase in compiler design role of a lexical
analyzer, software tools used, and the implementation details

- ### Phases of Compiler

including: ,A compiler consists of several phases
Tokenizing the input code. :Lexical Analysis .1
Checking grammatical structure. :Syntax Analysis .2
Ensuring meaningful statements. :Semantic Analysis .3
Creating an intermediate representation. :Intermediate Code Generation .4
Improving performance and efficiency. :Optimization .5
.Producing machine code :Code Generation .6

- ## Lexical Analyzer

A Lexical Analyzer is responsible for scanning the source code and converting it into tokens.
It identifies keywords, operators, identifiers, and other elements

- ## Software Tools

Various software tools are used in compiler construction.

- ### Computer Program

It .A compiler is a special type of program that translates source code into machine code ensures the correctness of syntax and semanti

- **Programming Language**

Lexical analyzers are often implemented using programming languages like Python, C, or Java. The implementation in this document is in Python.

- **Implementation of a Lexical Analyzer**

Below is the Python implementation of a lexical analyzer:

```
import string

# Character classes
LETTER = 0
DIGIT = 1
UNKNOWN = 99
EOF = -1

# Token codes
INT_LIT = 10
IDENT = 11
ASSIGN_OP = 20
ADD_OP = 21
SUB_OP = 22
```

```
MULT_OP = 23
DIV_OP = 24
LEFT_PAREN = 25
RIGHT_PAREN = 26


class LexicalAnalyzer:
    def __init__(self, input_string):
        self.input_string = input_string
        self.index = 0
        self.char_class = None
        self.lexeme = ""
        self.next_char = ""
        self.next_token = None
        self.get_char()

    def add_char(self):
        self.lexeme += self.next_char

    def get_char(self):
        if self.index < len(self.input_string):
```

```
            self.next_char = self.input_string[self.index]
            self.index += 1
            if self.next_char.isalpha():
                self.char_class = LETTER
            elif self.next_char.isdigit():
                self.char_class = DIGIT
            else:
                self.char_class = UNKNOWN
        else:
            self.char_class = EOF


    def get_non_blank(self):
        while self.next_char.isspace():
            self.get_char()


    def lookup(self, char):
        if char == '(': return LEFT_PAREN
        if char == ')': return RIGHT_PAREN
        if char == '+': return ADD_OP
        if char == '-': return SUB_OP
```

```
    if char == '*': return MULT_OP
    if char == '/': return DIV_OP
    return EOF

def lex(self):
    self.lexeme = ""
    self.get_non_blank()
    if self.char_class == LETTER:
        self.add_char()
        self.get_char()
        while self.char_class in {LETTER, DIGIT}:
            self.add_char()
            self.get_char()
        self.next_token = IDENT
    elif self.char_class == DIGIT:
        self.add_char()
        self.get_char()
        while self.char_class == DIGIT:
            self.add_char()
            self.get_char()
```

```
            self.next_token = INT_LIT
        elif self.char_class == UNKNOWN:
            self.next_token = self.lookup(self.next_char)
            self.add_char()
            self.get_char()
        elif self.char_class == EOF:
            self.next_token = EOF
            self.lexeme = "EOF"
        print(f"Next token is: {self.next_token}, Next lexeme is
{self.lexeme}")
        return self.next_token


# Example usage
expression = "(sum + 47) / total"
lexer = LexicalAnalyzer(expression)
while lexer.next_token != EOF:
    lexer.lex()
```

code description: -

Token Types

The lexer recognizes the following token types:

Identifiers: Represented by IDENT (code: 11)

Integer Literals: Represented by INT_LIT (code: 10)

Operators:

Assignment Operator: ASSIGN_OP (code: 20)

Addition Operator: ADD_OP (code: 21)

Subtraction Operator: SUB_OP (code: 22)

Multiplication Operator: MULT_OP (code: 23)

Division Operator: DIV_OP (code: 24)

Parentheses:

Left Parenthesis: LEFT_PAREN (code: 25)

Right Parenthesis: RIGHT_PAREN (code: 26)

End of File: Represented by EOF (code: -1)

Class Documentation

LexicalAnalyzer

The LexicalAnalyzer class is responsible for tokenizing the input string.

Constructor

def __init__(self, input_string: str)

Parameters:

input_string: A string containing the expression to be tokenized.

Methods

add_char()

Appends the current character to the lexeme.

get_char()

Reads the next character from the input string and updates the character class.

get_non_blank()

Skips any whitespace characters in the input string.

lookup(char: str) -> int

Looks up the token code for a given character.

Parameters:

char: A single character to look up.

Returns: The token code corresponding to the character.

lex() -> int

Analyzes the input string and returns the next token.

Returns: The token code of the next token.

Prints: The next token and its corresponding lexeme.

Example Usage

To see the lexer in action, you can run the provided example code. It will tokenize the expression (sum + 47) / total and print each token along with its lexeme.

- ## References

Create Your Own Domain-Specific :Language Implementation Patterns **.(2022) .T ,Parr.1**
and
***General Programming Languages with Python*.**
.Introduction to Compiler Design **.(2021) .D ,Parsons.2**

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.

Al-Motamayez District 6th of October, P.O Box 77, Giza, Egypt.
+(202) 38247455 / 6 / 7 +(202) 38247417 / 38247428 16878
info@must.edu.eg www.must.edu.eg