MISR UNIVERSITY
FOR SCIENCE & TECHNOLOGY
College of Information Technology

جـامعـة مـصـر
للعلـوم والتكنـولـوجيـا
كليـة تكنولوجيـا المعلومـات

LEXICAL ANALYZER

Build Scanner

Prepared By
Student Name
Soad Mohamed ali ali
Student ID
200039167
Under Supervision.
Name of Doctor
Nehal Abdel Salam
Name of T.A
Fares Imad Al_Din.

1. Introduction

1.1. phases of Compiler

1. Lexical Analysis (Scanner)

- Your code implements this phase
- Responsibilities:
  - Reads source code character by character
  - Groups characters into tokens (lexemes)
  - Identifies token types using regular patterns
  - Removes whitespace/comments
  - Handles simple errors (like invalid characters)

2. Syntax Analysis (Parser)

- Not implemented in your code
- Would use your tokens to build parse trees
- Checks grammar/structure against language rules

3. Semantic Analysis

- Not implemented
- Performs type checking
- Verifies variable declarations
- Ensures semantic correctness

4. Intermediate Code Generation

- Not implemented
- Produces abstract machine code (e.g., three-address code)

5. Optimization

- Not implemented

- Improves intermediate code efficiency

6. Code Generation

- Not implemented
- Produces target machine code

1.2. P

2. Lexical Analyzer

Breaks lexem into token which is machine language

3. Software Tools

3.1. Computer Program

Xcode

3.2. Programming Language

C++

4. Implementation of a Lexical Analyzer

```
#include <iostream>     // library for input and output
#include <fstream>      // library for file input and
output
#include <cctype>       // library for character
classification
#include <string>       // library for string class
#include <unistd.h>     // for realpath
    #include <limits.h>     // for PATH_MAX

    // Character classes
    #define LETTER 0        // for letter characters
    #define DIGIT 1         // for digit characters
    #define UNKNOWN 99      // for unknown characters
    #define END_OF_FILE -1  // for end of file marker

    // Token codes
```

```
19.     #define INT_LIT 10      // integer literal
20.     #define IDENT 11        // identifier
21.     #define ASSIGN_OP 20    // assignment operator
22.     #define ADD_OP 21       // addition operator
23.     #define SUB_OP 22       // subtraction operator
24.     #define MULT_OP 23      // multiplication operator
25.     #define DIV_OP 24       // division operator
26.     #define LEFT_PAREN 25   // left parenthesis
27.     #define RIGHT_PAREN 26  // right parenthesis
28.
29.     using namespace std;    // using standard namespace
30.
31.     // Global declarations
32.     int charClass;          // current character class
33.     string lexeme;          // current lexeme being
   built
34.     char nextChar;          // next character in input
35.     int nextToken;          // code for next token
36.     ifstream inFile;        // input file stream
37.
38.     // Function to add character to lexeme
39.     void addChar() {
40.         lexeme += nextChar; // append character to
   lexeme
41.     }
42.
43.     // Function to get next character from input
44.     void getChar() {
45.         if (inFile.get(nextChar)) {  // if successfully
   read character
46.             if (isalpha(nextChar))  // if character is
   letter
47.                 charClass = LETTER;
48.             else if (isdigit(nextChar)) // if character
   is digit
49.                 charClass = DIGIT;
50.             else                        // otherwise
   unknown
51.                 charClass = UNKNOWN;
52.         } else {                        // if end of file
53.             charClass = END_OF_FILE;
54.         }
55.     }
```

```
56.
57.     // Function to skip whitespace
58.     void getNonBlank() {
59.         while (isspace(nextChar)) { // while character
   is whitespace
60.             getChar();              // get next
   character
61.         }
62.     }
63.
64.     // Function to lookup operators
65.     int lookup(char ch) {
66.         switch (ch) {               // check character
67.             case '(':               // left parenthesis
68.                 addChar();
69.                 nextToken = LEFT_PAREN;
70.                 break;
71.             case ')':               // right parenthesis
72.                 addChar();
73.                 nextToken = RIGHT_PAREN;
74.                 break;
75.             case '+':               // addition operator
76.                 addChar();
77.                 nextToken = ADD_OP;
78.                 break;
79.             case '-':               // subtraction
   operator
80.                 addChar();
81.                 nextToken = SUB_OP;
82.                 break;
83.             case '*':               // multiplication
   operator
84.                 addChar();
85.                 nextToken = MULT_OP;
86.                 break;
87.             case '/':               // division operator
88.                 addChar();
89.                 nextToken = DIV_OP;
90.                 break;
91.             case '=':               // assignment
   operator
92.                 addChar();
93.                 nextToken = ASSIGN_OP;
```

```
94.                 break;
95.             default:                // unknown operator
96.                 addChar();
97.                 nextToken = END_OF_FILE;
98.                 break;
99.         }
100.        return nextToken;
101.    }
102.
103.    // Lexical analyzer function
104.    int lex() {
105.        lexeme = "";                // initialize lexeme
106.        getNonBlank();              // skip whitespace
107.
108.        switch (charClass) {        // based on character
    class
109.            case LETTER:            // if letter
110.                addChar();         //call the function to
    add it to the lexeme
111.                getChar();        // call the function to
    get the next char
112.                while (charClass == LETTER || charClass
    == DIGIT) { // while letter or digit
113.                    addChar();  //call the function to
    add it to the lexeme
114.                    getChar();  // call the function to
    get the next char
115.                }
116.                nextToken = IDENT; // set token to
    identifier
117.                break;
118.
119.            case DIGIT:             // if digit
120.                addChar();          //call the function
    to add it to the lexeme
121.                getChar();          // call the function
    to get the next char
122.                while (charClass == DIGIT) { // while
    more digits
123.                    addChar();     //call the function
    to add it to the lexeme
124.                    getChar();     // call the function
    to get the next char
```

```
125.                 }
126.                 nextToken = INT_LIT; // set token to
   integer literal
127.                 break;
128.
129.             case UNKNOWN:          // if unknown
130.                 lookup(nextChar); // lookup operator
131.                 getChar();
132.                 break;
133.
134.             case END_OF_FILE:      // if end of file
135.                 nextToken = END_OF_FILE;
136.                 lexeme = "EOF";
137.                 break;
138.         }
139.
140.         // Print token information
141.         cout << "Next token is: " << nextToken << ",
   Next lexeme is: " << lexeme << endl;
142.         return nextToken;
143.     }
144.
145.     // Main function
146.     int main() {
147.         // Open input file - CHANGED THIS LINE TO FIX
   THE ERROR
148.
   inFile.open("/Users/macbookpro/Downloads/front.in"); //
   absolute path to input file
149.
150.         if (!inFile.is_open()) {  // if file didn't
   open
151.             cout << "ERROR - cannot open front.in" <<
   endl; // print error
152.             return 1;             // return error code
153.         }
154.
155.         getChar();               // read first
   character
156.
157.         do {                     // loop until end of
   file
```

```
158.            lex();                  // call lexical
   analyzer
159.        } while (nextToken != END_OF_FILE);
160.
161.        inFile.close();             // close input file
162.        return 0;                   // return success
163.    }
```

164.References

Concepts of programming languages book

Youtube

Chatgbt

Deebseak

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?

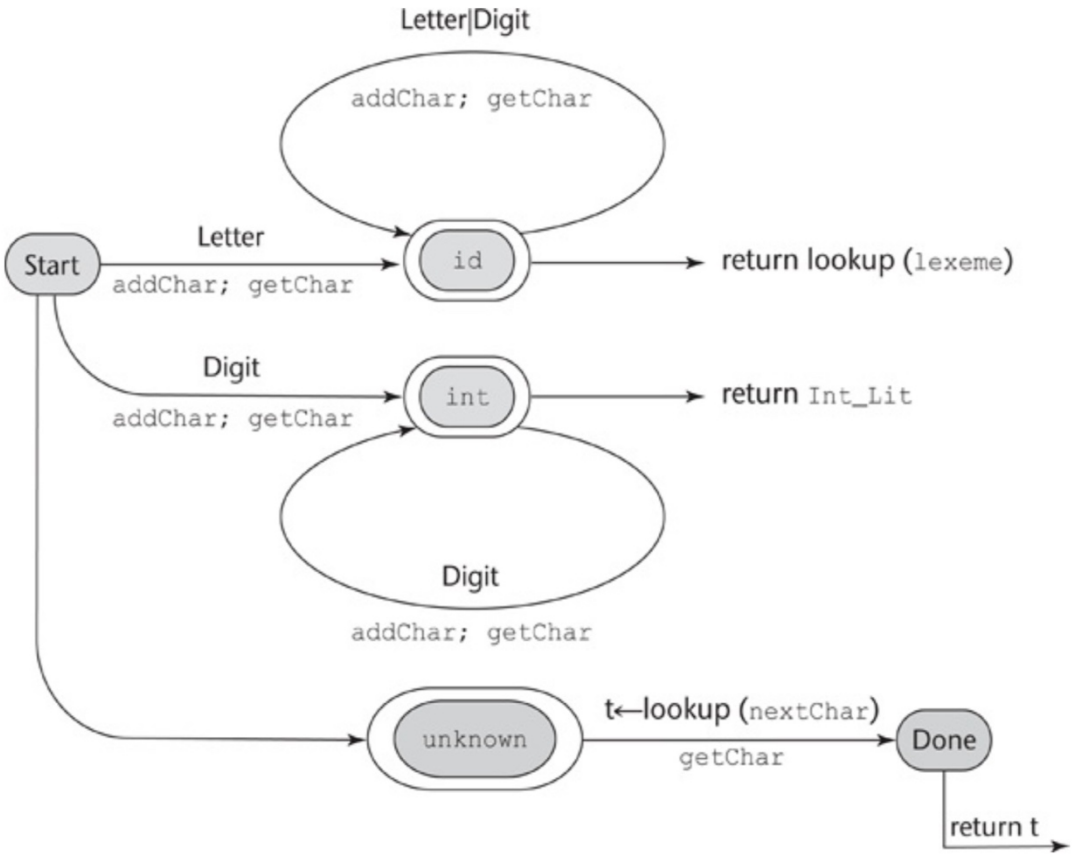
Figures and tables should: -

- 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

(sum + 47) / total

TOKEN	LEXEME
LEFT_PAREN	(
IDENT	Sum
ADD_OP	+
INT_LIT	47
RIGHT_PAREN	)
DIV_OP	/
IDENT	total

```
Next token is: 25, Next lexeme is: (
Next token is: 11, Next lexeme is: sum
Next token is: 21, Next lexeme is: +
Next token is: 10, Next lexeme is: 47
Next token is: 26, Next lexeme is: )
Next token is: 24, Next lexeme is: /
Next token is: 11, Next lexeme is: total
```


Letter|Digit
addChar; getChar
Start
Letter
addChar; getChar
id
return lookup (lexeme)
Digit
addChar; getChar
int
return Int_Lit
Digit
addChar; getChar
unknown
t←lookup (nextChar)
getChar
Done
return t