MISR UNIVERSITY
FOR SCIENCE & TECHNOLOGY
College of Information Technology

جـامعـة مـصـر
للعلـوم والتكنـولـوجيـا
كليـة تكنولوجيـا المعلومـات

LEXICAL ANALYZER

Build Scanner

Prepared By
Arwa Ahmed Mohamed Lasheen
200043986
Under Supervision
DR/ Nehal Abdelsalam
ENG/ Fares Imad Al_Din

1. Introduction

1.1. Phases of Compiler

phases:

A compiler operates in multiple phases, each responsible for a different aspect of translation.

Phase	Description
Lexical Analysis	Converts source code into tokens.
Syntax Analysis	Checks grammatical correctness and builds a parse tree.
Semantic Analysis	Ensures valid meaning and detects type errors.
Intermediate Code Generation	Produces an intermediate representation.
Code Optimization	Improves performance by reducing redundancies.
Code Generation	Converts optimized code to machine code.
Error Handling	Identifies and reports errors.

2. Lexical Analyzer

reads the source code character by character and groups them into meaningful tokens.

3. Software Tools

3.1. Computer Program

XCode

3.2. Programming Language

C

4. Implementation of a Lexical Analyzer

```c
#include <stdio.h> //For input/output functions
#include <ctype.h>//character classification functions
#include <string.h>//string manipulation functions

#define LETTER 0 //representing a letter character.
 #define DIGIT 1 //representing a digit character.
 #define UNKNOWN 99 //used for unknown characters.
 #define END_OF_FILE -1 //used to mark the end of the file
 #define INT_LIT 10 //used for integer literals.
 #define IDENT 11 //representing identifiers
 #define ASSIGN_OP 20 //representing the assignment operator
 #define ADD_OP 21//representing the addition operator
 #define SUB_OP 22 //representing the subtraction operator
 #define MULT_OP 23//representing the multiblication operator
 #define DIV_OP 24//representing the division operator
 #define LEFT_PAREN 25//representing the left parenthesis
 #define RIGHT_PAREN 26// representing the right parenthesis

 int charClass; // Stores character type
 char lexeme[100]; // Stores the current lexeme
 char nextChar; // Stores the next character
 int nextToken; // Stores the next token type
 FILE *inFile; // File pointer for input file handling

 void addChar() {
    strncat(lexeme, &nextChar, 1);// Appends nextChar to the lexeme string
 }

 void getChar() {
    nextChar = fgetc(inFile); // Reads the next character from the file and stores it in nextChar

     if (nextChar != EOF){ // Checks if the character is not the end of the file
        if (isalpha(nextChar)) // Checks if the character is a letter
   {
            charClass = LETTER; // Sets charClass to LETTER if the character is a letter.
```

```
40.     } else if (isdigit(nextChar)){ // Checks if the
   character is a digit
41.             charClass = DIGIT; // Sets charClass to DIGIT
   if the character is a digit
42.         } else {
43.             charClass = UNKNOWN; // Sets charClass to
   UNKNOWN if the character is neither a letter nor a digit.
44.         }
45.     } else {
46.         charClass = END_OF_FILE ; // Sets charClass to
   END_OF_FILE if the character is the end of the file.
47.     }
48.  }
49.
50.  void getNonBlank(){ // Defines the getNonBlank function
51.  while (isspace(nextChar)){ // Checks if the current
   character is a space
52.         getChar();//If it's a space, get the next
   character until it's no longer a space
53.     }
54.  }
55.
56.  int lookup(char ch) {// Define function lookup
57.      switch (ch) {// Starts a switch to check the value of
   ch
58.         case '(': // left parenthesis
59.             addChar(); // Calls addChar() to add the
   character to the lexeme
60.             nextToken = LEFT_PAREN; // puts nextToken to
   the constant LEFT_PAREN
61.             break; // Exits the switch block
62.         case ')': // right parenthesis
63.             addChar(); // Add it to the lexeme
64.             nextToken = RIGHT_PAREN; // Set token to
   RIGHT_PAREN
65.             break; // Exit this case
66.         case '+': // ADD_OP
67.             addChar();
68.             nextToken = ADD_OP; // sets nextToken to
   ADD_OP
69.             break; // Exit
70.         case '-': // SUB_OP
71.             addChar();
```

```
72.             nextToken = SUB_OP; // sets nextToken to SUB
   _OP
73.             break; // Exit
74.         case '*': //MULT_OP
75.             addChar();
76.             nextToken = MULT_OP; // sets nextToken to
   MULT_OP
77.             break; // Exit
78.         case '/': //DIV_OP
79.             addChar();
80.             nextToken = DIV_OP; // sets nextToken to
   DIV_OP
81.             break; // Exit
82.         case '=': // ASSIGN_OP
83.             addChar();
84.             nextToken = ASSIGN_OP; // Sets the nextToken
   to ASSIGN_OP
85.             break; // Exit
86.         default:
87.             addChar();
88.             nextToken = END_OF_FILE; //For any other char,
   add it and set token to EOF
89.             break; // Exit
90.     }
91.     return nextToken; //Return the detected token
92.  }
93.
94.  int lex() // Defines the lex function
95.  {
96.     int i = 0; // Declares a variable i
97.     memset(lexeme, 0, sizeof(lexeme));
98.  // Clears the lexeme array by setting all its elements to
   0
99.     getNonBlank();
100. //Calls the getNonBlank function to skip any whitespace
   characters and move to the next valid character.
101.
102.    switch ((charClass) )// Starts a switch based on the
   character class
103. {
104.         case LETTER: // If the character is a letter
105.             addChar();// Add the character to the lexeme
106.             getChar();// Get the next character
```

```
107.            while (charClass == LETTER || charClass ==
   DIGIT)
108. // continue adding characters if they are letters or
   digits
109. {
110.                addChar();// Add the character to the
   lexeme
111.                getChar();// Get the next character
112.            }
113.            nextToken = IDENT; // Set the token to
   identifier
114.            break; // Exit
115.
116.        case DIGIT: // If the character is a digit
117.            addChar(); // Add the character to the lexeme
118.            getChar();  //Get the next character
119.            while (charClass == DIGIT)
120. // continue adding characters as long as they are digits
121. {
122.                addChar();// Add the character to the
   lexeme
123.                getChar();//Get the next character
124.            }
125.            nextToken = INT_LIT; //Set the token to
   integer literal
126.            break; // Exit
127.
128.        case UNKNOWN:// If the character is not a letter
   or digit
129.            lookup(nextChar); // Call lookup()
130.            getChar();// Get the next character
131.            break; // Exit
132.
133.        case END_OF_FILE: // Checks if the charClass is
   END_OF_FILE
134.            nextToken = END_OF_FILE;
135. // Sets nextToken to END_OF_FILE, indicating the end of
   the file
136.            strcpy(lexeme, "EOF");
137. // Copies the string "EOF" into the lexeme array to
   represent the end of file.
138.            break; // Exit
139.    }
```

```
140.
141.     printf("Next token is: %d, Next lexeme is: %s\n",
   nextToken, lexeme);
142. //Prints the token code and the lexeme for debugging
   purposes.
143.     return nextToken; //Returns the nextToken to indicate
   the type of the token
144. }
145.
146. int main() {
147.     inFile = fopen("/Users/macbookpro/Downloads/front.in",
   "r");
148. // Opens the input file (front.in) in read mode and
   assigns the file pointer to inFile.
149.     if (!inFile)
150. // Checks if the file failed to open
151. {
152.         printf("ERROR - cannot open front.in\n");
153. // Prints an error message if the file cannot be opened
154.         return 1;
155.     }
156.
157.     getChar(); // Reads the first character from the file.
158.     do {
159.         lex();
160.     } while (nextToken != END_OF_FILE);
161. // Calls the lex function to process tokens until the end
   of the file is reached (nextToken becomes END_OF_FILE)
162.
163.     fclose(inFile); // Closes the input file after
   processing.
164.     return 0;
165. }
```

166. References

Concepts of programming languages book

Important Note: -

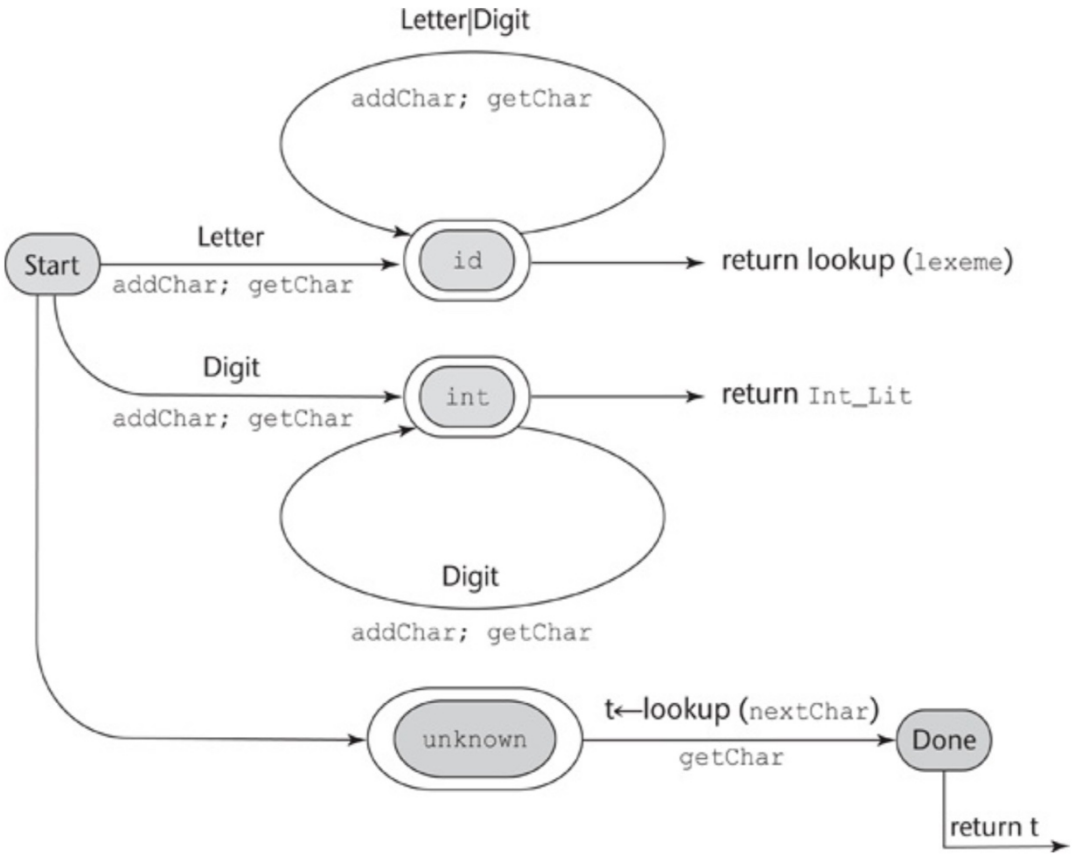
Technical reports include a mixture of text, tables, and figures. Consider how you can present the information best for your reader. Would a table or figure help to convey your ideas more effectively than a paragraph describing the same data?

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.
    - 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
    - Next token is: -1, Next lexeme is: EOF
    - Program ended with exit code: 0

(Sum + 47)/total

Lexeme	Taken
(	L_paren
Sum	identefier
+	Add_op
47	Int_litral
)	R_paren
/	Div_op
Total	identefier


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