A black background with grey leaves

AI-generated content may be incorrect.

Lexical Analyzer

Build Scanner

A grey logo on a black background

AI-generated content may be incorrect.

**Prepared By**

Name Student:shahd omar

ID:200044386.

**Under Supervision**

Name of Doctor:Nehal

Name of T.A:eslam

1. **Introduction).**

The lexical analyzer is an important part of the compilation. Its job is to accept the source code and divide it into tokens, which include keywords, variables, and operators. The tokens simplify the task of comprehending the code for the compiler.

In this project, I have used a C program which reads simple arithmetic expressions from a file, parses them, and translates them to tokens in order to make the rest of the compilation steps easier (like parsing or code generation).

* 1. **Phases of Compiler**

1. phases: A compiler operates in multiple phases, each responsible for a different aspect of translation.
2. Phase Description
3. Lexical Analysis
4. Syntax Analysis
5. Semantic Analysis
6. Intermediate Code Generation Produces an intermediate representation. Code Optimization Improves performance by reducing redundancies.
7. Converts source code into tokens.
8. Checks grammatical correctness and builds a parse tree. Ensures valid meaning and detects type errors.
9. Code Generation Converts optimized code to machine code. Error Handling Identifies and reports errors.
10. Lexical Analyzer
11. Phase Description
12. **Lexical Analyzer**

A Lexical Analyzer: reads the source code character by character and groups them into meaningful tokens.

1. **Software Tools**
   1. **Computer Program:**

VS code

* 1. **Programming Language:**

C

1. **Implementation of a Lexical Analyzer:**

4./\* front.c - a lexical analyzer system for simple arithmetic expressions \*/ #include <stdio.h> #includ; <ctype.h>-(stido.h for input/output)(ctype.h for checking chars)

1. /\* Global declarations / / Variables \*/ int charClass; // Character class char lexeme[100]; (global variables:charclass:type of char,lexeme:current of token,)
2. /\*char next char//int lex len;int token;int next token()file\*in\_fp,\*fopen(char next char:hold the current char,int lexlen:used to track length of current lexeme,file\*in\_fp \*fopen(): pointer ti input file ,function used to open it)
3. /\* Function declarations \*/ void addChar(); // Function to add nextChar to lexeme void getChar(); // Function to get the next character of input void getNonBlank(); // Function to skip whitespace characters int lex(); // Lexical analyzer function
4. /\* Character classes \*/ #define LETTER 0 // Character class for letters #define DIGIT 1 // Character class for digits #define UNKNOWN 99 // Character class for unknown characters
5. /\* Token codes \*/ #define INT\_LIT 10 // Token code for integer literals #define IDENT 11 // Token code for identifiers #define ASSIGN\_OP 20 // Token code for assignment operator #define ADD\_OP 21 // Token code for addition operator #define SUB\_OP 22 // Token code for subtraction operator #define MULT\_OP 23 // Token code for multiplication operator #define DIV\_OP 24 // Token code for division operator #define LEFT\_PAREN 25 // Token code for left parenthesis #define RIGHT\_PAREN 26 // Token code for right parenthesis
6. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ /\* main driver \*/
7. main(){
8. /\* Open the input data file and process its contents \*/ if ((in\_fp = fopen("front.in", "r")) == NULL) { (about to open file and start working if it fails open become null which is the file not found )
9. printf("ERROR - cannot open front.in \n"); }else{ (file didn’t open so its error message showed)(else:the file opened)
10. getChar(); // Get the first character from the input file do{(get char :read first character from file)
11. lex(); // Call the lexical analyzer
12. } while (nextToken != EOF); // Continue until EOF is reached keeps calling lex
13. }
14. } //curly brackets
15. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ /\* lookup - a function to lookup operators and parentheses and return the token / int lookup(char ch) { switch (ch) { case '(': addChar(); nextToken = LEFT\_PAREN; break; case ')': addChar(); nextToken = RIGHT\_PAREN; break; case '+': addChar(); nextToken = ADD\_OP; break; case '-': addChar(); nextToken = SUB\_OP; break; case '': addChar(); nextToken = MULT\_OP; break; case '/': addChar(); nextToken = DIV\_OP; break; default: addChar(); nextToken = EOF; break; } return nextToken; // Return the token }
16. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ /\* addChar - a function to add nextChar to lexeme \*/ void addChar() { if (lexLen <= 98) { lexeme[lexLen++] = nextChar; // Add nextChar to lexeme lexeme[lexLen] = 0; // Null- terminate the lexeme } else { printf("Error - lexeme is too long \n"); // Error message if lexeme is too long } }
17. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ /\* getChar - a function to get the next character of input and determine its character class \*/ void getChar() { if ((nextChar = getc(in\_fp)) != EOF) { if (isalpha(nextChar)) { charClass = LETTER; // Set character class to LETTER } else if (isdigit(nextChar)) { charClass = DIGIT; // Set character class to DIGIT } else { charClass = UNKNOWN; // Set character class to UNKNOWN } } else { charClass = EOF; // Set character class to EOF}}
18. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ /\* getNonBlank - a function to call getChar until it returns a non-whitespace character \*/ void getNonBlank() { while (isspace(nextChar)) { // While the character is whitespace getChar(); // Get the next character } }
19. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ /\* lex - a simple lexical analyzer for arithmetic expressions / int lex() { lexLen = 0; // Reset lexeme length getNonBlank(); // Skip any leading whitespace switch (charClass) { / Parse identifiers \*/ case LETTER: addChar(); getChar(); while (charClass == LETTER || charClass == DIGIT) { // While character is LETTER or DIGIT addChar(); getChar(); } nextToken = IDENT; break;
20. /\* Parse integer literals \*/ case DIGIT:
21. addChar();
22. getChar();
23. while (charClass == DIGIT) { // While character is DIGIT
24. addChar();
25. getChar(); }
26. nextToken = INT\_LIT; break; //integer literal token to int and then use break to exit
27. /\* Parentheses and operators \*/ case UNKNOWN://if char was symbol we call lookup to figure out what symbol it is
28. lookup(nextChar); // Lookup the character getChar(); // Get the next character
29. break;
30. /\* EOF \*/ case EOF:
31. nextToken = EOF; // Set token to EOF lexeme[0] = 'E'; // Set lexeme to "EOF" lexeme[1] = 'O';
32. lexeme[2] = 'F';
33. lexeme[3] = 0; // Null-terminate the lexeme
34. break; }
35. /\* End of switch \*/ //end of big switch statement inside the lex()function
36. printf("Next token is: %d, Next lexeme is %s\n", nextToken, lexeme); // Output the next token and lexeme return nextToken; //
37. Return the next token the function is now done
38. } curly brackets

44./\* End of function lex \*/ //end of function lex

1. **References**

Geekforgeeks,Wikipedia-lexical Analysis