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 Malak ahmed Mohamed abomay sheikh

ID 200064046

**Under Supervision**

Doctor Nehal abdel salam

T. A. Islam saied

1. **Introduction** 
   * This project is about creating a simple program called a Lexical Analyzer, which is like a helper for understanding math expressions written in code. The main job of this program is to read a string of characters and figure out what each part means, like numbers, letters, or symbols.
   * For example, if you give it something like:
   * (value1 - 23) \* result
   * It will break it down into parts, like:
   * • value1 is a name (identifier).
   * • 23 is a number (integer).
   * • \*, -, and () are symbols (operators or parentheses).
   * The program is written in C++ and uses simple tools like iostream to print things, cctype to check if something is a letter or number, and string to handle text.
   * I wrote this program to learn how computers read and understand code step by step, and it was fun to see how something so small can be so useful
   * **Phases of Compiler**

Next token is: 25, Next lexeme is (

Next token is: 11, Next lexeme is value1

Next token is: 22, Next lexeme is -

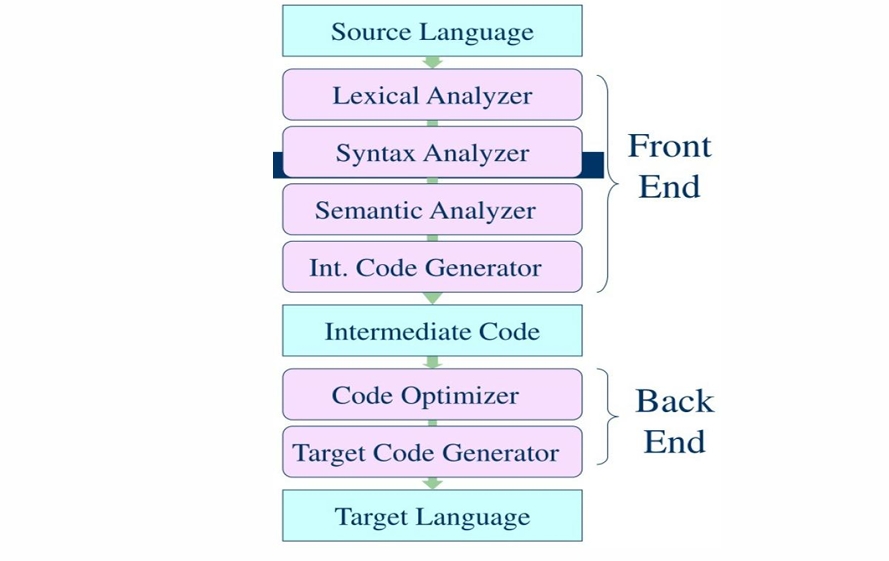
Next token is: 10, Next lexeme is 23

Next token is: 26, Next lexeme is )

Next token is: 23, Next lexeme is \*

Next token is: 11, Next lexeme is result

Next token is: -1, Next lexeme is EOF

1. **Lexical Analyzer**

#include <iostream>

#include <cctype>

#include <string>

using namespace std;

int charClass;

char nextChar;

int lexLen;

int nextToken;

string lexeme;

string input = "(value1 - 23) \* result";

int inputIndex = 0;

#define LETTER 0

#define DIGIT 1

#define UNKNOWN 99

#define INT\_LIT 10

#define IDENT 11

#define ASSIGN\_OP 20

#define ADD\_OP 21

#define SUB\_OP 22

#define MULT\_OP 23

#define DIV\_OP 24

#define LEFT\_PAREN 25

#define RIGHT\_PAREN 26

#define END\_OF\_INPUT -1

void addChar() {

if (lexLen <= 98) {

lexeme += nextChar;

lexLen++;

}

}

void getChar() {

if (inputIndex < input.length()) {

nextChar = input[inputIndex++];

if (isalpha(nextChar))

charClass = LETTER;

else if (isdigit(nextChar))

charClass = DIGIT;

else

charClass = UNKNOWN;

} else {

charClass = END\_OF\_INPUT;

}

}

void getNonBlank() {

while (isspace(nextChar))

getChar();

}

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 = END\_OF\_INPUT;

break;

}

return nextToken;

}

int lex() {

lexeme = "";

lexLen = 0;

getNonBlank();

switch (charClass) {

case LETTER:

addChar();

getChar();

while (charClass == LETTER || charClass == DIGIT) {

addChar();

getChar();

}

nextToken = IDENT;

break;

case DIGIT:

addChar();

getChar();

while (charClass == DIGIT) {

addChar();

getChar();

}

nextToken = INT\_LIT;

break;

case UNKNOWN:

lookup(nextChar);

getChar();

break;

case END\_OF\_INPUT:

nextToken = END\_OF\_INPUT;

lexeme = "EOF";

break;

}

cout << "Next token is: " << nextToken << ", Next lexeme is " << lexeme << endl;

return nextToken;

}

int main() {

getChar();

do {

lex();

} while (nextToken != END\_OF\_INPUT);

return 0;

}

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

# visual studio code c++

* 1. **Programming Language**

# The lexical analyzer is implemented using C++ due to its efficiency and low-level control over string and character manipulation, making it suitable for building compilers and interpreters.

1. **Implementation of a Lexical Analyzer**
2. #include <iostream> // Includes the input/output stream library for using cout and cin
3. #include <cctype> // Includes character handling functions like isalpha(), isdigit(), isspace()
4. #include <string> // Includes the string class for using string variables
5. using namespace std; // Allows using standard namespace elements without writing std:: prefix
6. int charClass; // Stores the type of the current character (LETTER, DIGIT, UNKNOWN)
7. char nextChar; // The current character being analyzed
8. int lexLen; // Length of the current lexeme (string being built)
9. int nextToken; // Token type returned by the lexer (e.g., IDENT, INT\_LIT, ADD\_OP)
10. string lexeme; // Stores the current lexeme (word or symbol being processed)
11. string input = "(value1 - 23) \* result"; // Input string to be analyzed
12. int inputIndex = 0; // Index to track the current position in the input string
13. #define LETTER 0 // Represents a letter character
14. #define DIGIT 1 // Represents a digit character
15. #define UNKNOWN 99 // Represents unknown characters (operators or symbols)
16. #define INT\_LIT 10 // Token code for integer literals
17. #define IDENT 11 // Token code for identifiers (like variable names)
18. #define ASSIGN\_OP 20 // Token code for assignment operator (=) [not used in this code]
19. #define ADD\_OP 21 // Token code for addition operator (+)
20. #define SUB\_OP 22 // Token code for subtraction operator (-)
21. #define MULT\_OP 23 // Token code for multiplication operator (\*)
22. #define DIV\_OP 24 // Token code for division operator (/)
23. #define LEFT\_PAREN 25 // Token code for left parenthesis '('
24. #define RIGHT\_PAREN 26 // Token code for right parenthesis ')'
25. #define END\_OF\_INPUT -1 // Indicates the end of the input string
26. void addChar() {
27. if (lexLen <= 98) { // If the lexeme is within allowed size
28. lexeme += nextChar; // Append the current character to lexeme
29. lexLen++; // Increase the lexeme length
30. }
31. }
32. void getChar() {
33. if (inputIndex < input.length()) { // If there are more characters in the input
34. nextChar = input[inputIndex++]; // Get the next character and move index
35. if (isalpha(nextChar)) // If the character is a letter
36. charClass = LETTER;
37. else if (isdigit(nextChar)) // If it's a digit
38. charClass = DIGIT;
39. else // Otherwise, it's unknown (e.g., operator)
40. charClass = UNKNOWN;
41. } else {
42. charClass = END\_OF\_INPUT; // No more characters; end of input
43. }
44. }
45. void getNonBlank() {
46. while (isspace(nextChar)) { // Skip whitespace characters (like space, tab)
47. getChar(); // Get the next non-whitespace character
48. }
49. }
50. int lookup(char ch) { // Identifies single-character symbols
51. switch (ch) {
52. case '(': addChar(); nextToken = LEFT\_PAREN; break;
53. case ')': addChar(); nextToken = RIGHT\_PAREN; break;
54. case '+': addChar(); nextToken = ADD\_OP; break;
55. case '-': addChar(); nextToken = SUB\_OP; break;
56. case '\*': addChar(); nextToken = MULT\_OP; break;
57. case '/': addChar(); nextToken = DIV\_OP; break;
58. default: addChar(); nextToken = END\_OF\_INPUT; break; // Unknown symbol
59. }
60. return nextToken; // Return the identified token
61. }
62. int lex() {
63. lexeme = ""; // Clear the current lexeme
64. lexLen = 0; // Reset lexeme length
65. getNonBlank(); // Skip any whitespace
66. switch (charClass) {
67. case LETTER: // If the character is a letter (start of identifier)
68. addChar(); // Add it to the lexeme
69. getChar(); // Get the next character
70. while (charClass == LETTER || charClass == DIGIT) {
71. addChar(); // Add letters/digits to the identifier
72. getChar();
73. }
74. nextToken = IDENT; // Token type is identifier
75. break;
76. case DIGIT: // If the character is a digit (start of number)
77. addChar();
78. getChar();
79. while (charClass == DIGIT) {
80. addChar(); // Add digits to the integer literal
81. getChar();
82. }
83. nextToken = INT\_LIT; // Token type is integer literal
84. break;
85. case UNKNOWN: // If the character is an operator or symbol
86. lookup(nextChar); // Identify it using the lookup function
87. getChar(); // Move to next character
88. break;
89. case END\_OF\_INPUT: // If we reached the end of the input
90. nextToken = END\_OF\_INPUT;
91. lexeme = "EOF"; // Set lexeme to End Of File
92. break;
93. }
94. // Output the token and its lexeme
95. cout << "Next token is: " << nextToken << ", Next lexeme is " << lexeme << endl;
96. return nextToken; // Return the token type
97. }
98. int main() {
99. getChar(); // Get the first character from input
100. do {
101. lex(); // Call the lexical analyzer
102. } while (nextToken != END\_OF\_INPUT); // Repeat until end of input
103. return 0; // Exit the program
104. }
105. **References**

# Concepts of Programming Languages By Robert W. Sebesta (12th Edition)

# Online C++ documentation

# Lecture notes

# Laps