CS224-Theory of Automata

Project Report: Lexical Analyzer

Group Members:

Raja Bilal Khurram (2023591)

Muhammad Ammar Saleem (2023378)

Course: CS224 – Theory of Automata

Submitted to Sajjid Ali

C++ Tokenizer

Introduction

A **Lexical Analyzer (LA)** is a fundamental component of a compiler's front-end. It processes the source code to identify and classify sequences of characters into meaningful tokens, such as keywords, identifiers, operators, and literals. This tokenization simplifies the parsing process and aids in syntax and semantic analysis.

In this project, we have developed a Lexical Analyzer for the **C++ language** using **Flex**, a tool for generating scanners. Our analyzer reads C++ source files, identifies various tokens, and outputs their type, value, and line number.

Features of Our Lexical Analyzer

- Recognition of **C++ keywords** (e.g., `int`, `float`, `if`, `else`, `return`)
- Identification of **identifiers**, **constants** (integers, floats, characters, strings)
- Parsing of **operators**: arithmetic, logical, bitwise, assignment, relational
- Detection of **punctuation marks** and **delimiters** (`;`, `,`, `()`, `{}`, `[]`)
- Handling of **comments**: single-line (`//`) and multi-line (`/*...*/`)
- Ignoring of **whitespace** while maintaining accurate **line numbers**
- Output of **token type**, **value**, and **line number** to the console

---

Regular Expressions for Token Classes

digit [0-9]
letter [A-Za-z_]
identifier {letter}({letter}|{digit})*
data_type int|float|double|char|bool|string|long|short|void
accessspecifier private|protected|public
keyword
if|else|while|for|return|break|continue|switch|case|default|sizeof|do|goto|enum|typedef|struct|class|const|static|volatile|signed|unsigned|try|catch|throw|new|delete

brackets [\{\}\[\]\(\)]
line_comment \/\/[^\n]*
block_comment "\/\([^]|\+[^/])\+\/"
integer_constant {digit}+
float_constant {digit}+"."{digit}+
character_constant \'([^\\']|\\.)\'
string_literal \"([^\\\"]|\\.)*\"
library \#include[\t]*<[^>]+>
delimiter [;,]
assignment_operator =
arithmetic_operator [+\-*/%]
relational_operator (==|!=|<=|>=|<|>)
logical_operator (\|\||&&|!)
bitwise_operator (&|\||\^|~|<<|>>)
increment_decrement (\+\+|--)

Sample Code and Output Explanation

Sample Input (`test1.cpp`)

```
int main() {
  float x = 3.14;
  // This is a comment
  if (x > 0) {
    x = x + 1;
  }
  return 0;
}
```

Sample Output

```
Line 1: Token = int → Data Type
Line 1: Token = main → Identifier
Line 1: Token = ( → Bracket/Parenthesis
Line 1: Token = ) → Bracket/Parenthesis
Line 1: Token = { → Bracket/Parenthesis
Line 2: Token = float → Data Type
Line 2: Token = x → Identifier
Line 2: Token = = → Assignment Operator
Line 2: Token = 3.14 → Float Constant
Line 2: Token = ; → Delimiter
Line 3: Token = // This is a comment → Line Comment
Line 4: Token = if → Keyword
Line 4: Token = ( → Bracket/Parenthesis
Line 4: Token = x → Identifier
Line 4: Token = > → Relational Operator
Line 4: Token = 0 → Integer Constant
Line 4: Token = ) → Bracket/Parenthesis
```

```
Line 4: Token = { → Bracket/Parenthesis
Line 5: Token = x → Identifier
Line 5: Token = = → Assignment Operator
Line 5: Token = x → Identifier
Line 5: Token = + → Arithmetic Operator
Line 5: Token = 1 → Integer Constant
Line 5: Token = ; → Delimiter
Line 6: Token = } → Bracket/Parenthesis
Line 7: Token = return → Keyword
Line 7: Token = 0 → Integer Constant
Line 7: Token = ; → Delimiter
Line 8: Token = } → Bracket/Parenthesis
```

Main Code Explanation

`int main` Function

```
int main(int argc, char **argv) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <source-file>\\n", argv[0]);
        return EXIT_FAILURE;
    }
    yyin = fopen(argv[1], "r");
    if (!yyin) {
        perror("fopen");
        return EXIT_FAILURE;
    }
    yylex();
    fclose(yyin);
    return EXIT_SUCCESS;
}
```

This function checks for the input source file passed as a command-line argument. It opens the file and passes it to Flex's `yylex()` function, which begins tokenizing. After processing, the file is closed.

Use of Theory of Automata in Our Project

In our Lexical Analyzer project, we leveraged several concepts from the theory of automata to effectively tokenize and classify C++ source code. The theory of automata provides the foundation for understanding how regular expressions can be transformed into finite state machines (FSMs), which are essential for recognizing patterns in the input text. Specifically, we utilized **Deterministic Finite Automata (DFA)** to process the regular expressions corresponding to different token classes (such as identifiers, keywords, operators, etc.).

Each token type in the C++ language can be described by a regular expression, and the Lex tool, which we used for the project, automatically constructs an NFA (Non-deterministic Finite Automaton) from these regular expressions. This NFA is then converted into an efficient DFA that can quickly match patterns in the input text. The use of DFA ensures that we process each character of the input source code exactly once, making the lexical analysis both time and space-efficient.

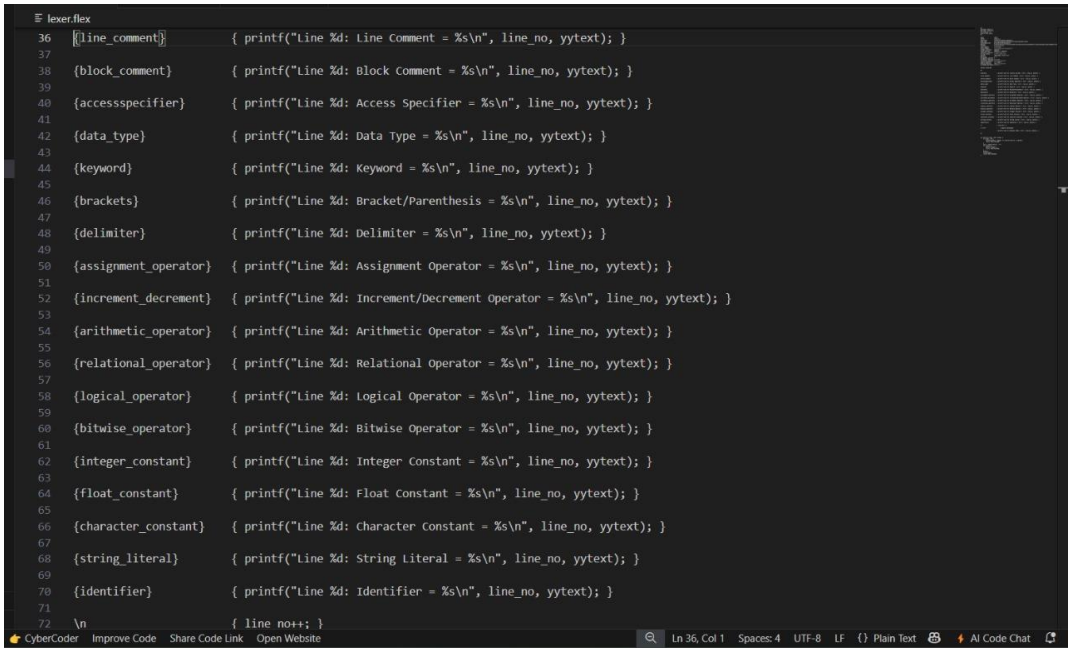
Additionally, the concept of **state transitions** in automata was directly applied to track the progress of the lexical analyzer as it reads characters from the input. For example, when processing identifiers, the automaton starts in an initial state and transitions through various states as it encounters valid characters (letters and digits), ultimately recognizing a valid identifier.

By applying automata theory, we ensured that our lexical analyzer is capable of efficiently recognizing and classifying tokens while maintaining the order and structure of the source code. This implementation also helps handle complex constructs such as nested comments and multi-character operators, showcasing the power of automata theory in real-world applications like compiler design.

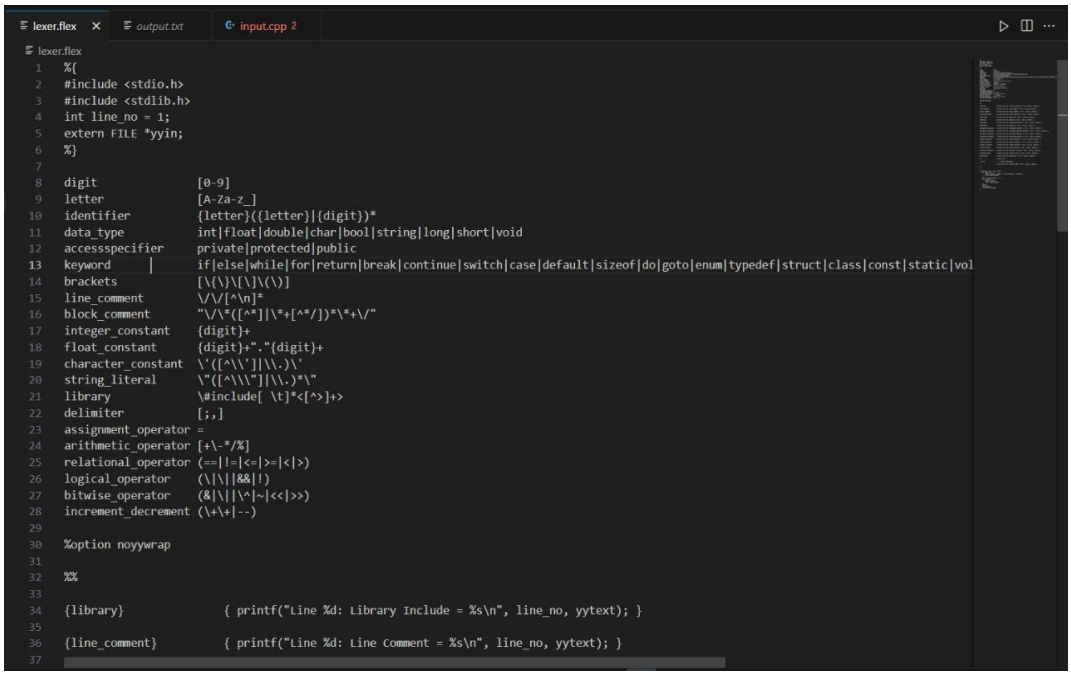
Conclusion

In conclusion, this project successfully demonstrates the development of a Lexical Analyzer for the C++ programming language using Flex. By defining appropriate regular expressions and token patterns, we were able to accurately identify and classify a wide variety of lexical components such as keywords, identifiers, constants, operators, and delimiters. The analyzer effectively handles real-world C++ syntax, including comments and whitespace, while maintaining accurate line tracking. This foundational work plays a critical role in compiler design and provides practical insight into how source code is broken down during the initial phases of compilation.

Our Lexical Analyzer Code :



```
lexer.flex
{line_comment}          { printf("Line %d: Line Comment = %s\n", line_no, yytext); }

{block_comment}         { printf("Line %d: Block Comment = %s\n", line_no, yytext); }

{accessspecifier}       { printf("Line %d: Access Specifier = %s\n", line_no, yytext); }

{data_type}             { printf("Line %d: Data Type = %s\n", line_no, yytext); }

{keyword}               { printf("Line %d: Keyword = %s\n", line_no, yytext); }

{brackets}              { printf("Line %d: Bracket/Parenthesis = %s\n", line_no, yytext); }

{delimiter}             { printf("Line %d: Delimiter = %s\n", line_no, yytext); }

{assignment_operator}   { printf("Line %d: Assignment Operator = %s\n", line_no, yytext); }

{increment_decrement}   { printf("Line %d: Increment/Decrement Operator = %s\n", line_no, yytext); }

{arithmetic_operator}   { printf("Line %d: Arithmetic Operator = %s\n", line_no, yytext); }

{relational_operator}   { printf("Line %d: Relational Operator = %s\n", line_no, yytext); }

{logical_operator}      { printf("Line %d: Logical Operator = %s\n", line_no, yytext); }

{bitwise_operator}      { printf("Line %d: Bitwise Operator = %s\n", line_no, yytext); }

{integer_constant}      { printf("Line %d: Integer Constant = %s\n", line_no, yytext); }

{float_constant}        { printf("Line %d: Float Constant = %s\n", line_no, yytext); }

{character_constant}    { printf("Line %d: Character Constant = %s\n", line_no, yytext); }

{string_literal}        { printf("Line %d: String Literal = %s\n", line_no, yytext); }

{identifier}            { printf("Line %d: Identifier = %s\n", line_no, yytext); }

\n                      { line_no++; }
CyberCoder  Improve Code  Share Code Link  Open Website   Ln 36, Col 1  Spaces: 4  UTF-8  LF  {} Plain Text  AI Code Chat
```



```
lexer.flex   output.txt   input.cpp 2
lexer.flex
%{
#include <stdio.h>
#include <stdlib.h>
int line_no = 1;
extern FILE *yyin;
%}

digit               [0-9]
letter              [A-Za-z_]
identifier          {letter}({letter}|{digit})*
data_type           int|float|double|char|bool|string|long|short|void
accessspecifier     private|protected|public
keyword             if|else|while|for|return|break|continue|switch|case|default|sizeof|do|goto|enum|typedef|struct|class|const|static|vol
brackets            [\{\}\[\]\(\)]
line_comment        \/\/[^\n]*
block_comment       "\/\*([^*]|\*+[^*/])*\*+\/"
integer_constant    {digit}+
float_constant      {digit}+"."{digit}+
character_constant  \'([^\\\']|\\.)\'
string_literal      \"([^\\\"]|\\.)*\"
library             \#include[ \t]*<[^>]+>
delimiter           [;,]
assignment_operator =
arithmetic_operator [+\-*/%]
relational_operator (==|!=|<=|>=|<|>)
logical_operator    (\|\||&&|!)
bitwise_operator    (&|\||\^|~|<<|>>)
increment_decrement (\+\+|--)

%option noyywrap

%%

{library}               { printf("Line %d: Library Include = %s\n", line_no, yytext); }

{line_comment}          { printf("Line %d: Line Comment = %s\n", line_no, yytext); }
```

lexer.flex | output.txt | input.cpp 2

lexer.flex

```
67
68  {string_literal}        { printf("Line %d: String Literal = %s\n", line_no, yytext); }
69
70  {identifier}            { printf("Line %d: Identifier = %s\n", line_no, yytext); }
71
72  \n                      { line_no++; }
73
74  [ \t\r]+                ;   // Ignore whitespace
75
76  .                       { printf("Line %d: Unknown Token = %s\n", line_no, yytext); }
77
78  %%
79
80
81  int main(int argc, char **argv) {
82      if (argc < 2) {
83          fprintf(stderr, "Usage: %s <source-file>\n", argv[0]);
84          return EXIT_FAILURE;
85      }
86      yyin = fopen(argv[1], "r");
87      if (!yyin) {
88          perror("fopen");
89          return EXIT_FAILURE;
90      }
91      yylex();
92      fclose(yyin);
93      return EXIT_SUCCESS;
94  }
95
```

Sample 1:

input.cpp > main()

```
1   #include <iostream>
2   #include <cmath>
3   using namespace std;
4
5   int main() {
6       int a = 5;
7       float b = 10.5;
8       double c = 3.14159;
9       char d = 'X';
10      string message = "Hello, Lexical Analyzer!";
11
12      cout << "Value of a: " << a << endl;
13      cout << "Value of b: " << b << endl;
14      cout << "Value of c: " << c << endl;
15      cout << "Character d: " << d << endl;
16
17      float sum = a + b;
18      double area = c * pow(a, 2);
19
20      cout << "Sum of a and b: " << sum << endl;
21      cout << "Area of circle with radius a: " << area << endl;
22
23      if (a > 0) {
24          cout << "a is positive." << endl;
25      } else {
26          cout << "a is non-positive." << endl;
27      }
28
29      for (int i = 0; i < 5; i++) {
30          cout << "Loop iteration: " << i << endl;
31      }
32
33      return 0;
34  }
35
```

CyberCoder | Improve Code | Share Code Link | Open Website | Ln 6, Col 15 | Spaces: 4 | UTF-8 | CRLF | {} C++ | AI Code Chat | Win32

Sample 1 (output) :

output.txt

```
Line 1: Library Include = #include <iostream>
Line 2: Library Include = #include <cmath>
Line 3: Identifier = using
Line 3: Identifier = namespace
Line 3: Identifier = std
Line 3: Delimiter = ;
Line 5: Data Type = int
Line 5: Identifier = main
Line 5: Bracket/Parenthesis = (
Line 5: Bracket/Parenthesis = )
Line 5: Bracket/Parenthesis = {
Line 6: Data Type = int
Line 6: Identifier = a
Line 6: Assignment Operator = =
Line 6: Integer Constant = 5
Line 6: Delimiter = ;
Line 7: Data Type = float
Line 7: Identifier = b
Line 7: Assignment Operator = =
Line 7: Float Constant = 10.5
Line 7: Delimiter = ;
Line 8: Data Type = double
Line 8: Identifier = c
Line 8: Assignment Operator = =
Line 8: Float Constant = 3.14159
Line 8: Delimiter = ;
Line 9: Data Type = char
Line 9: Identifier = d
Line 9: Assignment Operator = =
Line 9: Character Constant = 'X'
Line 9: Delimiter = ;
Line 10: Data Type = string
Line 10: Identifier = message
Line 10: Assignment Operator = =
Line 10: String Literal = "Hello, Lexical Analyzer!"
Line 10: Delimiter = ;
Line 12: Identifier = cout
```

output.txt

```
Line 10: String Literal = "Hello, Lexical Analyzer!"
Line 10: Delimiter = ;
Line 12: Identifier = cout
Line 12: Bitwise Operator = <<
Line 12: String Literal = "Value of a: "
Line 12: Bitwise Operator = <<
Line 12: Identifier = a
Line 12: Bitwise Operator = <<
Line 12: Identifier = endl
Line 12: Delimiter = ;
Line 13: Identifier = cout
Line 13: Bitwise Operator = <<
Line 13: String Literal = "Value of b: "
Line 13: Bitwise Operator = <<
Line 13: Identifier = b
Line 13: Bitwise Operator = <<
Line 13: Identifier = endl
Line 13: Delimiter = ;
Line 14: Identifier = cout
Line 14: Bitwise Operator = <<
Line 14: String Literal = "Value of c: "
Line 14: Bitwise Operator = <<
Line 14: Identifier = c
Line 14: Bitwise Operator = <<
Line 14: Identifier = endl
Line 14: Delimiter = ;
Line 15: Identifier = cout
Line 15: Bitwise Operator = <<
Line 15: String Literal = "Character d: "
Line 15: Bitwise Operator = <<
Line 15: Identifier = d
Line 15: Bitwise Operator = <<
Line 15: Identifier = endl
Line 15: Delimiter = ;
Line 17: Data Type = float
Line 17: Identifier = sum
```

output.txt

```
Line 17: Data Type = float
Line 17: Identifier = sum
Line 17: Assignment Operator = =
Line 17: Identifier = a
Line 17: Arithmetic Operator = +
Line 17: Identifier = b
Line 17: Delimiter = ;
Line 18: Data Type = double
Line 18: Identifier = area
Line 18: Assignment Operator = =
Line 18: Identifier = c
Line 18: Arithmetic Operator = *
Line 18: Identifier = pow
Line 18: Bracket/Parenthesis = (
Line 18: Identifier = a
Line 18: Delimiter = ,
Line 18: Integer Constant = 2
Line 18: Bracket/Parenthesis = )
Line 18: Delimiter = ;
Line 20: Identifier = cout
Line 20: Bitwise Operator = <<
Line 20: String Literal = "Sum of a and b: "
Line 20: Bitwise Operator = <<
Line 20: Identifier = sum
Line 20: Bitwise Operator = <<
Line 20: Identifier = endl
Line 20: Delimiter = ;
Line 21: Identifier = cout
Line 21: Bitwise Operator = <<
Line 21: String Literal = "Area of circle with radius a: "
Line 21: Bitwise Operator = <<
Line 21: Identifier = area
Line 21: Bitwise Operator = <<
Line 21: Identifier = endl
Line 21: Delimiter = ;
Line 23: Keyword = if
Line 23: Bracket/Parenthesis = (
```

Desktop 2

output.txt

```
Line 21: Bitwise Operator = <<
Line 21: Identifier = endl
Line 21: Delimiter = ;
Line 23: Keyword = if
Line 23: Bracket/Parenthesis = (
Line 23: Identifier = a
Line 23: Relational Operator = >
Line 23: Integer Constant = 0
Line 23: Bracket/Parenthesis = )
Line 23: Bracket/Parenthesis = {
Line 24: Identifier = cout
Line 24: Bitwise Operator = <<
Line 24: String Literal = "a is positive."
Line 24: Bitwise Operator = <<
Line 24: Identifier = endl
Line 24: Delimiter = ;
Line 25: Bracket/Parenthesis = }
Line 25: Keyword = else
Line 25: Bracket/Parenthesis = {
Line 26: Identifier = cout
Line 26: Bitwise Operator = <<
Line 26: String Literal = "a is non-positive."
Line 26: Bitwise Operator = <<
Line 26: Identifier = endl
Line 26: Delimiter = ;
Line 27: Bracket/Parenthesis = }
Line 29: Keyword = for
Line 29: Bracket/Parenthesis = (
Line 29: Data Type = int
Line 29: Identifier = i
Line 29: Assignment Operator = =
Line 29: Integer Constant = 0
Line 29: Delimiter = ;
Line 29: Identifier = i
Line 29: Relational Operator = <
Line 29: Integer Constant = 5
Line 29: Delimiter = ;
```

output.txt

```
Line 25: Bracket/Parenthesis = {
Line 26: Identifier = cout
Line 26: Bitwise Operator = <<
Line 26: String Literal = "a is non-positive."
Line 26: Bitwise Operator = <<
Line 26: Identifier = endl
Line 26: Delimiter = ;
Line 27: Bracket/Parenthesis = }
Line 29: Keyword = for
Line 29: Bracket/Parenthesis = (
Line 29: Data Type = int
Line 29: Identifier = i
Line 29: Assignment Operator = =
Line 29: Integer Constant = 0
Line 29: Delimiter = ;
Line 29: Identifier = i
Line 29: Relational Operator = <
Line 29: Integer Constant = 5
Line 29: Delimiter = ;
Line 29: Identifier = i
Line 29: Increment/Decrement Operator = ++
Line 29: Bracket/Parenthesis = )
Line 29: Bracket/Parenthesis = {
Line 30: Identifier = cout
Line 30: Bitwise Operator = <<
Line 30: String Literal = "Loop iteration: "
Line 30: Bitwise Operator = <<
Line 30: Identifier = i
Line 30: Bitwise Operator = <<
Line 30: Identifier = endl
Line 30: Delimiter = ;
Line 31: Bracket/Parenthesis = }
Line 33: Keyword = return
Line 33: Integer Constant = 0
Line 33: Delimiter = ;
Line 34: Bracket/Parenthesis = }

CyberCoder  Improve Code  Share Code Link  Open Website   Ln 85, Col 30  Spaces: 4  UTF-16 LE  CRLF  {} Plain Text  AI Code Chat
```

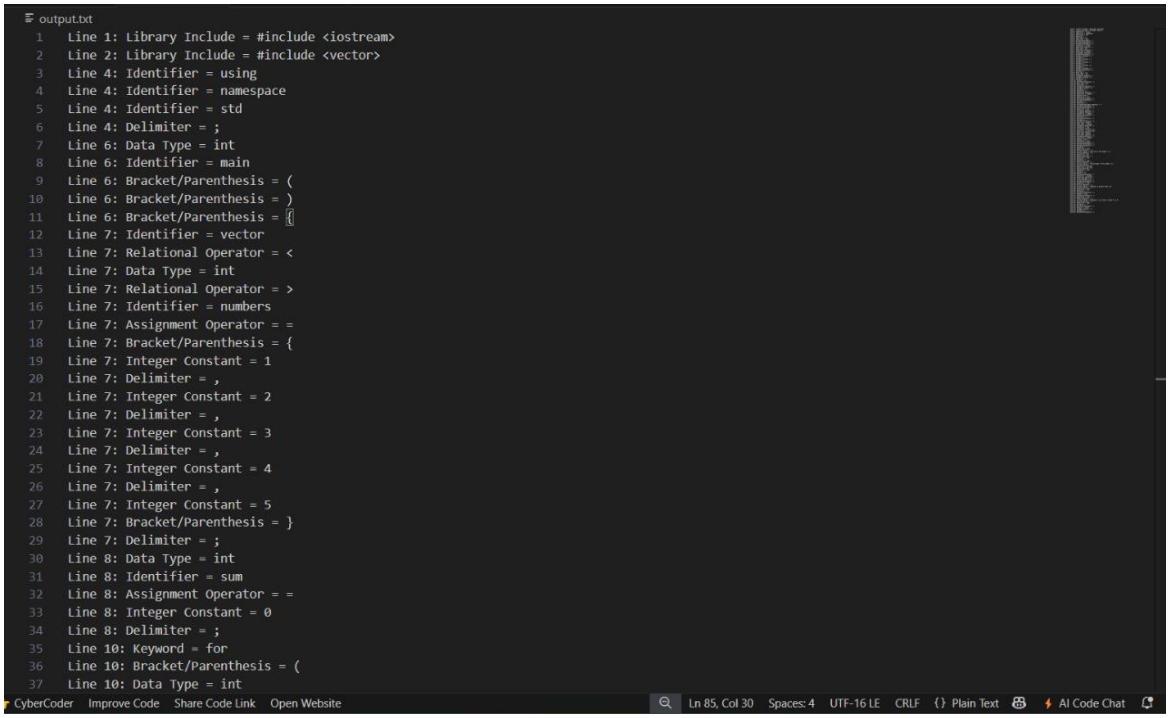
Sample 2:

input.cpp > ...

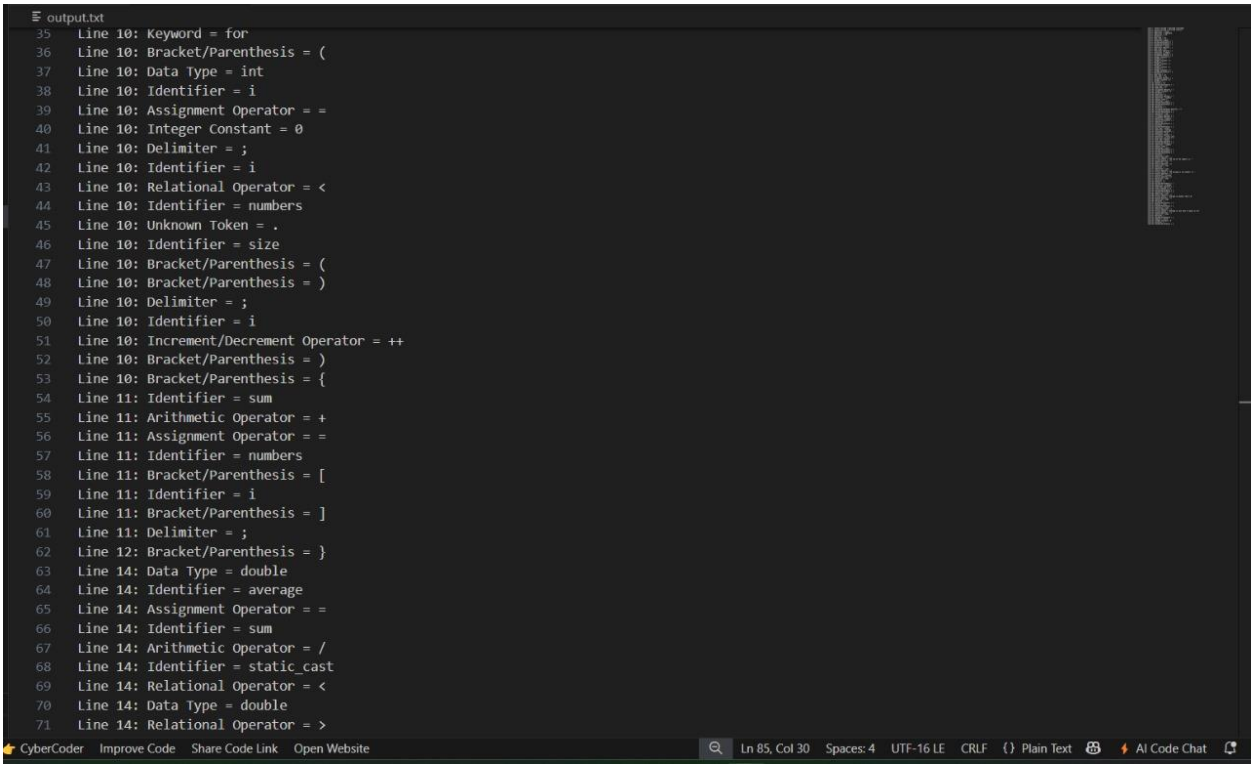
```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> numbers = {1, 2, 3, 4, 5};
    int sum = 0;

    for (int i = 0; i < numbers.size(); i++) {
        sum += numbers[i];
    }

    double average = sum / static_cast<double>(numbers.size());

    cout << "The sum of the numbers is: " << sum << endl;
    cout << "The average of the numbers is: " << average << endl;

    if (average > 3.0) {
        cout << "Average is greater than 3.0" << endl;
    } else {
        cout << "Average is less than or equal to 3.0" << endl;
    }

    return 0;
}
```

CyberCoder Improve Code Share Code Link Open Website   Ln 27, Col 1 Spaces: 4 UTF-8 CRLF {} C++ AI Code Chat Win32

Sample 2 (output) :



```
output.txt
Line 1: Library Include = #include <iostream>
Line 2: Library Include = #include <vector>
Line 4: Identifier = using
Line 4: Identifier = namespace
Line 4: Identifier = std
Line 4: Delimiter = ;
Line 6: Data Type = int
Line 6: Identifier = main
Line 6: Bracket/Parenthesis = (
Line 6: Bracket/Parenthesis = )
Line 6: Bracket/Parenthesis = {
Line 7: Identifier = vector
Line 7: Relational Operator = <
Line 7: Data Type = int
Line 7: Relational Operator = >
Line 7: Identifier = numbers
Line 7: Assignment Operator = =
Line 7: Bracket/Parenthesis = {
Line 7: Integer Constant = 1
Line 7: Delimiter = ,
Line 7: Integer Constant = 2
Line 7: Delimiter = ,
Line 7: Integer Constant = 3
Line 7: Delimiter = ,
Line 7: Integer Constant = 4
Line 7: Delimiter = ,
Line 7: Integer Constant = 5
Line 7: Bracket/Parenthesis = }
Line 7: Delimiter = ;
Line 8: Data Type = int
Line 8: Identifier = sum
Line 8: Assignment Operator = =
Line 8: Integer Constant = 0
Line 8: Delimiter = ;
Line 10: Keyword = for
Line 10: Bracket/Parenthesis = (
Line 10: Data Type = int
CyberCoder  Improve Code  Share Code Link  Open Website   Ln 85, Col 30  Spaces: 4  UTF-16 LE  CRLF  {} Plain Text  AI Code Chat
```



```
output.txt
Line 10: Keyword = for
Line 10: Bracket/Parenthesis = (
Line 10: Data Type = int
Line 10: Identifier = i
Line 10: Assignment Operator = =
Line 10: Integer Constant = 0
Line 10: Delimiter = ;
Line 10: Identifier = i
Line 10: Relational Operator = <
Line 10: Identifier = numbers
Line 10: Unknown Token = .
Line 10: Identifier = size
Line 10: Bracket/Parenthesis = (
Line 10: Bracket/Parenthesis = )
Line 10: Delimiter = ;
Line 10: Identifier = i
Line 10: Increment/Decrement Operator = ++
Line 10: Bracket/Parenthesis = )
Line 10: Bracket/Parenthesis = {
Line 11: Identifier = sum
Line 11: Arithmetic Operator = +
Line 11: Assignment Operator = =
Line 11: Identifier = numbers
Line 11: Bracket/Parenthesis = [
Line 11: Identifier = i
Line 11: Bracket/Parenthesis = ]
Line 11: Delimiter = ;
Line 12: Bracket/Parenthesis = }
Line 14: Data Type = double
Line 14: Identifier = average
Line 14: Assignment Operator = =
Line 14: Identifier = sum
Line 14: Arithmetic Operator = /
Line 14: Identifier = static_cast
Line 14: Relational Operator = <
Line 14: Data Type = double
Line 14: Relational Operator = >
CyberCoder  Improve Code  Share Code Link  Open Website   Ln 85, Col 30  Spaces: 4  UTF-16 LE  CRLF  {} Plain Text  AI Code Chat
```

output.txt

```
Line 14: Relational Operator = <
Line 14: Data Type = double
Line 14: Relational Operator = >
Line 14: Bracket/Parenthesis = (
Line 14: Identifier = numbers
Line 14: Unknown Token = .
Line 14: Identifier = size
Line 14: Bracket/Parenthesis = (
Line 14: Bracket/Parenthesis = )
Line 14: Bracket/Parenthesis = )
Line 14: Delimiter = ;
Line 16: Identifier = cout
Line 16: Bitwise Operator = <<
Line 16: String Literal = "The sum of the numbers is: "
Line 16: Bitwise Operator = <<
Line 16: Identifier = sum
Line 16: Bitwise Operator = <<
Line 16: Identifier = endl
Line 16: Delimiter = ;
Line 17: Identifier = cout
Line 17: Bitwise Operator = <<
Line 17: String Literal = "The average of the numbers is: "
Line 17: Bitwise Operator = <<
Line 17: Identifier = average
Line 17: Bitwise Operator = <<
Line 17: Identifier = endl
Line 17: Delimiter = ;
Line 19: Keyword = if
Line 19: Bracket/Parenthesis = (
Line 19: Identifier = average
Line 19: Relational Operator = >
Line 19: Float Constant = 3.0
Line 19: Bracket/Parenthesis = )
Line 19: Bracket/Parenthesis = {
Line 20: Identifier = cout
Line 20: Bitwise Operator = <<
Line 20: String Literal = "Average is greater than 3.0"
```

CyberCoder Improve Code Share Code Link Open Website Ln 85, Col 30 Spaces: 4 UTF-16 LE CRLF {} Plain Text AI Code Chat

```
Line 20: Bitwise Operator = <<
Line 20: String Literal = "Average is greater than 3.0"
Line 20: Bitwise Operator = <<
Line 20: Identifier = endl
Line 20: Delimiter = ;
Line 21: Bracket/Parenthesis = }
Line 21: Keyword = else
Line 21: Bracket/Parenthesis = {
Line 22: Identifier = cout
Line 22: Bitwise Operator = <<
Line 22: String Literal = "Average is less than or equal to 3.0"
Line 22: Bitwise Operator = <<
Line 22: Identifier = endl
Line 22: Delimiter = ;
Line 23: Bracket/Parenthesis = }
Line 25: Keyword = return
Line 25: Integer Constant = 0
Line 25: Delimiter = ;
Line 26: Bracket/Parenthesis = }
```

CyberCoder Improve Code Share Code Link Open Website Ln 85, Col 30 Spaces: 4 UTF-16 LE CRLF {} Plain Text AI Code Chat