Group member

YU QIANSHUO : 1210027225
WANG KUN : 1210025783
ZHENG AO : 1220005554

Group member contribution
YU QIANSHUO : DFA implementation
WANG KUN : Scanner implementation
ZHENG AO : Token implementation

Compile
`g++ token.cpp dfa.cpp scanner.cpp -o scanner -std=c++17`

Run
`./scanner while_etc.pc`

Features Implemented
- **Tokenization**: The program reads an input file and tokenizes the input text based on a predefined set of tokens.
- **Error Handling**: The program detects invalid tokens and prints an error message when an error state is encountered.

Scanner Implementation

1. **File Reading**:
    - The program can accept a command-line argument as the input file path and attempts to open the file.

2. **Scanner Initialization**:
    - The `Scanner` class is initialized with an input stream (in this case, a file stream) and stored as a member variable.

3. **Scanning Process**:
    - The `scan` method implements a loop that continuously reads characters from the input stream and calls the `Transition` function (implementation not provided).

4. **Error Handling**:
    - If an error state (`ERROR`) is encountered during scanning, the program prints an error message and terminates scanning early.

5. **End of Scanning**:

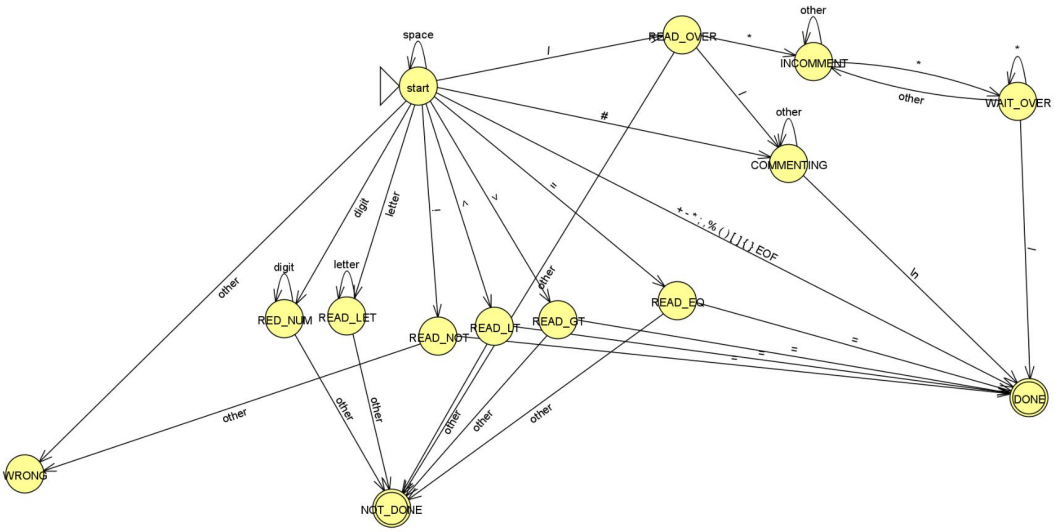
- The scanning loop terminates when the end of the input stream (EOF) is reached.

6. **Token List Creation and Initialization**:
   - In the `main` function, a `Token` object and a `Token_List` object are created to store the tokens generated during the scanning process.

DFA Implementation

1. **DFA Initialization**:
   - The `DFA` class is initialized with a transition table and a set of accepting states.
   - The transition table is implemented as a 2D array of integers, where the row index represents the current state and the column index represents the input character.
2. **Transition Function**:
   - The `Transition` function takes a character as input and updates the current state of the DFA based on the transition table.
   - If the input character is not a valid transition for the current state, the function returns an error state (`ERROR`). Otherwise, it returns the next state.
3. **Accepting State Check**:
   - The `IsAccepting` function checks if the current state of the DFA is an accepting state.
   - If the current state is an accepting state, the function returns the corresponding token type. Otherwise, it returns an error state (`ERROR`).
   - The token type is an enum class that represents the different types of tokens that can be recognized by the DFA.

DFA Graph



Token Implementation

1. **Token Type Definition**:
   - The `TokenType` enum class defines the different types of tokens that can be recognized by

the DFA.

- The token types include keywords, identifiers, operators, separators, and literals.
- The enum class also includes an `ERROR` type to represent invalid tokens.

2. **Token Structure**:

- The `Token` struct stores the type and value of a token.
- The type is of the `TokenType` enum class, and the value is a string representing the token value.
- The struct also includes a `ToString` method to convert the token type to a string for printing.
- The `Token` struct is used to store tokens generated during the scanning process.
- The `Token_List` class stores a list of `Token` objects and provides methods to add tokens to the list and print the list.

Running result

1. **Test file**:

We write a file called while_etc.pc which is full of while statement.

The detailed code is recorded in the while_etc.pc file in the same file directory.

```
while (x > y)
    x = x + 1;   // C-style single-line statement
    y = y - 1;   // Python-style indentation, but still valid syntax

while x > y :
    x = x + 1   # Python-style indentation
    y = y - 1   # Indented, so part of the loop body

while x > y :
{
    x = x + 1;     // C-style block with braces
    y = y - 1;
}
while x > y :
{
    x = x + 1;    // Indentation within braces
        y = y - 1;   // Incorrect indentation but valid C-style syntax
}
while x > y :
    x = x + 1   # No semicolon, newline ends the statement
    y = y - 1   # Another single-line statement without a semicolon

while (x > y) {
    x = x + 1;   // Semicolon allowed but optional
    y = y - 1    // No semicolon needed on this single-line statement
}
while (x < 10) count = count + 1;  // Increment count while x is less than 10
while (x < 10) {
    if (count > 5) break;  // Exit loop if count exceeds 5
    count++;
}
while (x < 10) {
    do {
        x++;
    } while (x < 5);
}
while x < 10:
    count += 1
    if count == 5:
        break  # Exit loop if count reaches 5

while (x < 10) {
    for (int i = 0; i < 5; i++) {
        count += i;  // Nested for loop
    }
    x++;
}
```

2. **Result**:

After running `./scanner while_etc.pc`, we get the final result:

```
Line 0 --- START : Start of the scanner
Line 1 --- WHILE : while
Line 1 --- WS : white space
Line 1 --- LPAREN : (
Line 1 --- ID : x
Line 1 --- WS : white space
Line 1 --- GT : >
Line 1 --- WS : white space
Line 1 --- ID : y
Line 1 --- RPAREN : )
Line 1 --- WS : white space
Line 1 --- ENTER : line break
Line 2 --- WS : white space
Line 2 --- WS : white space
Line 2 --- WS : white space
Line 2 --- WS : white space
Line 2 --- ID : x
Line 2 --- WS : white space
Line 2 --- ASSIGN : =
Line 2 --- WS : white space
Line 2 --- ID : x
Line 2 --- WS : white space
Line 2 --- PLUS : +
Line 2 --- WS : white space
Line 2 --- NUMBER : 1
Line 2 --- SEMICOLON : ;
Line 2 --- WS : white space
Line 2 --- WS : white space
Line 2 --- WS : white space
Line 2 --- COMMENT : // C-style single-line statement
Line 3 --- WS : white space
Line 3 --- WS : white space
Line 3 --- WS : white space
Line 3 --- WS : white space
Line 3 --- ID : y
Line 3 --- WS : white space
Line 3 --- ASSIGN : =
Line 3 --- WS : white space
Line 3 --- ID : y
Line 3 --- WS : white space
Line 3 --- MINUS : -
Line 3 --- WS : white space
Line 3 --- NUMBER : 1
Line 3 --- SEMICOLON : ;
Line 3 --- WS : white space
Line 3 --- WS : white space
Line 3 --- WS : white space
Line 3 --- COMMENT : // Python-style indentation, but still valid syntax
Line 4 --- WS : white space
Line 4 --- WS : white space
Line 4 --- WS : white space
Line 4 --- WS : white space
Line 4 --- ENTER : line break
Line 5 --- WHILE : while
Line 5 --- WS : white space
Line 5 --- ID : x
Line 5 --- WS : white space
Line 5 --- GT : >
Line 5 --- WS : white space
Line 5 --- ID : y
Line 5 --- WS : white space
Line 5 --- COLON : :
Line 5 --- ENTER : line break
Line 6 --- WS : white space
```

Taking the first behavior as an example, the scanner reads each character and stores the first token：while after reading the first word while. Then continue to read each character backwards. Skip to the next line when reading ENTER.

When the program reads an = in the second line and continues to read backwards, it finds that the next character is not =, so it records the last token as =. If the next character is also =, record the token as = =.

Program pair "> <!" The processing of is similar to the processing of =, and the symbols such as "+,-,*" are recorded directly. Multiple treatments in the output result can verify this rule.

When the program reads a number 1 in the second line and then continues to read backwards, it finds that the next character is not a number, so it records the last token as the number 1.

The last token in the third line records a comment. Scanner records a/first. Since the grammar requires that both//and/* symbols can be used as comment, we wait for the next character to be read. Finally, a comment of type//was found, so the token was recorded.

The remaining results are similar to those before. The results are as follows:

```
Line 6 --- WS : white space
Line 6 --- WS : white space
Line 6 --- WS : white space
Line 6 --- WS : white space
Line 6 --- ID : x
Line 6 --- WS : white space
Line 6 --- ASSIGN : =
Line 6 --- WS : white space
Line 6 --- ID : x
Line 6 --- WS : white space
Line 6 --- PLUS : +
Line 6 --- WS : white space
Line 6 --- NUMBER : 1
Line 6 --- WS : white space
Line 6 --- WS : white space
Line 6 --- WS : white space
Line 6 --- COMMENT : # Python-style indentation
Line 7 --- WS : white space
Line 7 --- WS : white space
Line 7 --- WS : white space
Line 7 --- WS : white space
Line 7 --- ID : y
Line 7 --- WS : white space
Line 7 --- ASSIGN : =
Line 7 --- WS : white space
Line 7 --- ID : y
Line 7 --- WS : white space
Line 7 --- MINUS : -
Line 7 --- WS : white space
Line 7 --- NUMBER : 1
Line 7 --- WS : white space
Line 7 --- WS : white space
Line 7 --- WS : white space
Line 7 --- COMMENT : # Indented, so part of the loop body
Line 8 --- WS : white space
Line 8 --- WS : white space
Line 8 --- WS : white space
Line 8 --- WS : white space
Line 8 --- ENTER : line break
Line 9 --- WHILE : while
Line 9 --- WS : white space
Line 9 --- ID : x
Line 9 --- WS : white space
Line 9 --- GT : >
Line 9 --- WS : white space
Line 9 --- ID : y
Line 9 --- WS : white space
Line 9 --- COLON : :
Line 9 --- WS : white space
Line 9 --- ENTER : line break
Line 10 --- LBRACE : {
```

```
Line 10 --- LBRACE : {
Line 10 --- WS : white space
Line 10 --- ENTER : line break
Line 11 --- WS : white space
Line 11 --- WS : white space
Line 11 --- WS : white space
Line 11 --- WS : white space
Line 11 --- ID : x
Line 11 --- WS : white space
Line 11 --- ASSIGN : =
Line 11 --- WS : white space
Line 11 --- ID : x
Line 11 --- WS : white space
Line 11 --- PLUS : +
Line 11 --- WS : white space
Line 11 --- NUMBER : 1
Line 11 --- SEMICOLON : ;
Line 11 --- WS : white space
Line 11 --- WS : white space
Line 11 --- WS : white space
Line 11 --- WS : white space
Line 11 --- WS : white space
Line 11 --- COMMENT : // C-style block with braces
Line 12 --- WS : white space
Line 12 --- WS : white space
Line 12 --- WS : white space
Line 12 --- WS : white space
Line 12 --- ID : y
Line 12 --- WS : white space
Line 12 --- ASSIGN : =
Line 12 --- WS : white space
Line 12 --- ID : y
Line 12 --- WS : white space
Line 12 --- MINUS : -
Line 12 --- WS : white space
Line 12 --- NUMBER : 1
Line 12 --- SEMICOLON : ;
Line 12 --- WS : white space
Line 12 --- ENTER : line break
Line 13 --- RBRACE : }
Line 13 --- ENTER : line break
Line 14 --- WHILE : while
Line 14 --- WS : white space
Line 14 --- ID : x
Line 14 --- WS : white space
Line 14 --- GT : >
Line 14 --- WS : white space
Line 14 --- ID : y
Line 14 --- WS : white space
Line 14 --- COLON : :
Line 14 --- ENTER : line break
Line 15 --- LBRACE : {
Line 15 --- ENTER : line break
Line 16 --- WS : white space
Line 16 --- WS : white space
Line 16 --- WS : white space
Line 16 --- WS : white space
Line 16 --- ID : x
Line 16 --- WS : white space
Line 16 --- ASSIGN : =
Line 16 --- WS : white space
Line 16 --- ID : x
Line 16 --- WS : white space
Line 16 --- PLUS : +
Line 16 --- WS : white space
Line 16 --- NUMBER : 1
Line 16 --- SEMICOLON : ;
Line 16 --- WS : white space
Line 16 --- WS : white space
Line 16 --- WS : white space
Line 16 --- WS : white space
Line 16 --- COMMENT : // Indentation within braces
```

```
Line 17 --- WS : white space
Line 17 --- WS : white space
Line 17 --- WS : white space
Line 17 --- WS : white space
Line 17 --- WS : white space
Line 17 --- WS : white space
Line 17 --- WS : white space
Line 17 --- WS : white space
Line 17 --- ID : y
Line 17 --- WS : white space
Line 17 --- ASSIGN : =
Line 17 --- WS : white space
Line 17 --- ID : y
Line 17 --- WS : white space
Line 17 --- MINUS : -
Line 17 --- WS : white space
Line 17 --- NUMBER : 1
Line 17 --- SEMICOLON : ;
Line 17 --- WS : white space
Line 17 --- WS : white space
Line 17 --- WS : white space
Line 17 --- COMMENT : // Incorrect indentation but valid C-style syntax
Line 18 --- RBRACE : }
Line 18 --- ENTER : line break
Line 19 --- WHILE : while
Line 19 --- WS : white space
Line 19 --- ID : x
Line 19 --- WS : white space
Line 19 --- GT : >
Line 19 --- WS : white space
Line 19 --- ID : y
Line 19 --- WS : white space
Line 19 --- COLON : :
Line 19 --- ENTER : line break
Line 20 --- WS : white space
Line 20 --- WS : white space
Line 20 --- WS : white space
Line 20 --- WS : white space
Line 20 --- ID : x
Line 20 --- WS : white space
Line 20 --- ASSIGN : =
Line 20 --- WS : white space
Line 20 --- ID : x
Line 20 --- WS : white space
Line 20 --- PLUS : +
Line 20 --- WS : white space
Line 20 --- NUMBER : 1
Line 20 --- WS : white space
Line 20 --- WS : white space
Line 20 --- WS : white space
Line 20 --- COMMENT : # No semicolon, newline ends the statement
Line 21 --- WS : white space
Line 21 --- WS : white space
Line 21 --- WS : white space
Line 21 --- WS : white space
Line 21 --- ID : y
Line 21 --- WS : white space
Line 21 --- ASSIGN : =
Line 21 --- WS : white space
Line 21 --- ID : y
Line 21 --- WS : white space
Line 21 --- MINUS : -
Line 21 --- WS : white space
Line 21 --- NUMBER : 1
Line 21 --- WS : white space
Line 21 --- WS : white space
Line 21 --- WS : white space
Line 21 --- COMMENT : # Another single-line statement without a semicolon
```

```
Line 22 --- WS : white space
Line 22 --- WS : white space
Line 22 --- WS : white space
Line 22 --- WS : white space
Line 22 --- ENTER : line break
Line 23 --- WHILE : while
Line 23 --- WS : white space
Line 23 --- LPAREN : (
Line 23 --- ID : x
Line 23 --- WS : white space
Line 23 --- GT : >
Line 23 --- WS : white space
Line 23 --- ID : y
Line 23 --- RPAREN : )
Line 23 --- WS : white space
Line 23 --- LBRACE : {
Line 23 --- ENTER : line break
Line 24 --- WS : white space
Line 24 --- WS : white space
Line 24 --- WS : white space
Line 24 --- WS : white space
Line 24 --- ID : x
Line 24 --- WS : white space
Line 24 --- ASSIGN : =
Line 24 --- WS : white space
Line 24 --- ID : x
Line 24 --- WS : white space
Line 24 --- PLUS : +
Line 24 --- WS : white space
Line 24 --- NUMBER : 1
Line 24 --- SEMICOLON : ;
Line 24 --- WS : white space
Line 24 --- WS : white space
Line 24 --- WS : white space
Line 24 --- COMMENT : // Semicolon allowed but optional
Line 25 --- WS : white space
Line 25 --- WS : white space
Line 25 --- WS : white space
Line 25 --- WS : white space
Line 25 --- ID : y
Line 25 --- WS : white space
Line 25 --- ASSIGN : =
Line 25 --- WS : white space
Line 25 --- ID : y
Line 25 --- WS : white space
Line 25 --- MINUS : -
Line 25 --- WS : white space
Line 25 --- NUMBER : 1
Line 25 --- WS : white space
Line 25 --- WS : white space
Line 25 --- WS : white space
Line 25 --- WS : white space
Line 25 --- COMMENT : // No semicolon needed on this single-line statement
Line 26 --- RBRACE : }
Line 26 --- ENTER : line break
Line 27 --- WHILE : while
Line 27 --- WS : white space
Line 27 --- LPAREN : (
Line 27 --- ID : x
Line 27 --- WS : white space
Line 27 --- LT : <
Line 27 --- WS : white space
Line 27 --- NUMBER : 10
Line 27 --- RPAREN : )
Line 27 --- WS : white space
Line 27 --- ID : count
Line 27 --- WS : white space
Line 27 --- ASSIGN : =
Line 27 --- WS : white space
Line 27 --- ID : count
Line 27 --- WS : white space
Line 27 --- PLUS : +
Line 27 --- WS : white space
Line 27 --- NUMBER : 1
Line 27 --- SEMICOLON : ;
Line 27 --- WS : white space
Line 27 --- WS : white space
Line 27 --- COMMENT : // Increment count while x is less than 10
```

```
Line 28 --- WHILE : while
Line 28 --- WS : white space
Line 28 --- LPAREN : (
Line 28 --- ID : x
Line 28 --- WS : white space
Line 28 --- LT : <
Line 28 --- WS : white space
Line 28 --- NUMBER : 10
Line 28 --- RPAREN : )
Line 28 --- WS : white space
Line 28 --- LBRACE : {
Line 28 --- ENTER : line break
Line 29 --- WS : white space
Line 29 --- WS : white space
Line 29 --- WS : white space
Line 29 --- WS : white space
Line 29 --- IF : if
Line 29 --- WS : white space
Line 29 --- LPAREN : (
Line 29 --- ID : count
Line 29 --- WS : white space
Line 29 --- GT : >
Line 29 --- WS : white space
Line 29 --- NUMBER : 5
Line 29 --- RPAREN : )
Line 29 --- WS : white space
Line 29 --- ID : break
Line 29 --- SEMICOLON : ;
Line 29 --- WS : white space
Line 29 --- WS : white space
Line 29 --- COMMENT : // Exit loop if count exceeds 5
Line 30 --- WS : white space
Line 30 --- WS : white space
Line 30 --- WS : white space
Line 30 --- WS : white space
Line 30 --- ID : count
Line 30 --- PLUS : +
Line 30 --- PLUS : +
Line 30 --- SEMICOLON : ;
Line 30 --- ENTER : line break
Line 31 --- RBRACE : }
Line 31 --- ENTER : line break
Line 32 --- WHILE : while
Line 32 --- WS : white space
Line 32 --- LPAREN : (
Line 32 --- ID : x
Line 32 --- WS : white space
Line 32 --- LT : <
Line 32 --- WS : white space
Line 32 --- NUMBER : 10
Line 32 --- RPAREN : )
Line 32 --- WS : white space
Line 32 --- LBRACE : {
Line 32 --- ENTER : line break
Line 33 --- WS : white space
Line 33 --- WS : white space
Line 33 --- WS : white space
Line 33 --- WS : white space
Line 33 --- ID : do
Line 33 --- WS : white space
Line 33 --- LBRACE : {
Line 33 --- ENTER : line break
Line 34 --- WS : white space
Line 34 --- WS : white space
Line 34 --- WS : white space
Line 34 --- WS : white space
Line 34 --- WS : white space
Line 34 --- WS : white space
Line 34 --- WS : white space
Line 34 --- WS : white space
Line 34 --- ID : x
Line 34 --- PLUS : +
Line 34 --- PLUS : +
Line 34 --- SEMICOLON : ;
Line 34 --- ENTER : line break
Line 35 --- WS : white space
Line 35 --- WS : white space
Line 35 --- WS : white space
Line 35 --- WS : white space
Line 35 --- RBRACE : }
Line 35 --- WS : white space
Line 35 --- WHILE : while
Line 35 --- WS : white space
Line 35 --- LPAREN : (
Line 35 --- ID : x
Line 35 --- WS : white space
Line 35 --- LT : <
Line 35 --- WS : white space
Line 35 --- NUMBER : 5
Line 35 --- RPAREN : )
Line 35 --- SEMICOLON : ;
Line 35 --- ENTER : line break
Line 36 --- RBRACE : }
Line 36 --- ENTER : line break
Line 37 --- WHILE : while
Line 37 --- WS : white space
Line 37 --- ID : x
Line 37 --- WS : white space
Line 37 --- LT : <
Line 37 --- WS : white space
Line 37 --- NUMBER : 10
Line 37 --- COLON : :
Line 37 --- ENTER : line break
Line 38 --- WS : white space
Line 38 --- WS : white space
Line 38 --- WS : white space
Line 38 --- WS : white space
Line 38 --- ID : count
Line 38 --- WS : white space
Line 38 --- PLUS : +
Line 38 --- ASSIGN : =
Line 38 --- WS : white space
Line 38 --- NUMBER : 1
Line 38 --- ENTER : line break
```

```
Line 39 --- WS : white space
Line 39 --- WS : white space
Line 39 --- WS : white space
Line 39 --- WS : white space
Line 39 --- IF : if
Line 39 --- WS : white space
Line 39 --- ID : count
Line 39 --- WS : white space
Line 39 --- EQ : ==
Line 39 --- WS : white space
Line 39 --- NUMBER : 5
Line 39 --- COLON : :
Line 39 --- ENTER : line break
Line 40 --- WS : white space
Line 40 --- WS : white space
Line 40 --- WS : white space
Line 40 --- WS : white space
Line 40 --- WS : white space
Line 40 --- WS : white space
Line 40 --- WS : white space
Line 40 --- WS : white space
Line 40 --- ID : break
Line 40 --- WS : white space
Line 40 --- WS : white space
Line 40 --- COMMENT : # Exit loop if count reaches 5
Line 41 --- WS : white space
Line 41 --- WS : white space
Line 41 --- WS : white space
Line 41 --- WS : white space
Line 41 --- WS : white space
Line 41 --- WS : white space
Line 41 --- WS : white space
Line 41 --- ENTER : line break
Line 42 --- WHILE : while
Line 42 --- WS : white space
Line 42 --- LPAREN : (
Line 42 --- ID : x
Line 42 --- WS : white space
Line 42 --- LT : <
Line 42 --- WS : white space
Line 42 --- NUMBER : 10
Line 42 --- RPAREN : )
Line 42 --- WS : white space
Line 42 --- LBRACE : {
Line 42 --- ENTER : line break
Line 43 --- WS : white space
Line 43 --- WS : white space
Line 43 --- WS : white space
Line 43 --- WS : white space
Line 43 --- ID : for
Line 43 --- WS : white space
Line 43 --- LPAREN : (
Line 43 --- INT : int
Line 43 --- WS : white space
Line 43 --- ID : i
Line 43 --- WS : white space
Line 43 --- ASSIGN : =
Line 43 --- WS : white space
Line 43 --- NUMBER : 0
Line 43 --- SEMICOLON : ;
Line 43 --- WS : white space
Line 43 --- ID : i
Line 43 --- WS : white space
Line 43 --- LT : <
Line 43 --- WS : white space
Line 43 --- NUMBER : 5
Line 43 --- SEMICOLON : ;
Line 43 --- WS : white space
Line 43 --- ID : i
Line 43 --- PLUS : +
Line 43 --- PLUS : +
Line 43 --- RPAREN : )
Line 43 --- WS : white space
Line 43 --- LBRACE : {
Line 43 --- ENTER : line break
Line 44 --- WS : white space
Line 44 --- WS : white space
```

```
Line 44 --- WS : white space
Line 44 --- WS : white space
Line 44 --- WS : white space
Line 44 --- WS : white space
Line 44 --- WS : white space
Line 44 --- WS : white space
Line 44 --- WS : white space
Line 44 --- WS : white space
Line 44 --- ID : count
Line 44 --- WS : white space
Line 44 --- PLUS : +
Line 44 --- ASSIGN : =
Line 44 --- WS : white space
Line 44 --- ID : i
Line 44 --- SEMICOLON : ;
Line 44 --- WS : white space
Line 44 --- WS : white space
Line 44 --- COMMENT : // Nested for loop
Line 45 --- WS : white space
Line 45 --- WS : white space
Line 45 --- WS : white space
Line 45 --- WS : white space
Line 45 --- RBRACE : }
Line 45 --- ENTER : line break
Line 46 --- WS : white space
Line 46 --- WS : white space
Line 46 --- WS : white space
Line 46 --- WS : white space
Line 46 --- ID : x
Line 46 --- PLUS : +
Line 46 --- PLUS : +
Line 46 --- SEMICOLON : ;
Line 46 --- ENTER : line break
Line 47 --- RBRACE : }
Line 47 --- ENTER : line break
Line 48 --- ENTER : line break
Line 49 --- ENDFILE : The end of file
PS D:\compiler\compiler-main>
```