**Question # 5**: Explain this function which perform semantics analysis in this mini compiler?  
Ans: The **Semantic Analysis** function in the mini compiler is responsible for checking the correctness of the syntax and ensuring that the expression makes sense logically. This phase doesn't deal with the structure of the code (which is handled by the **Parser**), but rather ensures that the parsed tokens follow the rules of the language and the operation being performed is valid.  
**Purpose**:

* The **Semantic Analysis** phase ensures that the tokens represent a valid, meaningful expression. It checks for issues such as:
  + Type mismatches (e.g., trying to add a string to a number).
  + Invalid operations (e.g., division by zero).
  + Correct use of operators and operands.
  + Ensuring that the expression follows the intended logic.

**Implementation**:

* In the current code, the **Semantic Analysis** function is a placeholder and does not yet contain any specific logic.
* However, it can be extended to handle various semantic checks, such as:
  + **Checking for undefined variables**: If the expression involves variables, you can check whether they are defined or initialized.
  + **Checking for type errors**: For example, ensuring that both operands of an addition operation are numbers.
  + **Checking for division by zero**: Ensuring that no division operation has a zero denominator.
  + **Parentheses matching**: Ensuring that every opening parenthesis has a corresponding closing parenthesis.
  + **Operator validation**: Ensuring that operators like +, -, \*, and / are used correctly (e.g., ensuring you don't use a + operator between two non-numeric values).

Code : static void SemanticAnalysis(List<Token> tokens)

{

for (int i = 0; i < tokens.Count; i++)

{

// Check for division by zero

if (tokens[i].Type == "DIVIDE" && i + 1 < tokens.Count && tokens[i + 1].Type == "NUMBER" && tokens[i + 1].Value == "0")

{

throw new Exception("Error: Division by zero is not allowed.");

}

// Check for invalid operators

if (tokens[i].Type == "PLUS" || tokens[i].Type == "TIMES")

{

if (i == 0 || i == tokens.Count - 1 || tokens[i - 1].Type != "NUMBER" || tokens[i + 1].Type != "NUMBER")

{

throw new Exception($"Error: Invalid operator '{tokens[i].Value}' at position {i}.");

}

}

}

}