****

**COMSATS University Islamabad (CUI) Attock Campus**

Compiler Construction Lab  
(Semester Project)

MiniCompiler

***By***

**Kashif Hussain CIIT/FA20-BCS-019/ATK**

***Submitted To:*Mr. Syed Bilal Haider**

Semester Project Report :

MiniCompiler

Q4:

*How functions works?Step by step*

### Scanner Class

#### public Token GetNextToken()

1. **Input:** None (called without parameters)
2. **Processing Steps:**
   * Checks if the current position is at or beyond the end of the input string.
   * If true, returns an EndOfFile token.
   * Otherwise, retrieves the current character and increments the position.
   * Switches on the current character:
     + If it's an operator or parenthesis, returns a corresponding token.
     + If it's a digit, calls **ScanNumber** to parse a number token.
     + If it's a letter, calls **ScanIdentifier** to parse an identifier token.
     + Otherwise, returns an Error token for unexpected characters.

#### private Token ScanNumber(char firstDigit)

1. **Input:** The first digit of the number.
2. **Processing Steps:**
   * Initializes a string with the first digit.
   * Iterates through consecutive digits in the input, appending them to the string.
   * Returns a Number token with the parsed numeric value.

#### private Token ScanIdentifier(char firstLetter)

1. **Input:** The first letter of the identifier.
2. **Processing Steps:**
   * Initializes a string with the first letter.
   * Iterates through consecutive letters, digits, or underscores in the input, appending them to the string.
   * Returns an Identifier token with the parsed identifier.

#### private void ThrowUnexpectedCharacter(char character, int position)

1. **Input:** The unexpected character and its position.
2. **Processing Steps:**
   * Throws an exception with an error message containing the unexpected character and its position.

### Parser Class

#### public void Parse()

1. **Input:** None (called without parameters).
2. **Processing Steps:**
   * Calls the **Expression** method.
   * Outputs a success message.

#### private void Expression()

1. **Input:** None (called without parameters).
2. **Processing Steps:**
   * Calls the **Term** method.
   * While the current token is a Plus or Minus operator, processes the operator, calls **Match** method, and then calls **Term**.
   * Outputs a message for each performed operation.

#### private void Term()

1. **Input:** None (called without parameters).
2. **Processing Steps:**
   * Calls the **Factor** method.
   * While the current token is a Multiply or Divide operator, processes the operator, calls **Match** method, and then calls **Factor**.
   * Outputs a message for each performed operation.

#### private void Factor()

1. **Input:** None (called without parameters).
2. **Processing Steps:**
   * Checks if the current token is a Number or Identifier, generates code to load it into the register, and calls **Match**.
   * If the current token is a LeftParen, processes the parenthesized expression.
   * If the current token is an Error, handles the error, outputs a message, and moves to the next token.

#### private void Match(TokenType expectedToken)

1. **Input:** The expected token type.
2. **Processing Steps:**
   * Compares the expected token type with the current token type.
   * If they match, calls **GetNextToken** to move to the next token.
   * Otherwise, throws an exception.

### Main Program

#### static void Main()

1. **Input:** None (called without parameters).
2. **Processing Steps:**
   * Prompts the user to enter the source code.
   * Reads the user input and initializes a **Scanner** and **Parser**.
   * Calls **Parse** method on the **Parser**.

These functions collectively perform lexical analysis, syntax analysis, and basic code generation for arithmetic expressions. The main program serves as an entry point, allowing users to input source code for parsing.