LAB MANUAL

Integration: Lexical Analyzer and Symbol Table (Phase-1)
Course: Compiler Construction
Lab: Integration of Lexical Analyzer with Symbol Table
Instructor: Mr. Waheed Ahmad Khan

Activity Overview

The objective of this lab is to implement the integration between the **Lexical Analyzer** and the **Symbol Table**, enabling the compiler to not only generate tokens but also store essential information about identifiers.

By the end of this lab, students will be able to:

- Understand how lexical analysis interacts with the symbol table.
- Generate tokens from source code.
- Insert identifier information (name, type, line number, and value) into the symbol table.

Reference: Read Section **6.4** from the book *"Compiler Construction — Principles and Practices" by Kenneth C. Louden*.

---

1) Useful Concepts

A **Lexical Analyzer** performs the following tasks:

- Reads the source code character by character.
- Groups characters to form **valid tokens** such as identifiers, numbers, keywords, operators, etc.
- Sends generated tokens to the parser.
- Also interacts with the **Symbol Table** to store variables and their properties:
- Name
- Data type
- Line number
- (Optional) Initial value

The Symbol Table plays a major role in later phases like syntax analysis, semantic analysis, and intermediate code generation.

Activity 1: Implementing a Basic Lexical Analyzer

Goal:

Generate tokens from a simple input program and detect identifiers. When an identifier is detected, store it in the symbol table.

Example Input:

```
int x = 10;
float y;
x = x + y;
```

Expected Token Output:

```
<KEYWORD, int>
<ID, x>
<ASSIGN, =>
<NUM, 10>
<SEMICOLON, ;>
...
```

Sample Code (C#): Lexical Analyzer + Symbol Table Integration

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;

class LexicalAnalyzer
{
    static void Main()
    {
        string code = "int x = 10; float y; x = x + y;";

        // Symbol Table: variable → (datatype, line number, value)
        Dictionary<string, (string type, int line, string value)> symbolTable =
            new Dictionary<string, (string, int, string)>();

        string[] lines = code.Split(';');
        int lineNumber = 1;

        foreach (var line in lines)
```

```
        {
            string trimmed = line.Trim();
            if (trimmed == "") continue;

            Console.WriteLine($"Processing Line {lineNumber}: {trimmed}");

            // Patterns
            string keywordPattern = @"\b(int|float|char)\b";
            string identifierPattern = @"[a-zA-Z_][a-zA-Z0-9_]*";
            string numberPattern = @"\b[0-9]+\b";

            // Tokenization
            foreach (Match match in Regex.Matches(trimmed, keywordPattern))
                Console.WriteLine($"<KEYWORD, {match.Value}>");

            foreach (Match match in Regex.Matches(trimmed, numberPattern))
                Console.WriteLine($"<NUM, {match.Value}>");

            foreach (Match match in Regex.Matches(trimmed, identifierPattern))
            {
                string id = match.Value;

                // Insert identifier into symbol table only once
                if (!symbolTable.ContainsKey(id) && id != "int" && id !=
"float" && id != "char")
                {
                    symbolTable.Add(id, ("unknown", lineNumber, "undefined"));
                }

                Console.WriteLine($"<ID, {id}>");
            }

            lineNumber++;
        }

        Console.WriteLine("\n--- SYMBOL TABLE ---");
        foreach (var entry in symbolTable)
        {
            Console.WriteLine($"{entry.Key}  →  Type: {entry.Value.type}, Line:
{entry.Value.line}, Value: {entry.Value.value}");
        }

        Console.ReadLine();
    }
}
```

Activity 2: Updating Symbol Table with Data Types and Values

Goal:

Enhance the lexical analyzer to:

- Detect data type declarations.
- Update the type and initial value in the symbol table.

Example:

```
int x = 10;
float y;
```

Expected Symbol Table:

```
x → Type: int, Value: 10, Line: 1
y → Type: float, Value: undefined, Line: 2
```

Code Snippet for Activity 2:

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;

class Activity2
{
    static void Main()
    {
        // Input source code
        string code = "int x = 10; float y; char c = 'A';";

        // Symbol Table: variable → (datatype, line number, value)
        Dictionary<string, (string type, int line, string value)> symbolTable =
            new Dictionary<string, (string, int, string)>();

        // Split input into lines based on ;
        string[] lines = code.Split(';');
        int lineNumber = 1;

        foreach (var line in lines)
        {
```

```
            string trimmed = line.Trim();
            if (trimmed == "") continue;

            Console.WriteLine($"\nProcessing Line {lineNumber}: {trimmed}");

            // Check if line starts with a declaration
            if (trimmed.StartsWith("int") || trimmed.StartsWith("float") ||
trimmed.StartsWith("char"))
            {
                string[] parts = trimmed.Split(' ');
                string type = parts[0];       // int / float / char
                string variablePart = parts[1];

                // Case 1: variable with initialization
                if (variablePart.Contains("="))
                {
                    string[] assignParts = variablePart.Split('=');
                    string varName = assignParts[0];
                    string value = assignParts[1];

                    symbolTable[varName] = (type, lineNumber, value);

                    Console.WriteLine($"Declared: {varName}, Type: {type},
Value: {value}");
                }
                else
                {
                    // Case 2: variable without initialization
                    string varName = variablePart;

                    symbolTable[varName] = (type, lineNumber, "undefined");

                    Console.WriteLine($"Declared: {varName}, Type: {type}, No
initial value");
                }
            }

            lineNumber++;
        }

        // Final Symbol Table Output
        Console.WriteLine("\n--- FINAL SYMBOL TABLE ---");
        foreach (var entry in symbolTable)
        {
            Console.WriteLine($"{entry.Key} → Type: {entry.Value.type}, Line:
{entry.Value.line}, Value: {entry.Value.value}");
        }
```

```
        Console.ReadLine();
    }
}
```

Graded Lab Task

You are required to write a C# program that accomplishes **both**:

Part (a): Token Generation & Symbol Table Construction

Your program must:

1. Read simple multi-line source code.
2. Generate tokens for:
3. Keywords
4. Identifiers
5. Operators
6. Numbers
7. Build a symbol table containing for each identifier:
8. Name
9. Type
10. Line number
11. Initial value
12. Display tokens and the symbol table.

Output Requirement:

- If a variable is declared and assigned → update value.
- If a variable is used before declaration → print an error.

Part (b): Identifier Validation Using Symbol Table

Your program must:

1. Read an assignment expression like:

```
x = x + y;
```

1. For each identifier used:
2. Verify it exists in the symbol table.
3. If missing → print error.

Example Output:

```
<ID, x>
<ID, y>
Error: Variable 'y' used before declaration.
```

Expected Final Output

Your program should show:

- All tokens generated
- A complete symbol table
- Any identifier-related errors

End of Manual