Question3:

using System;

using System.Collections.Generic;

namespace SymbolTableWithPalindromeCheck

{

class Program

{

static void Main(string[] args)

{

SymbolTable symbolTable = new SymbolTable();

int lineNumber = 1;

Console.WriteLine("Symbol Table with Palindrome Check");

Console.WriteLine("Enter variable declarations (empty line to exit):");

Console.WriteLine("Format: <type> <name> = <value>; Example: int val33 = 999;");

while (true)

{

Console.Write($"[Line {lineNumber}]> ");

string input = Console.ReadLine().Trim();

if (string.IsNullOrEmpty(input))

break;

try

{

// Parse the input

string[] parts = input.Split(new[] { ' ', '=', ';' }, StringSplitOptions.RemoveEmptyEntries);

if (parts.Length != 3)

{

Console.WriteLine("Invalid format. Use: <type> <name> = <value>;");

continue;

}

string type = parts[0];

string name = parts[1];

string value = parts[2];

// Add to symbol table if valid palindrome substring exists

var result = symbolTable.AddVariable(name, type, value, lineNumber);

if (result.success)

{

Console.WriteLine($"Added: {name} (palindrome: '{result.palindrome}')");

}

else

{

Console.WriteLine($"Rejected: {name} (no palindrome substring ≥3)");

}

}

catch (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

}

lineNumber++;

}

// Display symbol table

Console.WriteLine("\nSymbol Table Contents:");

Console.WriteLine("------------------------------------------------");

Console.WriteLine("| Line # | Variable Name | Type | Value | Palindrome |");

Console.WriteLine("------------------------------------------------");

symbolTable.Display();

Console.WriteLine("------------------------------------------------");

}

}

class SymbolTable

{

private readonly List<SymbolEntry> \_entries = new List<SymbolEntry>();

public (bool success, string palindrome) AddVariable(string name, string type, string value, int lineNumber)

{

string palindrome = FindPalindromeSubstring(name, 3);

if (palindrome != null)

{

\_entries.Add(new SymbolEntry(name, type, value, lineNumber, palindrome));

return (true, palindrome);

}

return (false, null);

}

public void Display()

{

foreach (var entry in \_entries)

{

Console.WriteLine($"| {entry.LineNumber,-6} | {entry.Name,-13} | {entry.Type,-6} | {entry.Value,-9} | {entry.Palindrome,-10} |");

}

}

private string FindPalindromeSubstring(string s, int minLength)

{

for (int len = s.Length; len >= minLength; len--)

{

for (int i = 0; i <= s.Length - len; i++)

{

string substring = s.Substring(i, len);

if (IsPalindrome(substring))

{

return substring;

}

}

}

return null;

}

private bool IsPalindrome(string s)

{

int left = 0;

int right = s.Length - 1;

while (left < right)

{

if (s[left] != s[right])

{

return false;

}

left++;

right--;

}

return true;

}

}

class SymbolEntry

{

public string Name { get; }

public string Type { get; }

public string Value { get; }

public int LineNumber { get; }

public string Palindrome { get; }

public SymbolEntry(string name, string type, string value, int lineNumber, string palindrome)

{

Name = name;

Type = type;

Value = value;

LineNumber = lineNumber;

Palindrome = palindrome;

}

}

}

**Output:**

