using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

namespace SymbolTableApp

{

// Represents a single symbol table entry

class SymbolTableEntry

{

public string Type { get; }

public string Name { get; }

public string Value { get; }

public int LineNumber { get; }

public SymbolTableEntry(string type, string name, string value, int lineNumber)

{

Type = type;

Name = name;

Value = value;

LineNumber = lineNumber;

}

}

// Handles validation logic like syntax and palindrome checking

class Validator

{

private static readonly Regex DeclarationPattern =

new Regex(@"^(int|float|char|string)\s+([a-zA-Z\_]\w\*)\s\*=\s\*(.+);$");

public static bool IsValidDeclaration(string input, out string type, out string name, out string value)

{

var match = DeclarationPattern.Match(input);

if (match.Success)

{

type = match.Groups[1].Value;

name = match.Groups[2].Value;

value = match.Groups[3].Value.Trim();

return true;

}

type = name = value = null;

return false;

}

public static bool ContainsPalindromeSubstring(string text)

{

for (int i = 0; i < text.Length; i++)

{

for (int j = i + 2; j < text.Length; j++)

{

var substr = text.Substring(i, j - i + 1);

if (IsPalindrome(substr))

return true;

}

}

return false;

}

private static bool IsPalindrome(string input)

{

int left = 0, right = input.Length - 1;

while (left < right)

{

if (input[left++] != input[right--])

return false;

}

return true;

}

}

// Manages symbol table operations

class SymbolTableManager

{

private readonly List<SymbolTableEntry> entries = new();

private int lineCounter = 1;

public void Start()

{

Console.WriteLine("=== Symbol Table Builder ===");

Console.WriteLine("Enter variable declarations (e.g., int abcba = 123;)");

Console.WriteLine("Type 'exit' to finish input.\n");

while (true)

{

Console.Write($"Line {lineCounter}: ");

var input = Console.ReadLine()?.Trim();

if (string.Equals(input, "exit", StringComparison.OrdinalIgnoreCase))

break;

ProcessLine(input);

lineCounter++;

}

DisplaySymbolTable();

}

private void ProcessLine(string input)

{

if (Validator.IsValidDeclaration(input, out string type, out string name, out string value))

{

if (Validator.ContainsPalindromeSubstring(name))

{

entries.Add(new SymbolTableEntry(type, name, value, lineCounter));

}

else

{

Console.WriteLine("✘ Skipped: Variable name does not contain a palindrome substring of length ≥ 3.");

}

}

else

{

Console.WriteLine("✘ Invalid syntax. Format must be: <type> <varName> = <value>;");

}

}

private void DisplaySymbolTable()

{

Console.WriteLine("\n--- Symbol Table ---");

Console.WriteLine("{0,-10} | {1,-15} | {2,-10} | {3}", "Type", "Variable", "Value", "Line");

Console.WriteLine(new string('-', 55));

if (entries.Count == 0)

{

Console.WriteLine("No valid entries found.");

return;

}

foreach (var entry in entries)

{

Console.WriteLine($"{entry.Type,-10} | {entry.Name,-15} | {entry.Value,-10} | {entry.LineNumber}");

}

}

}

class Program

{

static void Main()

{

var manager = new SymbolTableManager();

manager.Start();

Console.WriteLine("\nProgram ended. Press any key to exit...");

Console.ReadKey();

}

}

}

