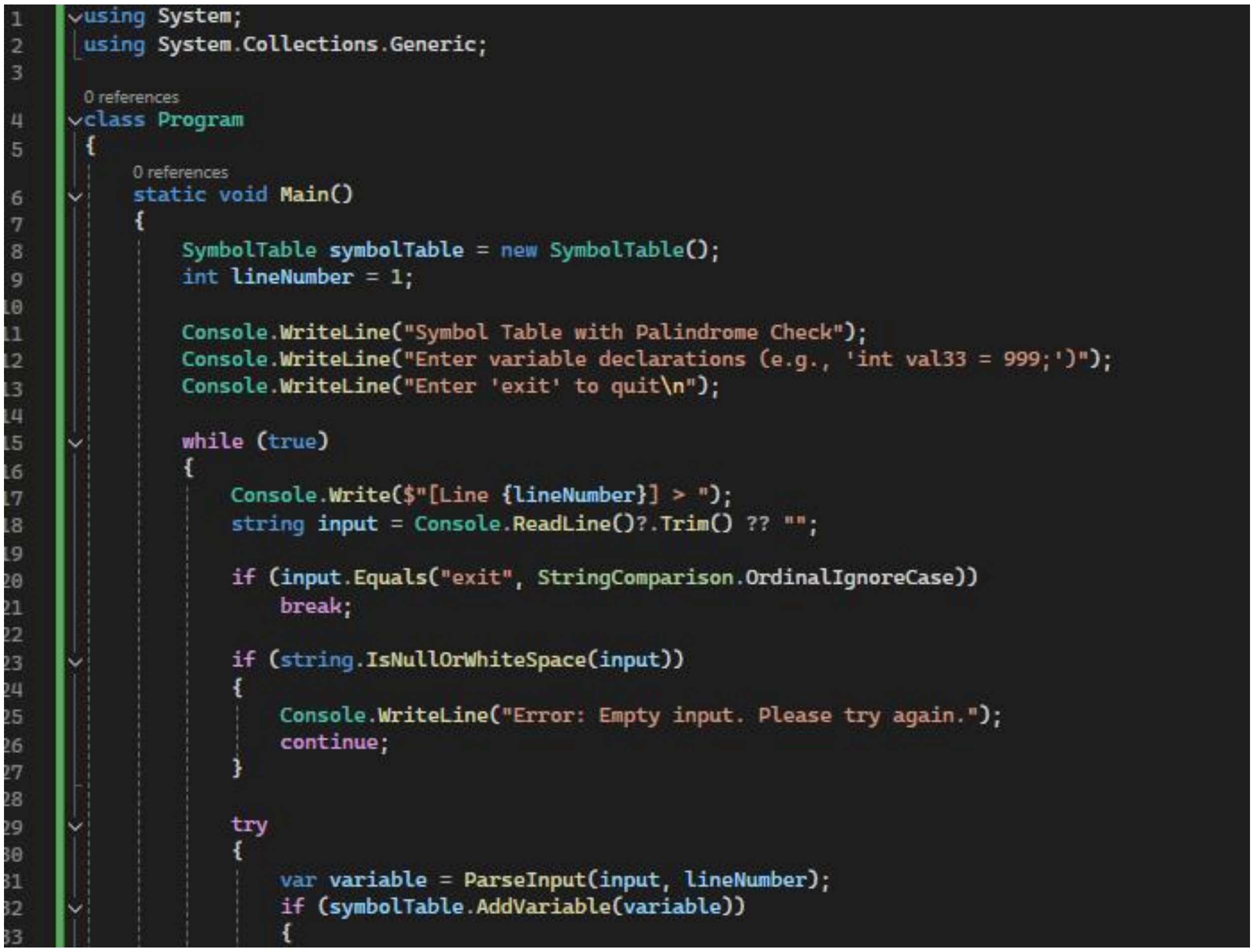
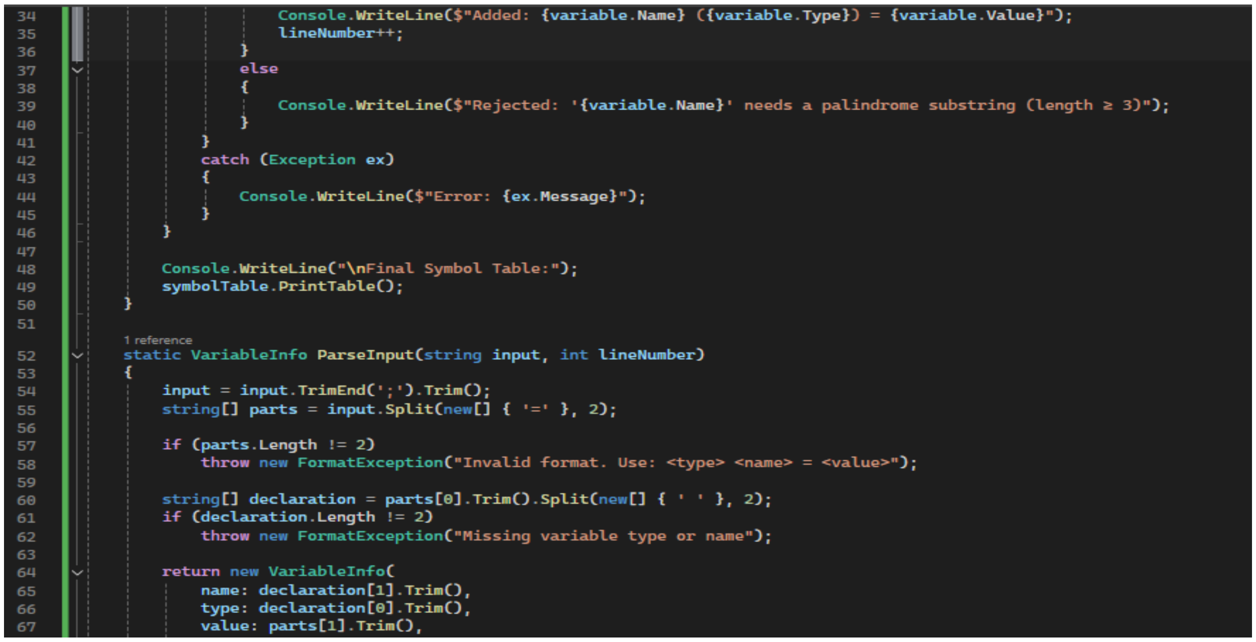
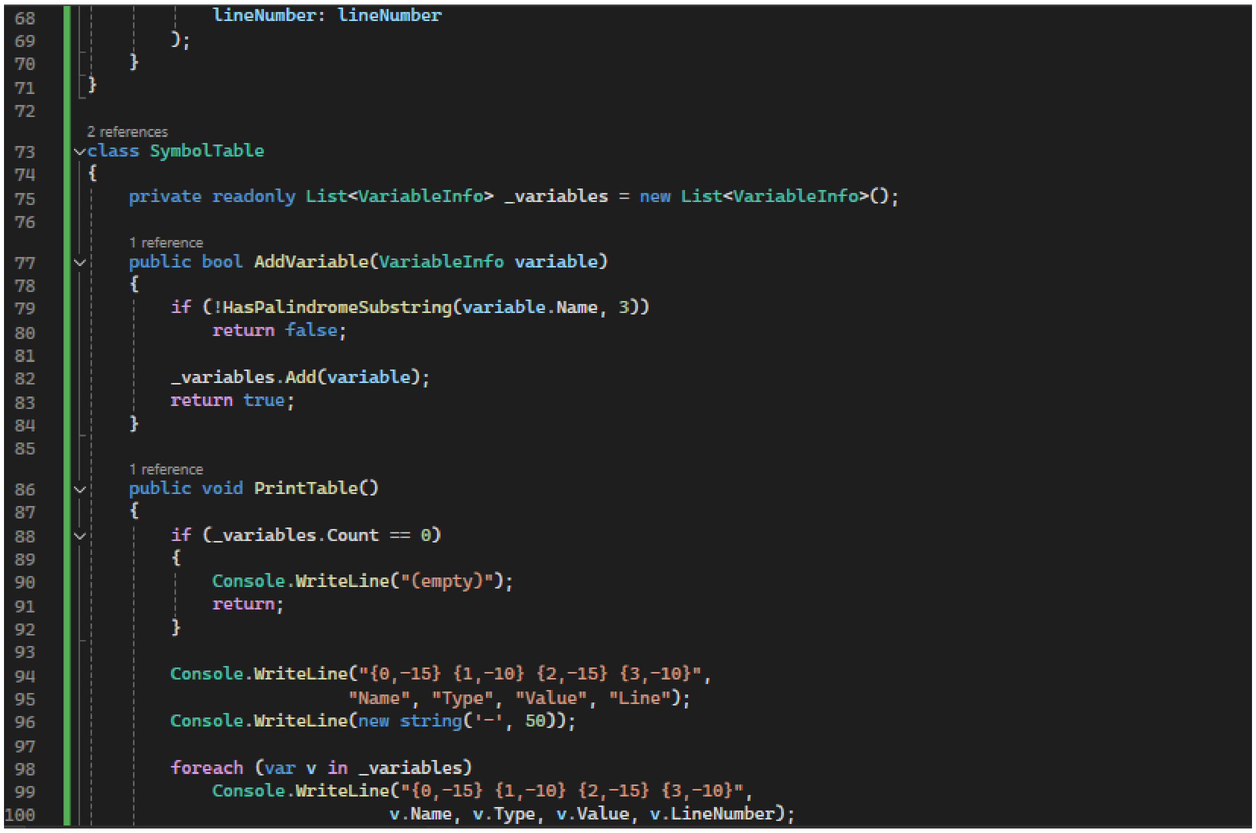
QUESTION 3



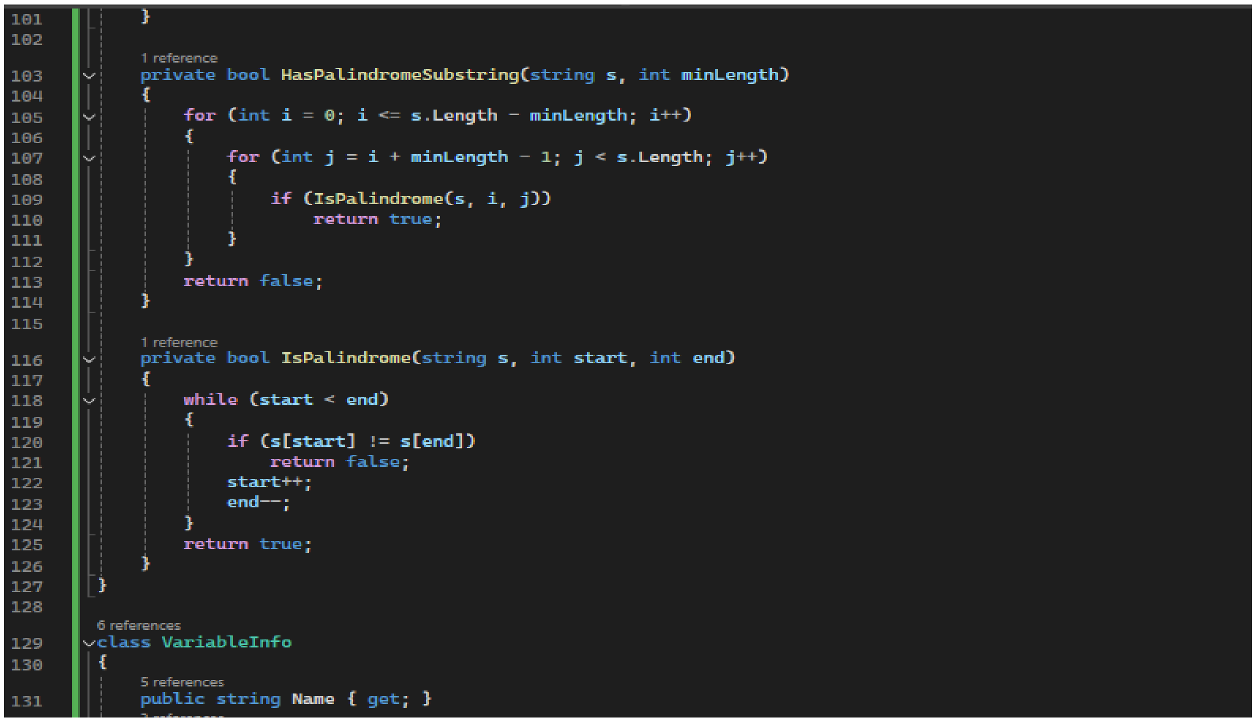
```
using System;
using System.Collections.Generic;

0 references
class Program
{
    0 references
    static void Main()
    {
        SymbolTable symbolTable = new SymbolTable();
        int lineNumber = 1;

        Console.WriteLine("Symbol Table with Palindrome Check");
        Console.WriteLine("Enter variable declarations (e.g., 'int val33 = 999;')");
        Console.WriteLine("Enter 'exit' to quit\n");

        while (true)
        {
            Console.Write($"[Line {lineNumber}] > ");
            string input = Console.ReadLine()?.Trim() ?? "";

            if (input.Equals("exit", StringComparison.OrdinalIgnoreCase))
                break;

            if (string.IsNullOrWhiteSpace(input))
            {
                Console.WriteLine("Error: Empty input. Please try again.");
                continue;
            }

            try
            {
                var variable = ParseInput(input, lineNumber);
                if (symbolTable.AddVariable(variable))
                {
```



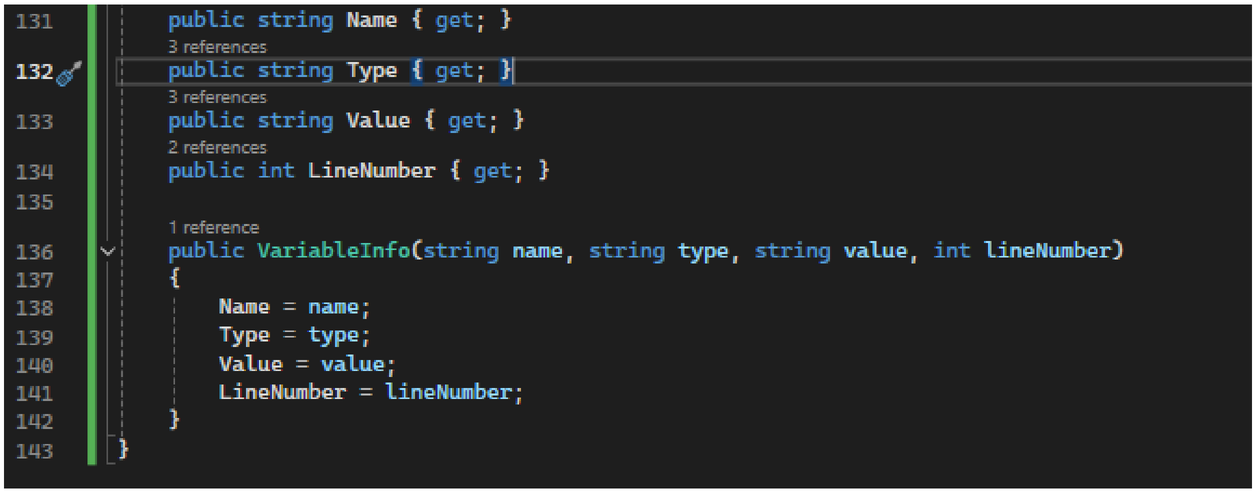
```
                    Console.WriteLine($"Added: {variable.Name} ({variable.Type}) = {variable.Value}");
                    lineNumber++;
                }
                else
                {
                    Console.WriteLine($"Rejected: '{variable.Name}' needs a palindrome substring (length ≥ 3)");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }

        Console.WriteLine("\nFinal Symbol Table:");
        symbolTable.PrintTable();
    }

    1 reference
    static VariableInfo ParseInput(string input, int lineNumber)
    {
        input = input.TrimEnd(';').Trim();
        string[] parts = input.Split(new[] { '=' }, 2);

        if (parts.Length != 2)
            throw new FormatException("Invalid format. Use: <type> <name> = <value>");

        string[] declaration = parts[0].Trim().Split(new[] { ' ' }, 2);
        if (declaration.Length != 2)
            throw new FormatException("Missing variable type or name");

        return new VariableInfo(
            name: declaration[1].Trim(),
            type: declaration[0].Trim(),
            value: parts[1].Trim(),
```



```
            lineNumber: lineNumber
        );
    }
}

class SymbolTable
{
    private readonly List<VariableInfo> _variables = new List<VariableInfo>();

    public bool AddVariable(VariableInfo variable)
    {
        if (!HasPalindromeSubstring(variable.Name, 3))
            return false;

        _variables.Add(variable);
        return true;
    }

    public void PrintTable()
    {
        if (_variables.Count == 0)
        {
            Console.WriteLine("(empty)");
            return;
        }

        Console.WriteLine("{0,-15} {1,-10} {2,-15} {3,-10}",
                          "Name", "Type", "Value", "Line");
        Console.WriteLine(new string('-', 50));

        foreach (var v in _variables)
            Console.WriteLine("{0,-15} {1,-10} {2,-15} {3,-10}",
                              v.Name, v.Type, v.Value, v.LineNumber);
```

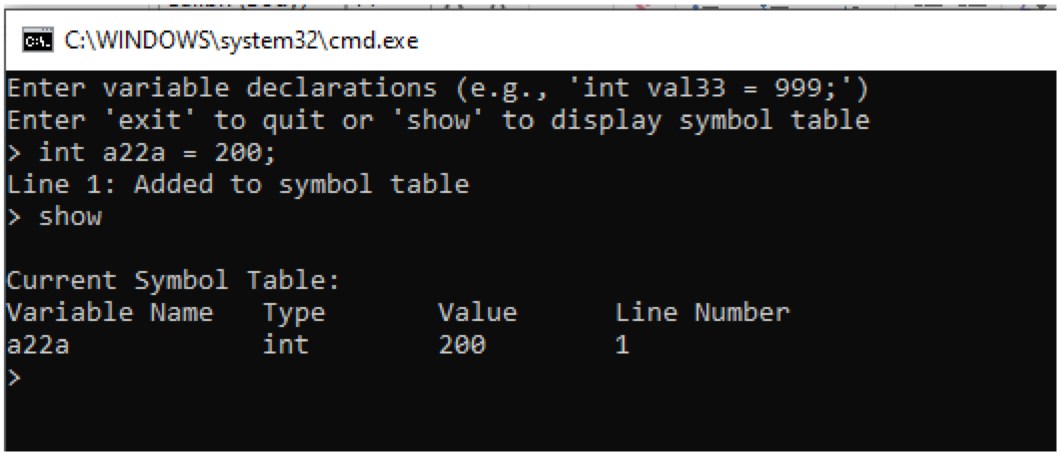


```
    }

    private bool HasPalindromeSubstring(string s, int minLength)
    {
        for (int i = 0; i <= s.Length - minLength; i++)
        {
            for (int j = i + minLength - 1; j < s.Length; j++)
            {
                if (IsPalindrome(s, i, j))
                    return true;
            }
        }
        return false;
    }

    private bool IsPalindrome(string s, int start, int end)
    {
        while (start < end)
        {
            if (s[start] != s[end])
                return false;
            start++;
            end--;
        }
        return true;
    }
}

class VariableInfo
{
    public string Name { get; }
```



```
    public string Name { get; }
    3 references
    public string Type { get; }
    3 references
    public string Value { get; }
    2 references
    public int LineNumber { get; }

    1 reference
    public VariableInfo(string name, string type, string value, int lineNumber)
    {
        Name = name;
        Type = type;
        Value = value;
        LineNumber = lineNumber;
    }
}
```

OUTPUT



C:\WINDOWS\system32\cmd.exe

```
Enter variable declarations (e.g., 'int val33 = 999;')
Enter 'exit' to quit or 'show' to display symbol table
> int a22a = 200;
Line 1: Added to symbol table
> show

Current Symbol Table:
Variable Name    Type       Value      Line Number
a22a             int        200        1
>
```