```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;

namespace KyleBushCompiler
{
    /// <summary>
    /// Contains all the symbols for a given application.
    /// </summary>
    public class SymbolTable
    {
        private const int TABLEWIDTH = 105;
        private const char DIVIDER_CHAR = '-';
        private List<Symbol> SymbolTableData { get; set; }

        /// <summary>
        /// Creates a new, empty Symbol Table.
        /// </summary>
        public SymbolTable()
        {
            SymbolTableData = new List<Symbol>();
        }

        /// <summary>
        /// Adds symbol with given kind and value to the symbol table, automatically setting the correct data_type,
        /// and returns the index where the symbol was located. If the symbol is already in the table,
        /// no change or verification is made, and this just returns the index where the symbol was found.
        /// </summary>
        /// <param name="symbol">The symbol to add to the symbol table</param>
        /// <param name="kind">The kind of symbol</param>
        /// <param name="value">The value associated with the given symbol</param>
        /// <returns>The index of the added symbol in the symbol table as an integer</returns>
        public int AddSymbol(string symbol, SymbolKind kind, int value)
        {
            SymbolTableData.Add(new Symbol(symbol, kind, DataType.Integer, value));
            return SymbolTableData.Count - 1;
        }

        /// <summary>
        /// Adds symbol with given kind and value to the symbol table, automatically setting the correct data_type,
        /// and returns the index where the symbol was located. If the symbol is already in the table,
        /// no change or verification is made, and this just returns the index where the symbol was found.
        /// </summary>
        /// <param name="symbol">The symbol to add to the symbol table</param>
        /// <param name="kind">The kind of symbol</param>
        /// <param name="value">The value associated with the given symbol</param>
        /// <returns>The index of the added symbol in the symbol table as an integer</returns>
        public int AddSymbol(string symbol, SymbolKind kind, double value)
        {
            SymbolTableData.Add(new Symbol(symbol, kind, DataType.Double, value));
            return SymbolTableData.Count - 1;
        }

        /// <summary>
        /// Adds symbol with given kind and value to the symbol table, automatically setting the correct data_type,
        /// and returns the index where the symbol was located. If the symbol is already in the table,
        /// no change or verification is made, and this just returns the index where the symbol was found.
        /// </summary>
        /// <param name="symbol">The symbol to add to the symbol table</param>
        /// <param name="kind">The kind of symbol</param>
        /// <param name="value">The value associated with the given symbol</param>
        /// <returns>The index of the added symbol in the symbol table as an integer</returns>
        public int AddSymbol(string symbol, SymbolKind kind, string value)
        {
            SymbolTableData.Add(new Symbol(symbol, kind, DataType.String, value));
            return SymbolTableData.Count - 1;
        }

        /// <summary>
        /// Returns the index where symbol is found, or -1 if not in the table
        /// </summary>
        /// <param name="symbol">The symbol to look for in the table.</param>
        /// <returns>The index of the symbol or -1 if not found</returns>
        public int LookupSymbol(string symbol)
        {
            return SymbolTableData.FindIndex(s => s.Name == symbol);
        }

        /// <summary>
        /// Return kind, data type, and value fields stored at index
        /// </summary>
        /// <param name="index">The index of the symbol to return</param>
        /// <returns></returns>
        public Symbol GetSymbol(int index)
        {
            return SymbolTableData[index];
        }

        /// <summary>
        /// Set appropriate fields at slot indicated by index
        /// </summary>
        /// <param name="index">The index of the symbol to update</param>
        /// <param name="kind">The kind of symbol</param>
        /// <param name="value">The value of the symbol</param>
        public void UpdateSymbol(int index, SymbolKind kind, int value)
        {
            SymbolTableData[index].Kind = kind;
            SymbolTableData[index].SetValue(value);
        }

        /// <summary>
        /// Set appropriate fields at slot indicated by index
        /// </summary>
        /// <param name="index">The index of the symbol to update</param>
        /// <param name="kind">The kind of symbol</param>
        /// <param name="value">The value of the symbol</param>
        public void UpdateSymbol(int index, SymbolKind kind, double value)
        {
            SymbolTableData[index].Kind = kind;
            SymbolTableData[index].SetValue(value);
        }

        /// <summary>
        /// Set appropriate fields at slot indicated by index
        /// </summary>
        /// <param name="index">The index of the symbol to update</param>
        /// <param name="kind">The kind of symbol</param>
        /// <param name="value">The value of the symbol</param>
        public void UpdateSymbol(int index, SymbolKind kind, string value)
        {
            SymbolTableData[index].Kind = kind;
            SymbolTableData[index].SetValue(value);
        }


        /// <summary>
        /// Prints the utilized rows of the symbol table in neat tabular format,
        /// showing only the value field which is active for that row
        /// </summary>
        public void PrintSymbolTable()
        {
            Console.WriteLine("SYMBOL TABLE");
            DrawHorizontalBorder(TABLEWIDTH, DIVIDER_CHAR);
            Console.WriteLine($"|{ "Name",-40 }|{ "Kind",10 }|{ "DataType",10 }|
              { "Value",40 }|");
            DrawHorizontalBorder(TABLEWIDTH, DIVIDER_CHAR);
            foreach (var symbol in SymbolTableData)
            {
                Console.WriteLine($"|{ symbol.Name,-40 }|{ symbol.Kind,10 }|
                  { symbol.DataType,10 }|{ symbol.GetValue(),40 }|");
            }
            DrawHorizontalBorder(TABLEWIDTH, DIVIDER_CHAR);
        }

        /// <summary>
        /// Draws a horizontal border using the given character repeated by the
          given length
        /// </summary>
        /// <param name="length">number of times to repeat character</param>
        /// <param name="character">character used to draw the border</param>
        public void DrawHorizontalBorder(int length, char character)
        {
            for (int i = 0; i < length; i++)
            {
                Console.Write(character);
            }
            Console.WriteLine();
        }
    }
}
```