

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Security.Cryptography.X509Certificates;
5 using System.Text;
6
7 namespace KyleBushCompiler
8 {
9     /// <summary>
10    /// Contains all the symbols for a given application.
11    /// </summary>
12    public class SymbolTable
13    {
14        private const int TABLEWIDTH = 105;
15        private const char DIVIDER_CHAR = '-';
16        private List<Symbol> SymbolTableData { get; set; }
17
18        /// <summary>
19        /// Creates a new, empty Symbol Table.
20        /// </summary>
21        public SymbolTable()
22        {
23            SymbolTableData = new List<Symbol>();
24        }
25
26        /// <summary>
27        /// Adds symbol with given kind and value to the symbol table,
28        /// automatically setting the correct data_type,
29        /// and returns the index where the symbol was located. If the symbol is
30        /// already in the table,
31        /// no change or verification is made, and this just returns the index
32        /// where the symbol was found.
33        /// </summary>
34        /// <param name="symbol">The symbol to add to the symbol table</param>
35        /// <param name="kind">The kind of symbol</param>
36        /// <param name="value">The value associated with the given symbol</
37        param>
38        /// <returns>The index of the added symbol in the symbol table as an
39        integer</returns>
40        public int AddSymbol(string symbol, SymbolKind kind, int value)
41        {
42            SymbolTableData.Add(new Symbol(symbol, kind, DataType.Integer,
43            value));
44            return SymbolTableData.Count - 1;
45        }
46
47        /// <summary>
48        /// Adds symbol with given kind and value to the symbol table,
49        /// automatically setting the correct data_type,
50        /// and returns the index where the symbol was located. If the symbol is
51        /// already in the table,
52        /// no change or verification is made, and this just returns the index
53        /// where the symbol was found.
54        /// </summary>
```

```
46     /// <param name="symbol">The symbol to add to the symbol table</param>
47     /// <param name="kind">The kind of symbol</param>
48     /// <param name="value">The value associated with the given symbol</param>
49     /// <returns>The index of the added symbol in the symbol table as an integer</returns>
50     public int AddSymbol(string symbol, SymbolKind kind, double value)
51     {
52         SymbolTableData.Add(new Symbol(symbol, kind, DataType.Double, value));
53         return SymbolTableData.Count - 1;
54     }
55
56     /// <summary>
57     /// Adds symbol with given kind and value to the symbol table, automatically setting the correct data_type,
58     /// and returns the index where the symbol was located. If the symbol is already in the table,
59     /// no change or verification is made, and this just returns the index where the symbol was found.
60     /// </summary>
61     /// <param name="symbol">The symbol to add to the symbol table</param>
62     /// <param name="kind">The kind of symbol</param>
63     /// <param name="value">The value associated with the given symbol</param>
64     /// <returns>The index of the added symbol in the symbol table as an integer</returns>
65     public int AddSymbol(string symbol, SymbolKind kind, string value)
66     {
67         SymbolTableData.Add(new Symbol(symbol, kind, DataType.String, value));
68         return SymbolTableData.Count - 1;
69     }
70
71     /// <summary>
72     /// Returns the index where symbol is found, or -1 if not in the table
73     /// </summary>
74     /// <param name="symbol">The symbol to look for in the table.</param>
75     /// <returns>The index of the symbol or -1 if not found</returns>
76     public int LookupSymbol(string symbol)
77     {
78         return SymbolTableData.FindIndex(s => s.Name == symbol);
79     }
80
81     /// <summary>
82     /// Return kind, data type, and value fields stored at index
83     /// </summary>
84     /// <param name="index">The index of the symbol to return</param>
85     /// <returns></returns>
86     public Symbol GetSymbol(int index)
87     {
88         return SymbolTableData[index];
89     }
90
```

```

91      /// <summary>
92      /// Set appropriate fields at slot indicated by index
93      /// </summary>
94      /// <param name="index">The index of the symbol to update</param>
95      /// <param name="kind">The kind of symbol</param>
96      /// <param name="value">The value of the symbol</param>
97      public void UpdateSymbol(int index, SymbolKind kind, int value)
98      {
99          SymbolTableData[index].Kind = kind;
100          SymbolTableData[index].SetValue(value);
101      }
102
103      /// <summary>
104      /// Set appropriate fields at slot indicated by index
105      /// </summary>
106      /// <param name="index">The index of the symbol to update</param>
107      /// <param name="kind">The kind of symbol</param>
108      /// <param name="value">The value of the symbol</param>
109      public void UpdateSymbol(int index, SymbolKind kind, double value)
110      {
111          SymbolTableData[index].Kind = kind;
112          SymbolTableData[index].SetValue(value);
113      }
114
115      /// <summary>
116      /// Set appropriate fields at slot indicated by index
117      /// </summary>
118      /// <param name="index">The index of the symbol to update</param>
119      /// <param name="kind">The kind of symbol</param>
120      /// <param name="value">The value of the symbol</param>
121      public void UpdateSymbol(int index, SymbolKind kind, string value)
122      {
123          SymbolTableData[index].Kind = kind;
124          SymbolTableData[index].SetValue(value);
125      }
126
127
128      /// <summary>
129      /// Prints the utilized rows of the symbol table in neat tabular format,
130      /// showing only the value field which is active for that row
131      /// </summary>
132      public void PrintSymbolTable()
133      {
134          Console.WriteLine("SYMBOL TABLE");
135          DrawHorizontalBorder(TABLEWIDTH, DIVIDER_CHAR);
136          Console.WriteLine($"{ "Name",-40 }|{ "Kind",10 }|{ "DataType",10 }|  ↗
137              { "Value",40 }|");
138          DrawHorizontalBorder(TABLEWIDTH, DIVIDER_CHAR);
139          foreach (var symbol in SymbolTableData)
140          {
141              Console.WriteLine($"{ symbol.Name,-40 }|{ symbol.Kind,10 }|  ↗
142                  { symbol.DataType,10 }|{ symbol.GetValue(),40 }|");

```

```
143     }
144
145     /// <summary>
146     /// Draws a horizontal border using the given character repeated by the given length
147     /// </summary>
148     /// <param name="length">number of times to repeat character</param>
149     /// <param name="character">character used to draw the border</param>
150     public void DrawHorizontalBorder(int length, char character)
151     {
152         for (int i = 0; i < length; i++)
153         {
154             Console.Write(character);
155         }
156         Console.WriteLine();
157     }
158 }
159 }
160
```