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

namespace KyleBushCompiler
{
    /// <summary>
    /// Contains all the reserve words for a language.
    /// </summary>
    public class ReserveTable
    {
        private const int TABLEWIDTH = 16;
        private const char DIVIDER_CHAR = '-';
        public List<ReservedWord> ReserveTableData { get; set; }

        /// <summary>
        /// Creates a new ReserveTable and initializes a list of ReservedWords.
        /// </summary>
        public ReserveTable()
        {
            ReserveTableData = new List<ReservedWord>();
        }

        /// <summary>
        /// Initializes the table with all the reserve words for the language.
        /// </summary>
        public void Initialize(List<ReservedWord> reservedWords)
        {
            ReserveTableData = reservedWords;
        }

        /// <summary>
        /// Returns the index of the row where the data was place, just adds to
          end of list.
        /// </summary>
        /// <param name="name">String name of reserved word</param>
        /// <param name="code">Integer code of reserved word</param>
        /// <returns>index of the row where the data was placed</returns>
        public int Add(string name, int code)
        {
            ReservedWord reservedWord = new ReservedWord(name, code);
            ReserveTableData.Add(reservedWord);
            return ReserveTableData.Count - 1;
        }

        /// <summary>
        /// Returns the code associated with name if name is in the table, else
          returns -1
        /// </summary>
        /// <param name="name">String name of reserved word</param>
        /// <returns></returns>
        public int LookupName(string name)
        {
            ReservedWord reservedWord = ReserveTableData.FirstOrDefault(x => x.Name == name);
            if (reservedWord == null)
            {
                return -1;
            }
            return reservedWord.Code;
        }

        /// <summary>
        /// Returns the associated name if code is there, else an empty string
        /// </summary>
        /// <param name="code">Intger code of reserved word</param>
        /// <returns></returns>
        public string LookupCode(int code)
        {
            ReservedWord reservedWord = ReserveTableData.FirstOrDefault(x => x.Code == code);
            if (reservedWord == null)
            {
                return "";
            }
            return reservedWord.Name;
        }

        /// <summary>
        /// Searches the table for the given code to test if it is valid.
        /// </summary>
        /// <param name="code">Integer code of reserved word</param>
        /// <returns>True if the code is valid, False if not.</returns>
        public bool isValidOpCode(int code)
        {
            ReservedWord reservedWord = ReserveTableData.FirstOrDefault(x => x.Code == code);
            if (reservedWord == null)
            {
                Console.WriteLine($"{code} is not a valid Op Code.");
                return false;
            }
            return true;
        }

        /// <summary>
        /// Prints the currently used contents of the Reserve table in neat tabular format
        /// </summary>
        public void PrintReserveTable()
        {
            Console.WriteLine("RESERVE TABLE");
            DrawHorizontalBorder(TABLEWIDTH, DIVIDER_CHAR);
            Console.WriteLine($"|{ "Name", -7 }|{ "Code", 5 }|");
            DrawHorizontalBorder(TABLEWIDTH, DIVIDER_CHAR);
            foreach (var code in ReserveTableData)
            {
                Console.WriteLine($"|{ code.Name, -7 }|{ code.Code, 5 }|");
            }
            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();
        }

    }
}

```