

```
1  /// PROGRAMMER : Jonas Smith
2  /// Purpose    : Get inputs from a user and solve a given linear equation.
3  /// Resources  : https://numerics.mathdotnet.com/LinearEquations.html - resource ➤
   to make this more general case with Math.net
4  ///          :
5  ///
6
7
8  using System;
9  using Project_1.classes;
10 using System.Collections.Generic;
11 using MathNet.Numerics.LinearAlgebra;
12
13 namespace Project_01
14 {
15     class ProjectOne
16     {
17         static void Main()
18         {
19             bool user_is_iterating = true;
20             List<char> variable_names = new List<char>() { 'a', 'b', 'c', 'd', ➤
               'r', 's' };
21             List<char> solution_names = new List<char>() { 'x', 'y' };
22             int row_n = 2;
23             int col_n = 2;
24
25             TestCases test_cases = new TestCases();
26
27             // flip this bit to begin testing
28             bool testing;
29             int test_index = 0;
30
31             testing = PromptForTesting();
32             Console.Clear();
33
34             PrintHeader();
35             PrintPurpose();
36             PrintDivider();
37
38             if (testing)
39             {
40                 Console.WriteLine("Press any key while testing");
41             }
42
43             while (user_is_iterating)
44             {
45                 List<variable> user_inputs = new List<variable>();
46
47                 if (testing)
48                 {
49                     if (test_index >= test_cases.cases.Count)
50                     {
```

```
51         goto stopTesting;
52     }
53
54     user_inputs = test_cases.cases[test_index];
55     test_index++;
56 }
57 else
58 {
59     user_inputs = GetUserInput(variable_names);
60 }
61
62 Vector<double> solution = CalculateSolutions(user_inputs, row_n, 7
    col_n);
63
64 PrintSolutions(solution, solution_names, user_inputs, testing);
65 UserIterations(ref user_is_iterating, testing);
66 }
67
68 stopTesting;;
69 if (testing)
70 {
71     Console.WriteLine("    Testing is finished");
72     Console.ReadLine();
73 }
74 }
75
76 static bool PromptForTesting()
77 {
78     bool correct_input = false;
79
80     while (!correct_input)
81     {
82         Console.Clear();
83         Console.WriteLine("Would you like to use the test data?");
84         Console.Write("    [y]es or [n]o : ");
85
86         string user_input = Console.ReadLine().ToLower();
87
88         if (user_input == "y")
89         {
90             return true;
91         }
92         else if (user_input == "n")
93         {
94             return false;
95         }
96     }
97
98     return false;
99 }
100
101
```

```
102     static void PrintHeader()
103     {
104         List<string> headers = new List<string>() { "Math 371", "Spring 2020", "Lienar System Solver", "Jonas Smith" };
105         int size = 32;
106         string buffer = "";
107
108         for (int i = 0; i < headers.Count; i++)
109         {
110
111             buffer = GetBuffer(headers[i], size);
112             Console.WriteLine("{0}{1}", buffer, headers[i]);
113         }
114     }
115
116     static string GetBuffer(string value, int length)
117     {
118         int buffer_length = (length - Convert.ToInt32(value.Length)) / 2;
119         string buffer = "";
120
121         for (int i = 0; i < buffer_length; i++)
122             buffer += " ";
123
124         return buffer;
125     }
126
127     static void PrintPurpose()
128     {
129         Console.WriteLine();
130         Console.WriteLine("    Take inputs from the user ");
131         Console.WriteLine("{a,b,c,d,r,s} and calculate x");
132         Console.WriteLine("and y from the system of linear");
133         Console.WriteLine("equations");
134         Console.WriteLine();
135         Console.WriteLine("similar to: ax + by = r");
136         Console.WriteLine("          cx + dy = s");
137     }
138
139
140
141     static void PrintDivider() => Console.WriteLine
142         ("_____\\n");
143
144     /// <summary>
145     ///     Prompts the user to enter a variable based on the list given
146     /// </summary>
147     static List<variable> GetUserInput(List<char> variables)
148     {
149         List<variable> user_inputs = new List<variable>();
150
151         for (int i = 0; i < variables.Count; i++)
152         {
```

```
152         user_inputs.Add(GetVariableInput(variables[i], i));
153     }
154
155     return user_inputs;
156 }
157
158 /// <summary>
159 /// Only allows numbers . and - to be entered into the fields
160 /// </summary>
161 /// <param name="var_name"></param>
162 /// <param name="index"></param>
163 /// <returns></returns>
164 static variable GetVariableInput(char var_name, int index)
165 {
166     bool user_input_wrong = true;
167
168     string message = "    Enter the value for";
169
170     string buffer = "";
171
172     for (int i = 0; i < message.Length; i++)
173     {
174         buffer += " ";
175     }
176
177     double input = 0.0;
178
179     while (user_input_wrong)
180     {
181         string prompt = string.Format("{0} {1} = ", buffer, var_name);
182
183         if (index == 0)
184         {
185             prompt = string.Format("{0} {1} = ", message, var_name);
186         }
187
188         Console.Write(prompt);
189
190         try
191         {
192             string _val = "";
193
194             ConsoleKeyInfo key;
195
196             do
197             {
198                 key = Console.ReadKey(true);
199                 if (key.Key != ConsoleKey.Backspace)
200                 {
201                     double val = 0;
202                     bool _x = double.TryParse(key.KeyChar.ToString(), out
```

```

203         if (_x)
204         {
205             _val += key.KeyChar;
206             Console.Write(key.KeyChar);
207         }
208
209         if (key.Key == ConsoleKey.OemPeriod)
210         {
211             _val += ".";
212             Console.Write(key.KeyChar);
213         }
214
215         if (key.Key == ConsoleKey.OemMinus)
216         {
217             _val += "-";
218             Console.Write(key.KeyChar);
219         }
220     }
221     else
222     {
223         if (key.Key == ConsoleKey.Backspace && _val.Length > 0)
224         {
225             _val = _val.Substring(0, (_val.Length - 1));
226             Console.Write("\b \b");
227         }
228     }
229     } while (key.Key != ConsoleKey.Enter);
230
231
232     input = Convert.ToDouble(_val);
233
234     user_input_wrong = false;
235 }
236 catch
237 {
238     index++;
239 }
240
241 Console.WriteLine();
242 }
243
244
245     return new variable(var_name, input);
246 }
247
248 /// <summary>
249 ///     Moves the user_input variable list into a set of arrays used to
250 ///     calculate the values.
251 /// </summary>
252 /// <param name="user_input"></param>
253 /// <returns></returns>

```

```
253     static Vector<double> CalculateSolutions(List<variable> user_input, int ↗
        row_n, int col_n)
254     {
255         // Initialize the row and col number given when the application first ↗
        // runs.
256         // This can be changed easily enough with the lists to allow ↗
        // this to work
257         // with any number of linear equations and variables.
258         int row = row_n;
259         int col = col_n;
260
261         // Using some linear algebra we can use  $Ax = b$ 
262         // Build matrix A
263         double[,] matrix_A = new double[row, row];
264
265         int index = 0;
266
267         for (int i = 0; i < row; i++)
268         {
269             for (int k = 0; k < col; k++)
270             {
271                 matrix_A[i, k] = user_input[index].value;
272                 index++;
273             }
274         }
275
276         var A = Matrix<double>.Build.DenseOfArray(matrix_A);
277
278         // Build the coefficient vector b
279         double[] matrix_B = new double[row];
280
281         for (int i = 0; i < col; i++)
282         {
283             matrix_B[i] = user_input[index].value;
284
285             index++;
286         }
287
288         var b = Vector<double>.Build.Dense(matrix_B);
289
290         // Solve!
291         Vector<double> x = A.Solve(b);
292
293         return x;
294     }
295
296     static void PrintSolutions(Vector<double> solutions, List<char> ↗
        solution_names, List<variable> user_input, bool testing)
297     {
298         if (testing)
299         {
300             Console.WriteLine("\n          Solutions for          ");
```

```
301
302     string varialbe_string = "";
303
304     for (int i = 0; i < user_input.Count; i++)
305     {
306
307         if (i != user_input.Count - 1)
308         {
309
310             varialbe_string += string.Format("{0}={1}, ", user_input
311 [i].name, user_input[i].value);
312         }
313         else
314         {
315
316             varialbe_string += string.Format("and {0}={1}",
317 user_input[i].name, user_input[i].value);
318         }
319     }
320     Console.WriteLine("{0}", varialbe_string);
321 else
322 {
323     Console.WriteLine("\n          Solutions          ");
324     Console.WriteLine("_____");
325 }
326
327 for (int i = 0; i < solutions.Count; i++)
328 {
329     string output = "";
330
331     // there is an infinite number of intersections.
332     if (Double.IsNaN(solutions[i]))
333     {
334         output = string.Format("{0} = {1}", solution_names[i], "
335 Infinitely many");
336     }
337     // If the object is not a number therefore we know we have a zero
338     // in the denominator
339     else if (Double.IsInfinity(solutions[i]))
340     {
341         output = string.Format("{0} = {1}", solution_names[i], " No
342 solution");
343     }
344     // one solution.
345     else
346     {
347         // the solutions is negative so we move the margin over one
348         characters
349         if (solutions[i] < 0)
350             output = string.Format("{0} = {1}", solution_names[i],
```

```
solutions[i].ToString("N5"));
347     else
348         output = string.Format("{0} = {1}", solution_names[i],
            solutions[i].ToString("N5"));
349     }
350
351     string buffer = GetBuffer(output, 32);
352
353     Console.WriteLine("{0}{1}", buffer, output);
354 }
355
356 if (testing)
357     Console.WriteLine("_____");
358
359 Console.WriteLine();
360 }
361
362 static void UserIterations(ref bool user_iteration, bool testing)
363 {
364     if (!testing)
365     {
366         bool correct_input = true;
367
368         while (correct_input)
369         {
370
371             Console.WriteLine("    Would you like to continue?");
372             Console.Write("        [y]es or [n]o    : ");
373
374             string user_input = Console.ReadLine().ToLower();
375
376             if (user_input == "y")
377             {
378                 correct_input = false;
379                 Console.Clear();
380                 Iterate();
381             }
382             else if (user_input == "n")
383             {
384                 user_iteration = false;
385                 correct_input = false;
386             }
387             else
388             {
389                 Console.Write(new String(' ', Console.BufferWidth));
390             }
391         }
392     }
393     else
394     {
395         Console.ReadLine();
396     }
```



```
397     }
398
399     static void Iterate()
400     {
401         PrintHeader();
402         PrintPurpose();
403         PrintDivider();
404     }
405 }
406 }
407
```