

Nested Array Flatten

Summary:

The Nested Array Flatten program will take a text representation of a nested int array and output its flattened value. For a proof of concept, text inputs will be first parsed into a nested array type, which will then be flattened into a single-dimensional int array before finally being shown as a text output.

By: Hyuk Jin Kwon

Hyukjink@sfu.ca

Methodology and Implementation:

The Nested Array Flatten program is a simple WPF application and is written in C# and Visual Studio Community 2015. All libraries and tools required for the program are available through the .NET Framework.

The program will only take valid text representations of a nested in array. Elements are separated using a single whitespace character, and only brackets ('[' and ']'), digits (0-9), and the negative sign ('-') are considered other valid characters.

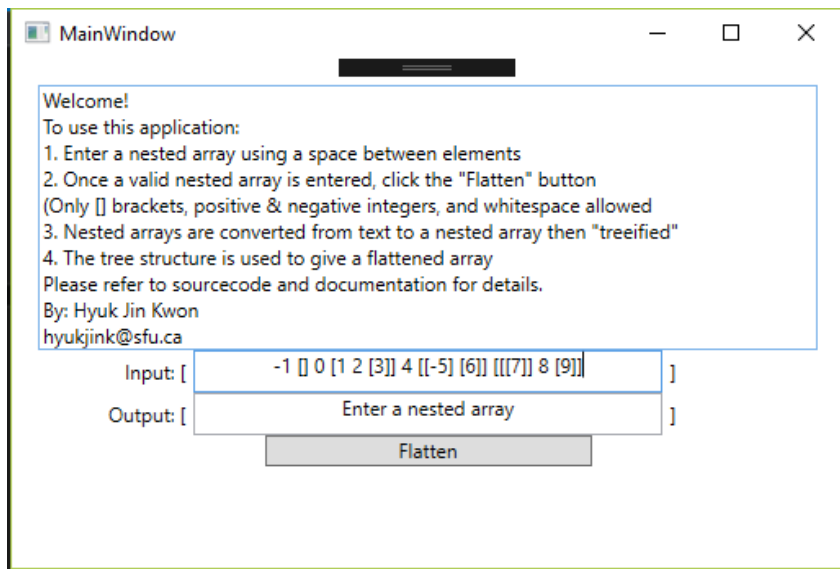


Fig. 1: A valid input of array elements

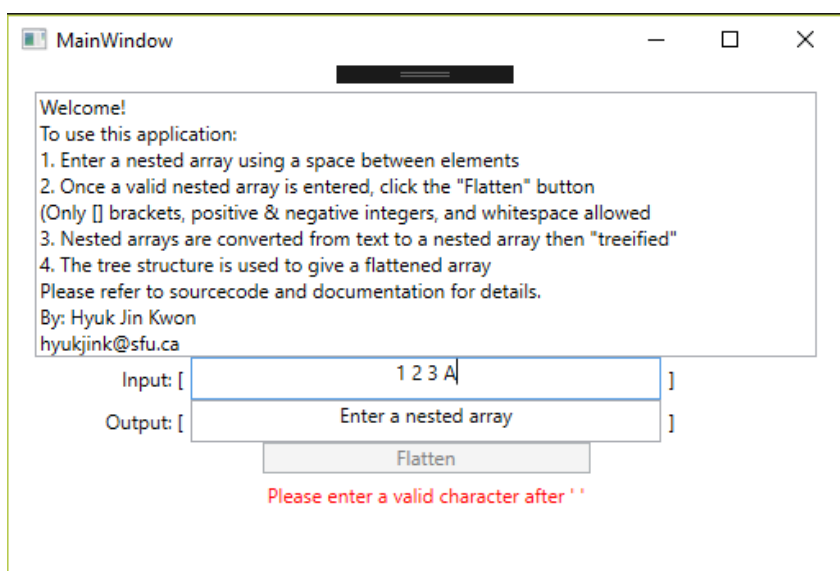
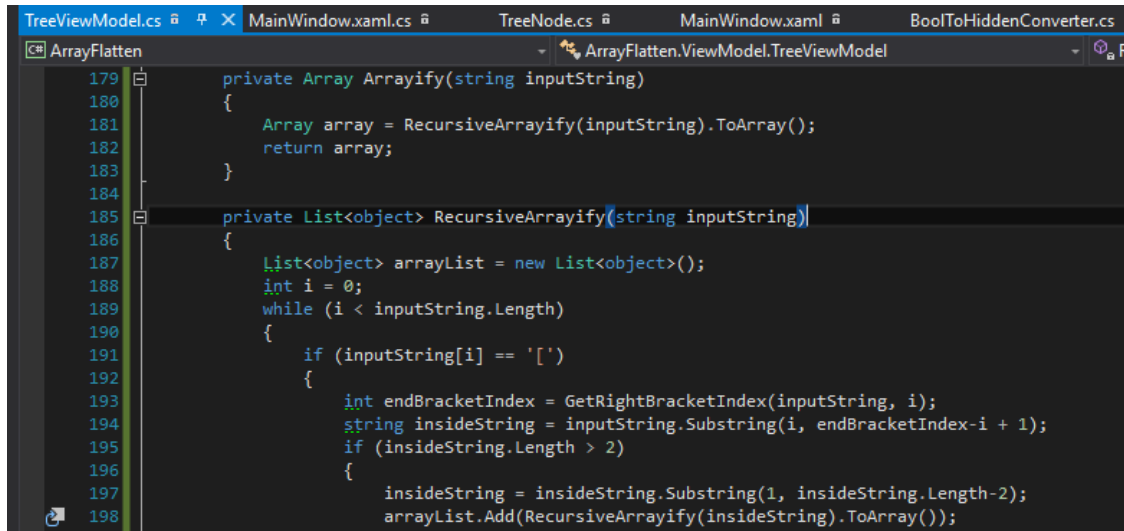


Fig. 2: An invalid input of array elements

Once a valid nested array input is accepted, the array is then converted into a nested Array object. For more details, see Line 179 of TreeViewModel.cs (Found under the “ViewModels” folder of the solution).

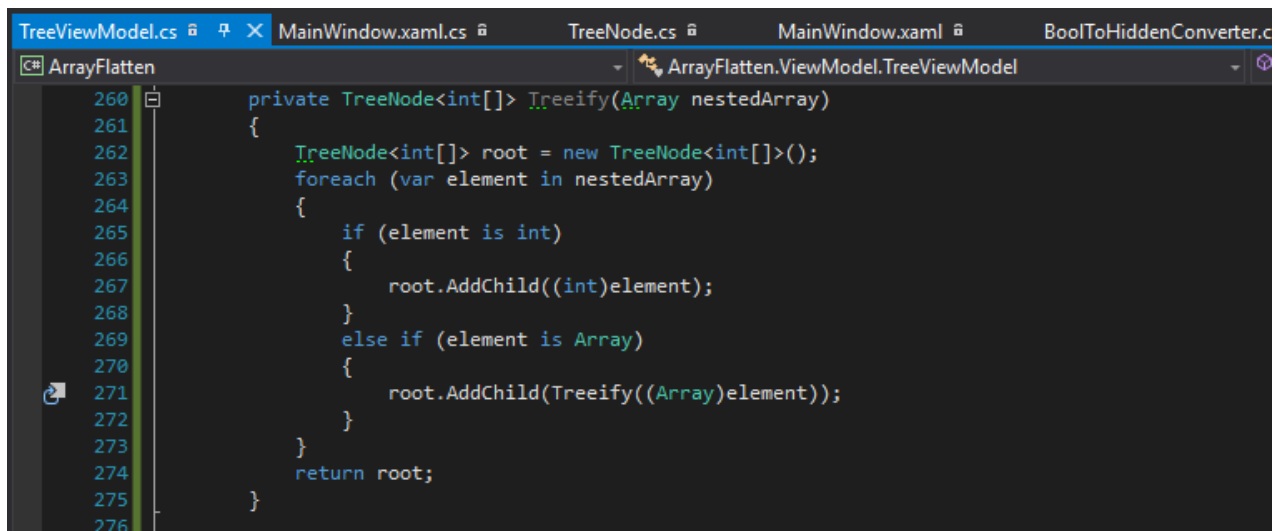
Note: This conversion to a nested array structure is purely for demonstration purposes, and could have been skipped if just a string representation of a nested array suffices.

A screenshot of a Visual Studio code editor showing the 'Arrayify' method in 'TreeViewModel.cs'. The method is private and takes a string 'inputString' as a parameter. It calls 'RecursiveArrayify' and returns the result as an 'Array'. The 'RecursiveArrayify' method is also private and takes a string 'inputString'. It creates a 'List<object>' named 'arrayList', initializes 'i' to 0, and enters a 'while' loop that continues as long as 'i' is less than 'inputString.Length'. Inside the loop, it checks if 'inputString[i]' is '['. If so, it finds the 'endBracketIndex' using 'GetRightBracketIndex', extracts 'insideString' from 'inputString' between 'i' and 'endBracketIndex - i + 1', and if 'insideString.Length' is greater than 2, it recursively calls 'RecursiveArrayify' on 'insideString' and adds the result to 'arrayList'. The loop increments 'i' and continues until the end of the string is reached.

```
179 private Array Arrayify(string inputString)
180 {
181     Array array = RecursiveArrayify(inputString).ToArray();
182     return array;
183 }
184
185 private List<object> RecursiveArrayify(string inputString)
186 {
187     List<object> arrayList = new List<object>();
188     int i = 0;
189     while (i < inputString.Length)
190     {
191         if (inputString[i] == '[')
192         {
193             int endBracketIndex = GetRightBracketIndex(inputString, i);
194             string insideString = inputString.Substring(i, endBracketIndex - i + 1);
195             if (insideString.Length > 2)
196             {
197                 insideString = insideString.Substring(1, insideString.Length - 2);
198                 arrayList.Add(RecursiveArrayify(insideString).ToArray());
199             }
200             i = endBracketIndex + 1;
201         }
202         else
203         {
204             arrayList.Add(inputString[i]);
205             i++;
206         }
207     }
208     return arrayList;
209 }
```

Fig. 3: Code snippet of input string being parsed into a nested array structure

Once a nested array structure is obtained, it is then ready for flattening. To do so, the nested array structure is then converted into a tree where leaf nodes contain integer values and inner nodes contain arrays. From there, a depth-first search traversal will order the elements and output them into a one-dimensional int array. For more details, see Line 260 of TreeViewModel.cs (Found under the “ViewModels” folder of the solution) and the TreeNode.cs file (Found under the “Models” folder of the solution).

A screenshot of a Visual Studio code editor showing the 'Treeify' method in 'TreeViewModel.cs'. The method is private and takes an 'Array nestedArray' as a parameter. It creates a 'TreeNode<int[]>' named 'root' and enters a 'foreach' loop over 'nestedArray'. For each 'element', it checks if 'element is int'. If so, it adds the element to 'root' using 'root.AddChild((int)element)'. If 'element is Array', it recursively calls 'Treeify' on the array and adds the result to 'root' using 'root.AddChild(Treeify((Array)element))'. After the loop, it returns 'root'.

```
260 private TreeNode<int[]> Treeify(Array nestedArray)
261 {
262     TreeNode<int[]> root = new TreeNode<int[]>();
263     foreach (var element in nestedArray)
264     {
265         if (element is int)
266         {
267             root.AddChild((int)element);
268         }
269         else if (element is Array)
270         {
271             root.AddChild(Treeify((Array)element));
272         }
273     }
274     return root;
275 }
276
```

Fig. 4: Code snippet of nested array being converted into a tree structure

Results

The efficiency of this program is based on the depth of the tree structure being created, and the depth-first search algorithm. Hence, the worst-case performance is $O(|V| + |E|)$ in time, where:

1. V is the sum of integer elements and inner arrays
2. E is the sum of all nestings of ints in arrays

Ex:

[1 [2] 3] nests "1" once, "2" twice, "3" once
Hence $E = 4$.