OBJECTIVES
- Write LabVIEW programs using arrays.

Part 1. Creating an Array of Controls or Indicators
Here are the steps you follow to create an array of indicators or controls on the front panel.

a) From the Controls > Modern > Array, Matrix & Cluster palette, place an array on the front panel. At this point you will have an empty array shell, whose terminal on the block diagram is colored black.

b) From the Controls palette, place a Boolean control or indicator, or a numeric control or indicator, or a string control or indicator inside the array shell. At this point the array will take on the data type of whatever item you placed inside the shell, and the terminal on the block diagram will change to the correct color (green for Boolean, blue for integer, orange for floating-point, or pink for string).

c) To display more than one of the elements in the array on the front panel, use the array’s resizing handles to pull down the lower edge of the array until the desired number of elements are visible.

For practice with arrays of controls and arrays of indicators, follow these steps:

1. Create a VI whose front panel has:
   - one array of knobs
   - one array of gauges
   - one array of push buttons
   - one array of LEDs
   - one array of string controls
   - one array of string indicators.
   Expand the arrays until you can see three elements in each array. In the array of knobs, set the first three knobs to different values between 0 and 10. In the array of push buttons, switch on the first and third buttons. In the array of string controls, type the names of your three favorite foods in the first three controls. On the block diagram, wire the array of knobs to the array of gauges, and wire the array of push buttons to the array of LEDs, and wire the array of string controls to the array of string indicators. Run the VI, and notice that each indicator takes on the value of the corresponding control.

2. Rewire this VI’s block diagram so that:
   - the value displayed on each gauge is equal to the square root of the value on the corresponding knob
   - each LED is on if the corresponding push button is switched off, and vice versa
   - the letters displayed in the string indicators are all uppercase even when the letters typed into the string controls are lowercase.
3. Modify the VI’s front panel by adding three toggle switches with the following labels:
   - Normal or Square Root?
   - Normal or Inverted?
   - Normal or Uppercase?
   Rewire the block diagram so that the switches control whether the numbers, Boolean values, and strings are displayed normally or in their modified forms.

4. Modify your program so that it continues to run until the user presses a STOP button on the front panel. Save this VI as Lab9ControlArray.vi and show me your working program.

---

**Part 2. Creating an Array of Constants**

You can also create an array of constants (either Boolean constants, or numeric constants, or string constants). Arrays of constants appear on the block diagram, not on the front panel. Here are the steps you follow to create an array of constants:

a) From the Functions > Programming > Array palette, place an array constant on the block diagram. At this point you will have an empty array shell, colored black. There will be no corresponding array on the front panel, because constants appear only on the block diagram.

b) Place either a Boolean constant (found on the Functions > Programming > Boolean palette), or a numeric constant (found on the Functions > Programming > Numeric palette), or a string constant (found on the Functions > Programming > String palette) inside the array shell. At this point the array will take on the data type of whatever item you placed inside the shell, and the array will change to the correct color.

c) To display more than one of the elements in the array, use the array’s resizing handles to pull down the lower edge of the array until the desired number of elements are visible.

d) Set the value of any of the elements in the array either by typing in the desired value (for numeric constants or string constants) or by clicking the element to change its value (for Boolean constants).

For practice with arrays of constants, follow these steps:

1. Create a new VI whose block diagram has:
   - one array of numeric constants
   - one array of Boolean constants
   - one array of string constants.

   On the front panel, place:
   - one array of gauges
   - one array of LEDs
   - one array of string indicators.

   Expand all of the arrays until you can see three elements in each array. In the array of numeric constants, set the first three elements to different values between 0 and 10. In the array of Boolean constants, set the first and third elements equal to TRUE. In the array of string constants, set the first three elements equal to the names of your three favorite actors. On the block diagram, wire the array of numeric constants to the array of gauges, and wire the array of Boolean constants to the array of LEDs, and wire the array of string constants to the array of string indicators. Run the VI, and notice that each indicator takes on the value of the corresponding constant.
2. Rewire this VI’s block diagram so that:
   - the value on each gauge is equal to the corresponding numeric constant divided by 2
   - each LED is lit if the corresponding Boolean constant is FALSE, and is dark if the constant is TRUE
   - each string indicator displays the reverse (spelled backwards) of the corresponding string constant.

3. Modify this VI so that all of the displayed values flip back and forth between their normal values and their modified values once every second, until the user presses a STOP button. Save this VI as Lab9ConstantArray.vi and show me your working program.

Part 3. Arrays and Digital I/O

In some earlier labs you’ve used the DAQ Assistant to input or output a single digital (Boolean) value. What you probably haven’t realized, though, is that by default when you use the DAQ Assistant to perform digital I/O tasks, it does not write out or read in a single Boolean value. Rather, the DAQ Assistant writes out or reads in an array of Boolean values.

To see that this is true, follow these steps:

1. Create a new VI whose front panel contains an array of push buttons. But for now you want your array to contain only one push button, so be careful not to turn on any of the push buttons other than the first one.

2. On the block diagram, place a DAQ Assistant and configure it to perform digital output on port0/line0, just as you have done in previous labs.

3. On the block diagram, wire your array of push buttons directly to the DAQ Assistant’s data input. Notice that LabVIEW accepts this connection, and that the thickness of the resulting wire indicates that we’re passing an entire array of Boolean values, not just a single Boolean value.

4. Place the entire block diagram inside a While loop with a STOP button.

5. Run a wire between the myDAQ’s DIO0 terminal and LED 0 on the ETS-7000 trainer. Also, as always, connect the myDAQ’s digital GND terminal to the trainer’s GND pin. Then turn on the trainer’s power.

6. Run the VI, and you should find that the first push button in your array controls the trainer’s LED.

7. Stop the VI. Now let’s extend this by adding more push buttons to the array. On your VI’s front panel, resize the array so that you can see eight push buttons. Initially most of these will be “grayed out.” To bring them out of this gray state, turn on the eighth push button, and you should see that now all eight of the push buttons are fully colored, instead of being grayed out.

8. On your VI’s block diagram, double-click the DAQ Assistant to reconfigure it. On the Configuration tab of the DAQ Assistant’s dialog box you should see the word “Details”
and a button with two arrowheads pointing to the right. Click this button, and you should see that currently you have one DigitalOut task that is using your myDAQ’s port0/line0.

9. Click the button labeled “Add Channels” (with a blue + sign) near the top of the dialog box. This will open another dialog box, which you have seen before, that lists all of the digital channels on your myDAQ.

10. We want to add seven more channels. So click **port0/line1**, then press the keyboard’s Shift key, and while you hold down the Shift key, click **port0/line7**. This should highlight all seven of the channels from **port0/line1** through **port0/line7**. Click the OK button to return to the original DAQ Assistant dialog box.

11. In this dialog box, you should now see eight channels listed. Click the OK button to accept this configuration.

12. Run seven more wires between the myDAQ’s DIO1 through DIO7 terminals and the seven corresponding LEDs on the trainer.

13. Run the VI, and you should find that each push button in your VI controls one LED on the trainer. Save this VI as **Lab9ArrayOutput.vi** and show me your working program.

The program you just wrote demonstrated output of an array of Boolean values. Now let’s see how to input an array of Boolean values.

1. Create a new VI whose front panel contains an array of LEDs. But for now you want your array to contain only one LED, so be careful not to turn on any of the LEDs other than the first one.

2. On the block diagram, place a DAQ Assistant and configure it to perform digital input on port0/line0, just as you have done in previous labs.

3. On the block diagram, wire your array of LEDs directly to the DAQ Assistant’s data output. Notice that LabVIEW accepts this connection, because the DAQ Assistant is producing an array of values, and so your can wire it directly to your array of LEDs.

4. Place the entire block diagram inside a While loop with a STOP button.

5. Run a wire between the myDAQ’s DIO0 terminal and Data Switch SW0 on the trainer. Make sure that the myDAQ’s digital GND terminal is connected to the trainer’s GND pin. Turn on the trainer’s power.

6. Run the VI, and you should find that the trainer’s data switch SW0 controls the first LED in your array.

7. Stop the VI, and then extend it by adding seven more LEDs to the VI’s array, and connecting seven more of the trainer’s data switches. (The steps you’ll need to follow are similar to steps you did above.)
8. Run the VI, and you should find that each data switch on the trainer controls one LED on your VI’s front panel.

9. Add two more single LEDs (not part of the array) to your front panel. Labels these LEDs All switches on and At least one switch on. Modify your block diagram so that these LEDs work as their names suggest. (Hint: Take a look at some of the functions that we haven’t used so far on the Functions > Programming > Boolean palette.)

10. Save this VI as Lab9ArrayInput.vi and show me your working program. ________

Part 4. Array Functions

As you saw in Parts 1 and 2 of this lab, you can use numeric functions (such as Multiply) on numeric arrays, and you can use Boolean functions (such as Not) on Boolean arrays, and you can use string functions (such as Reverse String) on string arrays. These are the same functions that you’ve used previously on single numeric values, single Boolean values, and single string values. So one nice feature of LabVIEW is that you can use these functions either on single values, or on arrays of values. Whenever you use any of these functions on an array, the operation is performed on each element in the array.

LabVIEW also provides many powerful array functions that work only with arrays. These are located on the Functions > Programming > Array palette, and can be used on arrays of any data type (numeric, Boolean, or string). For example:

- The Array Size function tells you how many elements an array contains.
- The Index Array function tells you the value of a particular element within the array.
- The Search 1D Array function tells you whether a particular value appears anywhere in an array, and tells you where the value appears in the array.
- The Sort 1D Array function reorders a numeric array’s elements from smallest to largest, and it reorders a string array’s elements in alphabetical order.

Practice with Array Functions

1. Create a new VI whose front panel has an array of string controls labeled Names and an array of string indicators labeled Sorted Names. Wire the block diagram so that when the program runs, the names that the user has entered into the string controls are sorted alphabetically and displayed in the string indicators. Test your program with about six or eight names.

2. Add a numeric indicator labeled Number of names entered to your VI’s front panel. Modify your block diagram so that when the program runs, it behaves as it did above and also displays the number of names that the user has entered into the string controls.

3. Add a string control labeled Name to search for and an LED labeled Search name found? to your VI’s front panel. Modify your block diagram so that when the program runs, it behaves as it did above and also lights up the LED if the name typed into the string control matches one of the names in the array. (To figure out how to do this, read the Detailed Help for the Search 1D Array function.)

4. Place your entire block diagram inside a While Loop so that the program will keep running until the user presses a STOP button on the front panel. Save this VI as Lab9ArrayFunctions.vi and show me your working program.
The next program applies some of the same functions (and a couple of new ones) to an array of numbers instead of an array of strings.

1. Create a new VI whose front panel contains an array of numeric controls labeled Input Numbers and two numeric indicators labeled Sum and Product. Expand the array to show eight or ten of the controls. Wire the block diagram to display the sum and product of the values that the user enters in the controls. Your program should work for any number of values that the user decides to enter. (Hint: Take a look at the functions in the Functions > Numeric palette. Two of these functions will make your job very easy.)

2. Modify this VI by placing a numeric indicator labeled Average on the front panel. Wire the block diagram so that in addition to displaying the sum and the product of the numbers, it also displays the average.

3. Add a Waveform Graph to the front panel, and wire this graph to your array so that the user’s numbers are plotted on a graph.

4. Add an array of numeric indicators labeled Sorted Input Numbers to the front panel. Modify your block diagram so that this new array displays the user’s numbers in numeric order from smallest to largest.

5. Add a horizontal toggle switch labeled Sorted in increasing order or decreasing order? Rewire the block diagram so that the sorted numbers are displayed in the order indicated by this toggle switch. (Hint: On the Array palette, one of the functions that you haven’t used yet will make this easy.)

6. Place your entire block diagram inside a While Loop so that the program will keep running until the user presses a STOP button on the front panel. Save this VI as Lab9ArrayAverage.vi and show me your working program.

Part 5. Auto-Indexing: Loops and Arrays

Arrays and loops go hand in hand, because of a powerful LabVIEW feature called auto-indexing. When you wire an array as an input to a loop and you enable auto-indexing, LabVIEW automatically applies the code inside the loop to each element in the array, one element at a time. This makes it very easy to apply the same code to every element in a group (array) of elements. The following program shows a simple example of this powerful feature.

1. Create a new VI whose front panel contains an array of string controls and a single string indicator (not an array of string indicators). Expand the array of string controls to show eight or ten of the controls. Wire the block diagram so that the strings entered by the user in the string controls are displayed one at a time in the string indicator, at a rate of one per second. Your program should work for any number of strings that the user decides to enter, and the program should end one second after the last string is displayed in the indicator. (Hint: Use a For Loop, with your array placed outside of the loop and nothing connected to the loop’s count terminal. If you enable indexing at the tunnel where the array data enters the loop, LabVIEW will automatically execute the loop the correct number of times, even though nothing is wired to the loop’s count terminal.) Save this VI as Lab9InputAutoIndex.vi and show me your working program.
The previous program used auto-indexing on a loop’s input tunnel. Auto-indexing is also often used on output tunnels of loops. In this case, we’re not feeding a pre-existing array into the loop. Rather, we’re using the loop to create a new array from scratch. This new array will contain one element for each iteration of the loop. The following program shows a simple example of this feature. Note: by default, LabVIEW enables auto-indexing for output tunnels on For Loops, and disables auto-indexing for output tunnels on While Loops. But you can easily enable auto-indexing on While Loops (or disable it on For Loops) by right-clicking the tunnel and selecting Enable Indexing.

1. Create a new VI whose front panel contains a numeric indicator and a STOP button. Wire the block diagram so that it generates and displays random numbers between 0 and 1, at a rate of one random number per second, until the user presses the STOP button. (This step is easy and does not require any arrays or auto-indexing.)

2. On the front panel, place an array of numeric indicators labeled Random Numbers. Expand the array to show eight or ten of the indicators. (We still want to be able to see each number as it’s generated, so don’t delete the numeric indicator from Step 1.) Modify the block diagram so that the program behaves as above, but now when the user stops the program, the front-panel array displays all of the random numbers that were generated while the program ran. Save this VI as Lab9OutputAutoIndex.vi and show me your working program.

Auto-indexing is powerful and easy to use, so you should take advantage of it whenever you can. However, there are times when auto-indexing doesn’t give us what we want. In particular, notice that the array produced by an auto-indexed output tunnel must contain one element for every loop iteration. What if we only want some of the loop iterations to create an element in the array? In such cases, you can’t use auto-indexing but can instead use the Build Array function to create an array that contains only the desired values.

1. Save a copy of Lab9OutputAutoIndex.vi under the new name Lab9BuildArray.vi. On the front panel, place another array of numeric indicators, labeled Random Numbers > 0.5. Expand the array to show eight or ten of the indicators.

2. Modify the block diagram so that the program behaves as above, but now when the user stops the program, this new front-panel array displays only the random numbers greater than 0.5 that were generated while the program ran. (Hint: Create a shift register on the loop and initialize it to hold an empty numeric array. This array will grow as the program runs. During each loop iteration, your code must examine the newest random number and decide whether or not to use the Build Array function to append this number to the growing array. Note that the Build Array function can be resized to have as many inputs as you want. For this program, you’ll want it to have two inputs.) Save this VI as Lab9BuildArray.vi and show me your working program.
Part 6. Two –Dimensional Arrays

So far, you’ve worked with one-dimensional arrays. Now let’s create a two-dimensional array, which has rows and columns. As you’ll learn, LabVIEW’s array functions can operate either on the individual elements of an array or on entire rows or columns of an array.

1. Create a new VI whose front panel contains an array of numeric controls. Right-click the border of the array and select Add Dimension. You should now be able to drag the array border both horizontally and vertically so that you have a two-dimensional array with rows and columns. Drag it until at least four columns and at least five rows are visible. Type a few numbers into each of the first few rows of the array.

2. On the front panel, place two numeric controls labeled Selected Row and Selected Column, and a numeric indicator labeled Value of Selected Element. Wire the block diagram so that when you run the program, the indicator displays the number in the cell whose row and column are selected.

3. On the front panel, place two new numeric indicators labeled Average of Selected Row and Average of Selected Column. When you run the program, it should display the average of the numbers in the selected row and the average of the numbers in the selected column. See the picture below for an illustration of how it should work. Your program should work for any number of values that the user decides to enter. (Hint: You don’t need much code on the block diagram to make this work. The key is to use the Index Array function to extract an entire row or an entire column of the array, and then to take the averages of those values. To learn how to do this, read LabVIEW’s detailed help on the Index Array function.)

4. Place your entire block diagram inside a While Loop so that the program will keep running until the user presses a STOP button on the front panel. Save this VI as Lab9TwoDimensionalArray.vi and show me your working program.

*** This lab had 10 named programs for me to check. If you didn’t finish them during class, finish them after class. Then upload all 10 programs, along with any related subVIs, to the website by the due date. Also turn in your lab sheets at the beginning of class.****