Introductory LabVIEW: Loops, Arrays, and Graphs

I. Loops

LabVIEW allows the repetition of section of the block diagram by providing two types of loops that are common to structured programming: the while loop and the for loop.

1. While Loop
   a. The while loop is available in the Functions>Structures sub-palette
   b. To implement a while loop, select it from the palette then click and drag a window around the section of the block diagram you want to repeat. When you release the mouse button, the while loop will appear.
   c. Two terminals appear with the while loop: the condition terminal and the iteration terminal
      i. The condition terminal is a Boolean input terminal that controls the iteration of the while loop.
         1. The VI will continue to repeat the diagram within the loop while the input to the condition terminal is TRUE.
         2. The condition terminal is checked at the end of each iteration, therefore the while loop always executes once.
      ii. The iteration terminal is a numeric output terminal that increments for each iteration of the loop
         1. The iteration terminal outputs a zero for the first iteration, a one for the second, ..., n-1 for the nth iteration
   d. Basically, a while loop is equivalent to the following pseudo-code:

      Do
      Execute diagram portion inside loop (which sets the condition)
      While the condition is TRUE

2. For Loop
   a. The for loop is also found in the Functions>Structures sub-palette
   b. You make a for loop in the same way you do a while loop; select it from the palette, then click and drag a window around the part of the diagram you want to repeat
   c. The for loop also has two terminals: the count terminal and the iteration terminal
      i. The iteration terminal for the for loop is identical to that of the while loop
      ii. The count terminal specifies the number of iterations the for loop will execute
   d. The for loop is equivalent to the following structured pseudocode

      For i = 0 to N-1
      Execute diagram portion inside the loop
II. Arrays

LabVIEW sports arrays of one or more dimensions with up to $2^{31}$ elements per dimension, memory permitting.

1. Creating arrays
   a. To create an array control or indicator on the front panel, select an array shell from the Controls>Array & Cluster sub-palette. Next, drag a control or indicator into the array shell.
      i. To create an array constant in the block diagram, select an array constant shell from the Functions>Array sub-palette. Drag a data constant into the array shell.
      ii. You can index elements in the array by using the up and down arrows.
   iii. To add a dimension to the array, right click on the index display and select **add dimension** from the pop-up menu.

2. Creating arrays with loops
   a. For and while loops can index and accumulate arrays at their boundaries automatically.
   b. When auto-indexing is enabled on a wire that passes out of a loop to an indicator, the loop will create and array and pass it to the indicator when the array is complete.
   c. If you only want the final value of an iterated calculation, auto-indexing can be disabled. The loop will pass a scalar value out of the loop to an indicator.
   d. You can enable or disable auto-indexing by right clicking on the box that appears when a wire passes through a loop boundary.
   e. Note that a wire that passes out of an auto-indexed loop is thicker than a wire that exits a loop with auto-indexing disabled. The thicker wire represents an array value.
   f. For loops enable auto-indexing by default.
   g. While loops disable auto-indexing by default.
   h. Arrays of two or more dimensions can be generated by nesting loops.

3. Using arrays
   a. Many array functions are available on the Functions>Array sub-palette.
   b. LabVIEW arithmetic functions are polymorphic, that is, inputs to these functions can be of different data types: array or scalar.
      i. An example of polymorphism for the **add** function is shown below.

![Diagram showing polymorphism for the add function in LabVIEW](image_url)
III. Graphs

1. A graph indicator is a two-dimensional display of one or more data arrays called plots or signals.

2. Graphs are found in the Controls>Graph sub-palette

3. There are two types of graphs: waveform graphs and x-y graphs, we will use waveform graphs extensively in this course

4. Simple graphs
   a. Since a graph is an array indicator, it can simply be wired to the output of an array control, an array function, or an auto-indexed loop
   b. Multiple dimension arrays can be wired similarly to produce multiple plots in the graph

5. Advanced graphs
   a. You can change the time scale in a graph by employing the aid of a bundle function
      i. A bundle function outputs a cluster datatype
         1. A cluster is to LabVIEW datatypes as a C structure is to C datatypes or as a telephone cable is to the multiple wires that exist inside it
         2. A cluster is a data type that groups other data types
      ii. A bundler then, is like a cable terminal –separate wires go in and a single cable that holds all the wires goes out
   b. Bundle terminals
      i. The top input terminal to a bundle function is the initial x value X0
      ii. The middle input to a bundle is the step in the x axis, Δx
      iii. The bottom input terminal is the data array
      iv. The output terminal is a single cluster that can be wired directly to a graph indicator
Lesson 5  Arrays and Graphs

Single-Plot Waveform Graph

(Xo = 0, ∆X = 1 assumed)

(Xo = 100, ∆X = 4)

Multiple-Plot Waveform Graph

(Xo = 0, ∆X = 1 assumed)

(Xo = 1, ∆X = 4)
Single-Plot XY Graph

Multiple-Plot XY Graph