Use **Highlight Execution** liberally in your future work. This LabVIEW feature is an invaluable debugging tool for troubleshooting VIs that are causing you problems.

**DO IT YOURSELF**

Write a VI called **Reaction Time** that measures a user's reaction time, defined as the elapsed time from when **Square LED** is lit until the user clicks the **Stop Button**. The front panel for this program includes a **Stop Button** and **Square LED** Boolean indicator, suitably enlarged, as well as a **Numeric Indicator** labeled **Reaction Time (ms)**, as shown below.

![ReactionTime.vi Front Panel](image)

**Reaction Time** should execute in the following sequence of steps: (1) The user clicks on the **Run** button, and then quickly moves the mouse cursor over to the **Stop Button**; when the **Run** button is pressed, **Square LED** and **Reaction Time (ms)** are initialized as **uninit** and **0**, respectively; (2) at a random time (within the range of 3 to 8 seconds) after the **Run** button has been clicked, **Square LED** becomes lit; (3) on the **Reaction Time (ms)** indicator, the value of the elapsed time in milliseconds since **Square LED** was lit is displayed continuously, then the displayed value is halted at the moment that the user clicks on the **Stop Button**; and (4) **Square LED** becomes unlit.

A few helpful tips follow.

1. A random number in the range from 0 up to 1 can be obtained using the **Random Number (0–1)** icon, which is found in **Functions >> Programming >> Numeric**.
2. Think carefully about the appropriate choice for the **Mechanical Action** option in the **Stop Button**'s pop-up window.
3. Properties of a front-panel object (e.g., its color, visibility, position) can be controlled from the block diagram using a **Property Node**. For example, to create a **Property Node** that lights **Square LED**, pop up on its block-diagram icon terminal and select...
Create>>Property Node>>Value. The resultant Property Node can then be placed anywhere on the block diagram (i.e., removed from the Square LED's icon terminal) and will appear as shown at the left in the next illustration. The outward-directed arrow indicates that this icon is presently configured as an indicator, which reads Square LED's current Value (TRUE or FALSE). To change this icon into a control, pop up on it and select Change To Write. Its arrow now is directed inward, indicating the Property Node is a control, as shown next at the right. By wiring a TRUE Boolean constant to this icon, Square LED's Value will be TRUE, causing the front-panel indicator to be lit. Conversely, wiring a FALSE Boolean constant to the icon will make Square LED unlit. You are free to place an unlimited number of Property Nodes on the block diagram for any given front-panel object.

Problems

1. Write a VI called Five Blinking LEDs (Sequence Structure), which sequentially illuminates five LEDs on its front panel. Place five Round LEDs on the front panel and label them 0 through 4. Using a Sequence Structure, develop a program that lights these LEDs one at a time in the order 0-1-2-3-4. Make each LED remain lit for 0.5 second and have the 0-1-2-3-4 lighting pattern repeat continuously until a front-panel Stop Button is clicked. To make multiple copies of an LED's block-diagram terminal, pop up on the terminal and select Create>>Property Node>>Value as described in the Do It Yourself project.

2. To demonstrate that a nested While Loop and Case Structure can accomplish the same function as a Sequence Structure, write a program called Five Blinking LEDs (State Machine). On the block diagram, use a nested While Loop and Case Structure to create code that repeatedly blinks five front-panel Round LEDs in the order 0-1-2-3-4 until a front-panel Stop Button is clicked, as outlined in Problem 1. You should find that this block-diagram architecture (which is called a state machine) provides a more elegant solution to the Five Blinking LED problem than a Sequence Structure (e.g., no Property Nodes are required). You may find the following icon useful: Quotient & Remainder in Functions>>Programming>>Numeric.

3. Write a program called Magic Stop Button, whose front panel initially appears blank, as shown in the next illustration.
As shown below, when this VI is run, a blinking Stop Button appears on the front panel until the user clicks on it. The Stop Button then disappears and the program stops. To write this program, first place a Stop Button on the front panel, pop up on it, and select Visible>>>Caption, and then make the caption read *Press this button to stop the VI.* Then pop up on the Stop Button and select Advanced>>>Hide Control, and then secure this choice by selecting Edit>>>Make Current Values Default and saving the VI.

Code the block diagram so that when the program is run, first the Stop Button becomes visible, then it blinks until it is clicked, and finally the Stop Button is made not visible. The features of the Stop Button (e.g., its visibility, blinking) can be controlled by creating appropriate Property Nodes.

4. The subdiagram shown below generates and plots 100 data samples on a Waveform Chart.
(a) Write a VI called **Two Charts (Simultaneous)**, which performs two such 100-sample plots simultaneously. That is, place two Waveform Charts on the front panel, labeled **Chart 1** and **Chart 2**, and program the block diagram so that the 100-sample plots on **Chart 1** and **Chart 2** are performed over the same time interval.

(b) Write a VI called **Two Charts (Data Dependency)**, which first completes the 100-sample plot on **Chart 1**, and then performs the 100-sample plot on **Chart 2**. Use artificial data dependency on the block diagram to accomplish this feat (i.e., do not use a Sequence Structure).

(c) Finally, add code to **Two Charts (Data Dependency)** so that the two Waveform Charts are cleared at the end of a run. To accomplish this task, place a single-frame Sequence Structure on the block diagram and use artificial data dependency to assure that this block-diagram object is the last item to execute before the VI completes a run. Within the Sequence Structure, place two plot-clearing Property Nodes, one associated with each Chart. The appropriate Property Node is made by popping up on the Chart's terminal and selecting **Create>>Property Node>>History Data**. After changing this Property Node to a control by selecting **Change to Write** in its pop-up menu, **Create>>Constant** will produce the needed input, called an **Empty Array**, as shown below.

```
Waveform Chart

<table>
<thead>
<tr>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>E</td>
</tr>
</tbody>
</table>

History
```

5. Write a program named **Parallel While Loops with Reset** in which two While Loops execute in parallel on the block diagram, simultaneously producing independent plots of random numbers (in the range 0 to 1) on two front-panel Waveform Charts, until stopped by the click of a single front-panel Stop Button. Label one of the Waveform Charts **Chart 1**, the other **Chart 2**.

   (a) First, try coding **Parallel While Loops with Reset** using the following block diagram to accomplish the intended goal (i.e., halt simultaneously executing While Loops with a single Stop Button). Run this program and show that it does not produce two simultaneous plots. Describe briefly what this diagram does do instead and explain why.
(b) For the correct diagram, one While Loop is stopped by wiring directly to the Stop Button's terminal and the other by wiring to a Value Property Node associated with this terminal. To produce the needed Property Node, pop up on the Stop Button's terminal and select Create>>Property Node>>Value. When wiring this Property Node, you will get a broken wire until the Stop Button's Mechanical Action is changed to something other than a "latch" mode (e.g., select Mechanical Action>>Switch When Pressed.) Complete the diagram for Parallel While Loops with Reset and verify that it runs as intended.

(c) When associated with a Value Property Node, the Stop Button must be in a "change value" (rather than a "latch") mode. Thus, when the Stop Button is pressed in order to stop the VI, the button will be left in its TRUE value as the VI completes execution, which is inconvenient for subsequent uses on this program. To remedy this problem, output a quantity from each While Loop and use artificial data dependency to reset the value of the Stop Button back to FALSE before the VI completes its execution. Run this final version of Parallel While Loops with Reset and verify that it executes as intended.

6. Code Real-Time Waveform Graph, whose block diagram is shown below. This diagram builds an array of data within a While Loop, and this ever-growing array is plotted on a Waveform Graph during each iteration.
On the front panel, turn off autoscaling on the Waveform Graph's x-axis and manually scale this axis to run from 0 to something large (e.g., 20000). Run Real-Time Waveform Graph and, based on your observations, identify the circumstances under which (e.g., after how many iterations) the design of this VI leads to poor performance. Explain why this diagram is a poor performer when run for many iterations.

7. Write a VI called For Loop Time (Icon vs. Mathscript), which compares the required array-building time $T$ of a LabVIEW For Loop icon and a Mathscript Node. To build the arrays, use the diagrams shown below.

For various values of $N$, what is $T$ for each diagram? Which diagram is the best performer?

8. The error in and error out terminals present on many LabVIEW icons can be used to sequence the execution of a collection of these icons via data degeneracy. As an example of this technique, use two Elapsed Time and one Prompt User for Input Express VIs (found in the subpalettes of Functions>>Express) to construct a program named Time To Press OK, which times how long it takes a user to click on a dialog-box OK button. Configure the Express VIs as shown below, then wire them together so that data dependency due to the error terminals causes them to execute sequentially from left to right. The Present (s) terminal of Elapsed Time outputs the (universal) time in seconds at which this icon executes, so the difference of this value from the two Elapsed Time icons gives the time elapsed from when the VI was started until the user clicks on the dialog-box OK button. Add code that calculates and displays this elapsed time (in seconds) on a front-panel numeric indicator named Time To Press OK (s).