# PHYC 500: Introduction to LabView

M.P. Hasselbeck, University of New Mexico

## Exercise 2 (v 1.2)

In Exercise 1, a VI was written to find the roots of a quadratic equation and take care of complex numbers. A Case Structure was used to alert the user if the roots were complex. The present exercise expands on this; the VI will continue to run and warn the user until real roots are obtained.

### While Loop

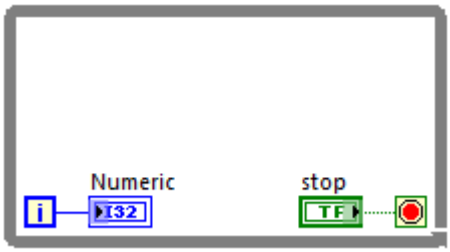
Open a blank VI and enable Context Help (CTRL-H or click **?** on the menu bar). Go to the Block Diagram, right-click, and select Structures: While Loop. A square dotted cursor will appear; use this to create a While Loop. This structure executes all the code contained within it over and over until it is commanded to stop.

Note that the Run arrow is broken; clicking on this broken arrow reveals that a "Conditional terminal is not wired". This terminal defines the stop condition for the While Loop and appears in the lower-right corner of the structure. The stop terminal is expecting a Boolean (T/F) input. Generate a Boolean control by right-clicking on the terminal and "Create Control". A Stop Button terminal appears along with a corresponding control on the Front Panel. This will result in a working VI, although with no useful code.


The blue terminal **[i]** in the lower-left is an integer that counts each time the While Loop executes. Right-click on it and create an indicator (see diagram below); this will also appear on the Front Panel. Note that the indicator icon is also colored blue and denoted as **I32**, which corresponds to a 32-bit integer. **Important:** The first iteration of a While Loop is always 0, not 1. Failure to realize that loops begin counting at 0 is one of the most common mistakes of new LabView programmers.

Go to the Front Panel and run the VI. Depending on the processor speed and computer resources, the loop will likely execute many millions of times in just a few seconds. Stop

the VI with the button on the panel.




It can be a dangerous programming practice to let a While Loop run as fast as possible. Its execution can be slowed down with the use of timing functions. Right-click the Block Diagram and select Timing: Wait (ms) and place this icon inside the While Loop. Right-click on the left icon input terminal and Create Constant. This will produce an integer constant of 0. Set the value to 1000 ms, i.e. 1-second and run the VI again. Verify that the loop count increments at 1-Hz.

Note: The functions **Wait (ms)** and **Wait until next ms Multiple** on the Timing palette are slightly different. The first is easiest to understand: the code will always pause for the specified time. The second function is used to precisely time the execution of loops. It forces the loop to execute exactly at the specified interval. This can be illustrated with an example. Assume there is code inside a While Loop that takes 50 ms to execute. If a Wait (ms) is placed in the loop and set for 100 ms, the loop will execute every 150 ms. If **Wait until next ms Multiple** is used with a value of 100 ms, the loop will execute every 100 ms.

Go to the Block Diagram and place the stop button icon outside the While Loop. Re-connect it to the conditional stop terminal inside the loop. The point where the wire passes through the While Loop structure is called a tunnel. Run the VI and notice that the stop button is unresponsive. The program can only be stopped with the Abort button on the menu bar. This illustrates an important aspect of data flow programming – when a loop is running it will ignore any input that is *outside* the structure. Data can be wired into a loop from outside, but once a loop is started anything present on input tunnels is ignored. Just as important: Data only propagates through output tunnels when the loop has stopped executing.

Open the saved VI from Exercise 1 that has the Case Structure with a user alert dialog. Go to the Block Diagram and place a While Loop around the entire code. Wire a connection from the Boolean output of the **<0** comparison function to the Stop terminal of the While Loop. This will appear as a green dotted line. The loop should continue executing for as long as complex roots exist, i.e. if the output is True. Right-click on the

stop terminal and change it to “Continue if True”. When real roots are found, the VI will stop. If the VI is run at this point, however, it will be difficult/impossible for the user to interact with the program if complex roots are detected. There is essentially no time to clear the dialog and enter new coefficients. The VI will be stuck in an infinite loop. To address this problem, place a 6-second wait inside the While Loop.

**Polling**

LabView is powerful and flexible enough to offer a variety of solutions to programming problems. Completely remove the Case Structure from the Block Diagram. Keep the **<0** Boolean output wired to the While Loop stop terminal. Right-click on the dotted green wire and Create: Indicator. A Boolean terminal will appear in the Block Diagram. Double-click this terminal to locate the corresponding indicator on the Front Panel. Right-click on this indicator, Replace: Boolean: Round LED. Select the LED with the arrow tool; hold down the Shift key and drag the LED to double its size. Use the Operate Value tool (pointed finger) to toggle between the two Boolean states. Use the paintbrush to change the True state from bright green to bright red. Select the False state; change the color from dark green to dark red. Change the indicator label to read “Complex Roots”. Remove the Wait function to allow the fastest possible operation. Run the VI and verify that the LED is on (bright red) when complex roots are present. This warning light alerts the user without the need to clear a dialog box by clicking the OK button.

Go back to the Block Diagram and create an indicator to display the loop count number as done above. Run the VI again and observe that many millions of cycles occur while waiting for user input. This action is called Polling, which consumes CPU resources even though nothing useful is taking place inside the program. This can be addressed by placing a Wait function in the While Loop, but a better solution is to use an Event Structure that is presented in Exercise 3. Save the current VI.