CLAD Homework 1 Solutions

1. **B**

   LabVIEW follows a dataflow model for running VIs. A block diagram node executes when it receives all required inputs. When a node executes, it produces output data and passes the data to the next node in the dataflow path. As a result, the value 7 is passed to the *multiplication* node and to the *subtraction* node. The value 2 is subtracted from 7 (equaling 5) and is passed to the *addition* node. The two values are added together (equaling 10) and is padded to the *multiplication* node. The two values are multiplied together (10 times 7) making Result equal to 70.

2. **B**

   LabVIEW follows a dataflow model for running VIs. A block diagram node executes when it receives all required inputs. When a node executes, it produces output data and passes the data to the next node in the dataflow path. As a result, 1 and 0 are passed to the *or* and the *and* nodes simultaneously. 1 or'ed with 0 equals 1 and 1 and'ed with 0 equals 0; therefore 1 and 0 are passed to the *exclusive or* node. 1 exclusive or'ed with 0 equals 1.

3. **C**

   VIs with the same name cannot be running at the same time, otherwise a conflict will occur.

4. **C**

   LabVIEW follows a dataflow model for running VIs. A block diagram node executes when it receives all required inputs. When a node executes, it produces output data and passes the data to the next node in the dataflow path. As a result, the *decrement* node executes and passes data to the *division* node giving it the second input it needs to execute. The data from the *division* node is passed to the *multiplication* node also giving it the second input it needs to execute. The data from the *multiplication* node is passed to the *addition* node giving it the second input it needs to execute. The data from the *addition* node is passed to the *Round Toward +Infinity* node giving it the input it needs to execute.

5. **B**

   When possible, it is always best to wire data directly to indicators. This helps maintain the Dataflow paradigm.

6. **C**

   Coercion dots indicate that a certain data type is being wired to terminal that accepts a different but compatible data type. When this happens, LabVIEW converts the data to the larger of the two data types. This requires the creation of a memory buffer to store the coerced data.

7. **D**

   Windows dialog buttons wait until a user releases before processing the click. When the user clicks and releases, the button returns to its default state. This behavior is similar to the Latch When Released mechanical action in LabVIEW.

8. **E**

   If you wire two different numeric data types to a numeric function that expects the inputs to be the same data type, LabVIEW converts one of the terminals to the same representation as the other terminal. LabVIEW chooses the representation that uses more bits. If the number of bits is the same, LabVIEW chooses unsigned over signed. Coercion dots appear on block diagram nodes to alert you that LabVIEW converted the value passed into the node to a different representation. Red coercion dots appear on block diagram nodes when you connect a wire of one numeric type to a terminal of a different numeric type. Blue coercion dots appear on the output terminals of Numeric functions when you manually configure fixed-point output settings for a function. Coercion dots can cause a VI to use more memory and increase its run time because of the extra memory buffers it has to create to store the coerced data. Try to keep data types consistent in the VIs you create. To use memory more efficiently, eliminate coercion dots at numeric terminals.

9. **B**

   Functions and VIs execute as soon as all of their inputs have data. Thus, it is the flow of data through the program that determines the order of execution.

10. **C**

    Since the mechanical action is set to Switch Until Released, two events are generated when a user clicks and releases. The first event is the FALSE to TRUE transition, and the second is the TRUE to FALSE transition. Latch actions are specifically designed to reset the value of the button after the change is read without generating a second event.