NAME _____

EET 2259 Lab 3
The Boolean Data Type

OBJECTIVES

- Understand the differences between numeric data and Boolean data.
-Write programs using LabVIEW's Boolean controls and indicators, Boolean constants, and Boolean functions.
-Using a DAQ card, perform simple digital input and output tasks.
-Select the appropriate mechanical action for LabVIEW switches and buttons.
-Write programs using LabVIEW's comparison functions, including the Select function.

Part 1. Boolean Data

As you know from the previous lab, objects of the numeric data type have many, many different possible values. To list a few possible values, a numeric object (such as a numeric constant) could be set equal to 1, or to 0.249726, or to -1097.1236, and so on.

In contrast, objects of the Boolean data type have just two possible values: **true** and **false**. Sometimes it's convenient to use other words (such as **on** and **off**) in place of **true** and **false**. In the table below, the entries in the first column mean the same thing as each other. All of the entries in the second column also mean the same thing as each other.

Word pairs used to name the two possible Boolean values

True	False
On	Off
Yes	No
1	0

The Boolean data type is useful because in many situations we're only interested in knowing the answer to a true-false question. For instance, if you're trying to get into a bar, the bouncer doesn't care about your exact age. He only cares about whether you're at least 21 years old. As far as the bouncer is concerned, all people fall into two categories: those who are at least 21, and those who aren't.

Another example is a temperature-monitoring system on a freezer that stores food. Suppose the food needs to be kept colder than 20 degrees Fahrenheit. Picture a red light on the door of the freezer, which stays off as long as the temperature inside is below 20 degrees, but lights up if the temperature rises above 20 degrees. The light is an example of a Boolean indicator.

Part 2. Boolean Controls and Indicators

For Boolean controls, LabVIEW uses buttons and switches that can be set to two positions. For Boolean indicators, LabVIEW uses LEDs. A dark LED indicates a value of **false**, while a lit LED indicates **true.**

Practice with Boolean Controls and Indicators

1. In LabVIEW, create a new blank VI and click anywhere on its front panel.

2. Under the **Modern** category of the Controls Palette, click the **Boolean** icon. This will open a palette containing about ten kinds of buttons and switches (which are Boolean

controls) and two kinds of LEDs (which are Boolean indicators). The buttons and switches differ from each other primarily in their appearance, but some of them also differ slightly in their behavior.

3. Place a **Push Button** and a **round LED** on the front panel.

4. On the block diagram, connect a wire between the push button and the LED.

5. Run your program continuously, and notice that the LED lights up when the push button is switched on, and the LED stays dark when the push button is switched off.

6. Press the Abort button (which looks like a STOP sign) to stop the program.

7. Let's change the LED's color. Move your mouse over the LED and **right-click** to open its shortcut menu. From the shortcut menu choose **Properties**. Notice the two color boxes showing you what color the LED will be when it's turned on or turned off. Click these boxes and choose new colors—make the LED bright blue when it's turned on and dark blue when it's turned off. Run the VI again.

Part 3. Boolean Constants

You know from the previous lab that you can place a numeric constant on a VI's block diagram and then set that constant equal to any of its possible values. Not surprisingly, you can also place a **Boolean constant** on a block diagram and set it equal to any of its possible values. Since Boolean constants have only two possible values, you don't have a whole lot of choices here: you can place a **true** Boolean constant or a **false** Boolean constant. Boolean constants aren't used nearly as often as numeric constants, but there are times when they come in handy.

Practice with Boolean Constants

1. Create a new blank VI, and place two LEDs on its front panel.

2. Click anywhere on the VI's block diagram.

3. Under the **Programming** category of the Functions Palette, click the **Boolean** icon. This will open a palette containing fourteen Boolean functions (which we'll look at soon) and the two Boolean constants, named **True Constant** and **False Constant**.

4. Place one True Constant and one False Constant on your block diagram.

5. Notice that if you move your mouse over either of the Boolean constants, the mouse pointer changes to a hand. If you then click the mouse, the Boolean constant will change into the opposite kind (**true** changes to **false**, and **false** changes to **true**.) This makes it easy for you to change a Boolean constant if you placed the wrong one on your block diagram. But it also makes it easy for you to accidentally change the value of a Boolean constant, so be careful about clicking near Boolean constants.

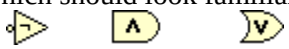
6. Wire each of your Boolean constants to one of the LEDs.

7. Run your program. Not surprisingly, you'll find that the LED connected to the True Constant lights up, and the LED connected to the False Constant stays dark.

Part 4. Boolean Functions

Now let's turn to Boolean functions. As you know from the previous lab, LabVIEW has many numeric functions that you can place on a VI's block diagram. A numeric function, such as **Add**, takes numeric values as its inputs, and produces a numeric value as its output. Similarly, a **Boolean function** takes Boolean values as its inputs, and produces a Boolean value as its output.

These Boolean functions are the logical functions that you're familiar with from your digital courses: **Not**, **And**, **Or**, and so on. The three functions just named are represented by the following block-diagram symbols, which should look familiar:



On the Functions Palette, you'll find these three Boolean functions (and several others) on the same palette where you found the True Constant and the False Constant. Recall that the other Boolean functions (such as **Exclusive Or** and **Not And**) can all be built up out the three basic functions (**Not**, **And**, and **Or**). For example, the **Not And** function is equivalent to a combination of the **Not** function with the **And** function. So I'll concentrate on the three basic ones, but you should feel free to use other ones whenever you think they'll make your job easier.

Practice with Boolean Functions

1. Create a VI whose front panel has a push button labeled **Push Button 1** and an LED. Wire the block diagram so that the LED lights up when the button is switched on, and the LED stays dark when the button is off.

2. Modify this VI so that the LED lights up when the button is switched off, and the LED stays dark when the button is switched on.

3. Modify this VI by adding another push button labeled **Push Button 2** to the front panel. Rewire the block diagram so that the LED lights up when both button are switched on, and the LED stays dark the rest of the time.

4. Modify this VI so that the LED lights up when either button is switched on, and the LED stays dark the rest of the time.

5. Modify this VI by adding another push button labeled **Push Button 3** and another LED to the front panel. Give one LED the label **All On**, and give the other LED the label **All Off**. Rewire the block diagram so that:
   - The LED labeled **All On** lights up when all three buttons are switched on, and it stays dark in every other case.
   - The LED labeled **All Off** lights up when all three buttons are switched off, and it stays dark in every other case.

   Save this VI as **Lab3BooleanFunction.vi**, and **show me your working program**.

Part 5. Digital Input and Digital Output

In previous labs you used the DAQ Assistant for **analog input** and **analog output,** which means inputting or outputting numeric values such as voltages. You can also use the DAQ Assistant for **digital input** and **digital output,** which means inputting or outputting Boolean values.

For this part of the lab you'll use the myDAQ. You'll also use the Boolean inputs (switches) and Boolean outputs (LEDs) on the red trainer that you've used in other classes. Follow these steps:

1. Plug your myDAQ into your computer's USB port.

2. On the red digital-analog trainer, locate the bank of eight data switches (below the breadboard) and the bank of eight LEDs (in the upper right-hand corner).

3. Run a black wire from the trainer's **GND** terminal to the myDAQ's screw terminal labeled **DGND.** (This stands for **digital ground**, in contrast to the two screw terminals labeled **AGND**, which stands for **analog ground**.) In making this connection, you've established a common ground between the trainer and the myDAQ.

4. Run a wire from one of the trainer's data switches to the myDAQ's screw terminal labeled **0** in the **DIO** group of screw terminals. (**DIO** stands for **digital input/output.**)

5. Turn the trainer's POWER switch on.

6. Start a new blank VI. Place a **DAQ Assistant Express VI** on the block diagram. In the dialog box that opens, select **Acquire Signals**, then select **Digital Input**, and then select **Line Input**.

7. Next, LabVIEW will let you choose which of the myDAQ's eight digital I/O lines you want to use. Select **port0/line0**, and then click the **Finish** button.

8. Next you'll see a dialog box that lets you set some parameters before you make your measurement. You don't need to change any of these settings, so just click **OK**.

9. On your VI's block diagram, your DAQ Assistant will have an output labeled **data**. Place a **From DDT Express VI** on the block diagram. (Recall from the previous lab that this Express VI converts data from the dynamic data type to other data types.)

10. A dialog box will open to let you choose the data type that you want to convert to.
    - In the box labeled **Resulting data type**, select **Single scalar**.
    - For the **Scalar data type**, select **Boolean (TRUE and FALSE)**.
    - Make sure that **Channel** is set to 0.
    - Then click **OK**.

11. Run a wire between the DAQ Assistant's data output and the From DDT's input. LabVIEW will automatically create another block before the From DDT. (To understand why LabVIEW adds this extra block, you need to understand something about arrays, which we'll study in a few weeks.)

12. On your VI's front panel, place an LED labeled **Switch Status**. On the block diagram, wire this LED to the From DDT's output.

13. Run the VI. You should see that your VI's LED lights up or stays dark, depending on whether the trainer's data switch is turned on or off.

14. (*In this step, and at all other times, be sure to turn off the trainer's POWER switch whenever you are connecting or disconnecting a wire between the trainer and the myDAQ.*) Connect another of the trainer's data switches to the myDAQ's **DIO1** terminal. Using another DAQ Assistant configured to perform digital input on the myDAQ's **port0/line1**, modify your block diagram so that the VI's LED lights up if both trainer data switches are turned on, but the VI's LED stays dark otherwise. Save this VI as **Lab3DigitalInput.vi**, and **show me your working program**.

You just used LabVIEW to read a digital input. Now let's use LabVIEW to produce a digital output.

1. Turn the trainer's POWER switch off.

2. Remove the wires connecting the trainer's data switches to the myDAQ's digital I/O terminals. (But leave the black wire establishing a common ground.)

3. Run a wire from the myDAQ's DIO0 screw terminal to one of the trainer's eight LEDs.

4. Turn the trainer's POWER switch on.

5. Start a new blank VI, and place a push button on the VI's front panel

6. Place a **DAQ Assistant Express VI** on the block diagram. Select **Generate Signals**, then select **Digital Output**, and then select **Line Output**. Then select **port0/line0** and click the **Finish** button.

7. Without changing any of the settings in the next dialog box, click the **OK** button.

8. On your VI's block diagram, your DAQ Assistant will have an input labeled **data**. Place a **To DDT Express VI** on the block diagram, and configure it to convert a single scalar whose data type is Boolean. Then connect your push button to the To DDT's input, and connect the To DDT's output to the DAQ Assistant's input. As above, LabVIEW will automatically create another block to make this connection work correctly.

9. Run the VI. You should see that the trainer's LED lights up or stays dark, depending on whether the VI's push button is switched on or off.

10. Add another push button to your VI's front panel. Then modify your block diagram so that the trainer's LED lights up if either (or both) of the VI's push buttons are turned on, but the trainer's LED stays dark if both push buttons are turned off. Save this VI as **Lab3DigitalOutput.vi**, and **show me your working program**.

Part 6. Mechanical Action of Buttons and Switches in LabVIEW

Buttons and switches in the real world can behave in different ways. For example, consider the difference between light switches and car horn buttons.

- When you enter a dark room and flip a light switch, the lights turn on. After you remove your hand from the switch, the switch stays in the "on" position, and the lights stay on.
- On the other hand, when you press the button for a car horn, the horn stays on for as long as you press the button, and the horn turns off when you release the button.

This difference between a light switch's behavior and a car horn's behavior is an example of what is called the **mechanical action** of a button or switch.

Here's another example: a doorbell button that causes the bell to ring only once, even if you hold the button down for a long time. (I mean the "ding-dong" kind of doorbell, not a buzzer that continues to buzz for as long as you press the switch.) If you want to ring the bell repeatedly, you must press and release the button repeatedly. This behavior is different from the light switch's behavior and the car horn's behavior.

Switches and buttons in LabVIEW are designed to behave like real switches and buttons. For any switch or button, LabVIEW lets you choose from six different mechanical actions:

- Switch When Pressed
- Switch When Released
- Switch Until Released
- Latch When Pressed
- Latch When Released
- Latch Until Released

Note that three of these contain the word **switch**, and the other three contain the word **latch**. What's the difference? Using the examples from above, a light switch switches, but a doorbell button latches. When you flip a light switch it changes the circuit's state from off to on, and then the circuit stays in that new state until you flip the switch again: that's an example of **switching**. On the other hand, when you press a doorbell button it sends a brief signal telling the circuit to do something once, and then you must release and press the button again if you want the circuit to perform that action again: that's an example of **latching**.

Looking again at the list of mechanical actions, note that some of them contain the word **pressed** and some contain the word **released**. A LabVIEW user will be using a mouse to operate switches in LabVIEW. The question here is, should something happen when the user presses the mouse button, or when she releases the mouse button?

Combining the different possibilities, the designers of LabVIEW came up with the six mechanical actions listed above. For most switches and buttons, the default behavior is Switch When Pressed.

To explore the possibilities, follow these steps:

1. Create a new blank VI. On the front panel, place three push buttons. Label these **Light switch**, **Doorbell button**, and **Car horn button**. Also place three LEDs labeled **Light**, **Doorbell**, and **Horn**. On the block diagram, wire each push button to the corresponding LED.

2. From the **Functions > Programming> Timing** palette, place a **Time Delay** on your block diagram and set the delay to 0.1 second.

3. Right-click one of your push buttons and select **Properties** from the shortcut menu. This will open a dialog box with five tabs, labeled **Appearance**, **Operation**, **Documentation**, and so on. Choose the **Operation** tab. You should then see a list of the six button behaviors.

4. Choose each of the six behaviors, one at a time. Read the description in the **Behavior Explanation** box, and notice that each explanation includes the standard electrical symbol for that behavior. Use the switch in the **Preview Selected Behavior** area to see each behavior in action. When previewing a behavior, **slowly** press and release the mouse button so that you can see whether something happens when you **press** the mouse button or when you **release** the mouse button. You should easily be able to see the difference between most of the mechanical actions.

5. When you're finished previewing the six mechanical actions, set each push button's mechanical action to the correct setting so that it behaves in the proper way. Remember, one of the push buttons should behave like a light switch, one should behave like a doorbell button, and one should behave like a car horn button. Save this VI as **Lab3MechanicalAction.vi** and **show me your working program**.

Part 7. Numeric and Boolean Data Types Compared

The following table summarizes some important similarities and differences between the numeric data type (in the left column) and the Boolean data type (in the right column). You should memorize the information in this table.

	Numeric	**Boolean**
Controls	Numeric control, slides, knob, dial.	Switches and buttons.
Indicators	Numeric indicator, gauge, meter, progress bars.	LEDs.
Values	Many possible values, such as 0, 1, 2, … May be integer values or floating point values.	Only two possible values: true and false.
Functions	Add, Subtract, Multiply, Divide, Square Root, Absolute Value, Round, and other numeric functions.	And, Or, Not, and other Boolean functions.
Color on Block Diagram	Blue (for integers) or orange (for floating-point numbers).	Green.

Part 8. Mixed-Type Functions

Each function listed in the **Functions** row of the table above has the same data type for its inputs and its output. For example, the **Or** function takes Boolean inputs and produces a Boolean output. The **Multiply** function takes numeric inputs and produces a numeric output.

On the other hand, some LabVIEW functions take inputs of one data type and produce an output of another data type. Probably the most important of these are the Comparison functions, such as **Equal?**, **Not Equal?**, **Greater?**, and **Less?**. These functions can take numeric inputs, or Boolean inputs, or string inputs, but they always produce a Boolean output (**true** or **false**). For example, if you wire two numeric values to the inputs of an **Equal?** function, then the function's output will be **true** if the two numeric values are equal, but the output will be **false** if the two numeric values

are not equal. This function is very useful, because it lets you write a program that compares two numbers and then performs some operation if the two numbers are equal to each other. For example, perhaps you're writing a program that uses a sensor to count the parts moving down an assembly line into a shipping crate, and your program must alert the operator that the crate is full when 50 parts have passed.

It's most common to use these Comparison functions with numeric inputs, but you can have inputs of the other types. For instance, you could use the **Equal?** function to check whether two strings are the same as each other. This could be used, for example, to see whether the password that a user typed in is equal to the correct password.

Practice with Comparison Functions 

1. First, let's see where to find Comparison functions. In LabVIEW, create a new blank VI, and click anywhere on the VI's block diagram.

2. Under the **Programming** category of the Functions Palette, click the **Comparison** icon. This will open a palette containing about 25 functions with icons such as the ones shown above. You can probably figure out the meanings of many of these functions from their names and icons.

3. Create a VI whose front panel has one dial and ten LEDs. Configure the dial so that it can only be set to integer values between 0 and 9. Adjust the colors of the LEDs so that you have one LED for each color in the resistor color code (black, brown, red, and so on). Wire the block diagram so that the correct LED lights up and the other nine LEDs stay dark, depending on the dial's value. Save this VI as **Lab3ColorCode.vi**, and **show me your working program**.

The next program will require a little more thinking on your part.

1. Create a VI whose front panel has a dial labeled **Input value** and a green LED labeled **In range**. Adjust the dial's representation and scale so that it can only be set to integers between 0 and 100. Wire the block diagram so that the LED lights up when the dial is set to a value greater than or equal to 60, and the LED stays dark the rest of the time.

2. Modify the VI so that the LED lights up when the dial is set to a value between 60 and 80 (in other words, greater than or equal to 60 but less than or equal to 80), and the LED stays dark the rest of the time. (Hint: For this step and the following steps you'll need to use some of the Boolean functions that you learned about earlier, as well as the ≥ and ≤ functions.)

3. Modify this VI by adding a red LED labeled **Out of range** to the front panel. The green LED should continue to behave as above, but the red LED should light up if the dial is set to a value less than 60 or greater than 80.

4. Modify this VI by adding a push button labeled **Check value?** to the front panel. Rewire the block diagram so that the VI behaves as above when the push button is turned on, but the LEDs don't light up at all when the push button is switched off. Save this VI as **Lab3CheckRange.vi**, and **show me your working program**.

How about a program that tells whether an integer is even or odd?

1. Create a new VI whose front panel has a numeric control labeled **Integer** and two LEDs, one labeled **Even** and one labeled **Odd**. Configure the numeric control so that it can only be set to integer values. Wire the block diagram so that the correct LED lights up depending on whether the integer in the control is even or odd. (Hint: There are several ways to do this. Probably the easiest way relies on LabVIEW's **Quotient & Remainder** function.) Save this VI as **Lab3EvenOdd.vi**, and **show me your working program**.

Part 9. The Select Function

A very useful function on the Comparison palette is the **Select** function. Although it is located on the Comparison palette, it is very different from most of the other comparison functions. The Select function has three inputs and one output. One of the inputs, called **s**, must be Boolean. The other two inputs, called **t** and **f**, can be either Boolean, numeric, or string, but they must be the same type as each other.

As the function's name implies, this function selects one of its inputs. Here's how it works: If input **s** is true, then the value at the **t** input is passed through to the function's output. But if input **s** is false, then the value at the **f** input is passed through to the function's output.

Practice with the Select Function 

1. Create a VI whose front panel has a dial labeled **Dial**, a horizontal pointer slide labeled **Slide**, a horizontal toggle switch labeled **Dial or Slide?**, and a numeric indicator labeled **Selected value**.

2. Wire the block diagram so that the numeric indicator displays the dial's value if the toggle switch points to the word "Dial," and displays the slide's value if the toggle switch points to the word "Slide." Save this VI as **Lab3Select.vi**, and **show me your working program**.

In Lab #2 you wrote Lab2Randoms.vi and Lab2RandomIntegers.vi. Both programs produced random numbers in the range specified by the user. But in one program the random numbers were floating-point, while in the other program they were integers. Let's use the Select function to combine these two programs into one.

1. Create a new VI whose front panel contains:
   - two numeric controls labeled **Lower Limit** and **Upper Limit**
   - a numeric indicator labeled **Random Number**
   - a horizontal toggle switch labeled **Floating-point or Integer?**

   Wire the block diagram so that the numeric indicator displays a random number in the range specified by the user and of the type (floating-point or integer) specified by the user. Save this VI as **Lab3Randoms.vi**, and **show me your working program**.

Your final program in this lab will find the maximum of three numbers entered by the user.

1. Create a VI whose front panel has two numeric controls labeled **a** and **b**, and one numeric indicator labeled **Maximum**. Wire the block diagram so that the numeric indicator displays the greater of the two numeric values. In other words, it displays the value of *a* if *a* is greater than *b*, and it displays the value of *b* if *b* is greater than *a*. It can display either value if *a* equals *b*. (Hint: Use a Select function and a Greater? function. You are not allowed to use the Max&Min function or the Array Max&Min function.)

2. Modify this VI by adding another numeric control labeled **c** to the front panel. Rewire the block diagram so that the numeric indicator displays the maximum of *a*, *b*, and *c*. As above, do not use the Max&Min or Array Max&Min function. Save this VI as **Lab3Maximum.vi**, and **show me your working program**.

_____

Part 10. Review Questions (Fill in the blanks.)

1. When you want to input or output a digital value, you must use one of the myDAQ's digital input/output lines. You do this by connecting a wire to one of the myDAQ's screw terminals. To input the value into the myDAQ's digital input/output port0/line0, you should connect your wire to the myDAQ's screw terminal labeled _____.

2. When you place a button or switch on the front panel, it has a default mechanical action. Different types of switches and buttons have different default actions. For each item listed below, indicate the default mechanical action by placing a checkmark in the correct column.

	Switch When Pressed	Switch When Released	Switch Until Released	Latch When Pressed	Latch When Released	Latch Until Released
						
						
						
						
						
						
						

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