A. Case Structures

A Case structure, shown at left, has two or more subdiagrams, or cases. Only one subdiagram is visible at a time, and the structure executes only one case at a time. An input value determines which subdiagram executes. The Case structure is similar to case statements or if...then...else statements in text-based programming languages.

The case selector identifier at the top of the Case structure, shown at left, contains the case selector identifier in the center and decrement and increment arrow buttons on each side. Use the decrement and increment arrow buttons to scroll through the available cases.

Wire an input value, or selector, to the selector terminal, shown at left, to determine which case executes. You must wire an integer, Boolean value, string, or enumerated type value to the selector terminal. You can position the selector terminal anywhere on the left border of the Case structure. If the selector terminal is Boolean, the structure has a TRUE case and a FALSE case. If the selector terminal is an integer, string, or enumerated type value, the structure can have up to $2^{31}-1$ cases.

You can specify a default case for the Case structure. You must specify a default case to handle out-of-range values or explicitly list every possible input value. For example, if you specified cases for 1, 2, and 3 but you get an input of 4, the Case structure executes the default case.

Right-click the Case structure border to add, duplicate, remove, or rearrange cases and to select a default case.

Input and Output Tunnels

You can create multiple input and output tunnels for a Case structure. Inputs are available to all subdiagrams, but subdiagrams do not need to use each input. When you create an output tunnel in one case, tunnels appear at the same position on the border in all the other cases. If at least one output tunnel is not defined, all output tunnels on the structure appear as white squares. You can define a different data source for the same output tunnel in each case, but the data types must be compatible.

Wire to the output tunnel for each unwired case, clicking the tunnel each time. You also can wire constants or controls to unwired cases by right-clicking the tunnel and selecting Create>Constant or Create>Control from the shortcut menu.
Examples

In the following examples, the numerics pass through tunnels to the Case structure and are either added or subtracted, depending on the value wired to the selector terminal.

**Boolean Case Structure**
The following example is a Boolean Case structure.

If the Boolean control wired to the selector terminal is TRUE, the VI adds the numerics. Otherwise, the VI subtracts the numerics.

**Integer Case Structure**
The following example is an integer Case structure.

*Integer* is a text ring control located on the *Controls*»*Ring & Enum* palette that associates numerics with text items. If the text ring control wired to the selector terminal is 0 (*add*), the VI adds the numerics. If the value is 1 (*subtract*), the VI subtracts the numerics.
String Case Structure
The following example is a string Case structure.

If String is add, the VI adds the numerics. If String is subtract, the VI subtracts the numerics.

Enum Case Structure
The following example is an enumerated constant Case structure.

Error Case Structure
The following example is an error cluster Case structure.

When an error cluster is wired to the selection terminal, the Case structure recognizes only the Status Boolean of the cluster.
Selecting a Case

To select a case, type the values in the case selector identifier or use the Labeling tool to edit the values.

Specify a single value or lists and ranges of values to select the case. For lists, use commas to separate values. Specify a range as 10..20, meaning all numbers from 10 to 20 inclusively. You also can use open-ended ranges. For example, .100 represents all numbers less than or equal to 100. You also can combine lists and ranges, for example .5, 6, 7..10, 12, 13, 14. When you enter a selector that contains overlapping ranges, the Case structure redisplays the selector in a more compact form. The previous example redisplays as .10, 12..14.

If you enter a selector value that is not the same type as the object wired to the selector terminal, the value appears red to indicate that you must delete or edit the value before the structure can execute, and the VI will not run. Also, because of the possible round-off error inherent in floating-point arithmetic, you cannot use floating-point numerics as case selector values. If you wire a floating-point value to the case, LabVIEW rounds the value to the nearest even integer. If you type a floating-point value in the case selector, the value appears red to indicate that you must delete or edit the value before the structure can execute.
Exercise 6-1  Square Root VI

Objective:  To use the Case structure.

Complete the following steps to build a VI that checks whether a number is positive. If it is, the VI calculates the square root of the number. Otherwise, the VI returns an error message.

Front Panel

1. Open a new VI and build the following front panel.

Block Diagram

2. Build the following block diagram.

   a. Place a Case structure located on the Functions » Structures palette.
   b. Click the decrement or increment arrow button to select the FALSE case.
   c. Place the Greater or Equal to 0? function located on the Functions » Comparison palette. This function returns TRUE if Numeric is greater than or equal to 0.
   d. Right-click the numeric constant and select Format & Precision from the shortcut menu. Set Digits of Precision to 1, select Floating Point Notation, and click the OK button. This ensures there is no data conversion between the constant and the numeric indicator outside the Case structure.
   e. Place the One Button Dialog function located on the Functions » Time & Dialog palette. This function displays a dialog box that will contain the message Error...Negative Number.
   f. Right-click the message terminal of the One Button Dialog function, select Create » Constant from the shortcut menu, type Error...Negative Number, and press the <Enter> key. Refer to Lesson 7, Strings and File I/O, for more information about strings.
g. Select the TRUE case and place the Square Root function located on the **Functions**\(\text{ Numeric}\) palette, as shown in the following block diagram. This function returns the square root of **Numeric**.

3. Save the VI as **Square Root.vi**.
4. Display the front panel and run the VI.

⚠️ **Caution** Do not run this VI continuously. Under certain circumstances, continuously running this VI could result in an endless loop.

   If **Numeric** is positive, the VI executes the TRUE case and returns the square root of **Numeric**. If **Numeric** is negative, the VI executes the FALSE case, returns \(-99999.0\), and displays a dialog box with the message **Error...Negative Number**.

5. Close the VI.

**End of Exercise 6-1**
Exercise 6-2  Temperature Control VI

Objective: To use the Case structure.

Complete the following steps to build a VI that detects when a temperature is out of range. If the temperature exceeds the limit, an LED turns on and a beep sounds.

Front Panel

1. Open the Temperature Running Average VI, which you built in Exercise 4-5.
2. Modify the front panel as follows.

3. Right-click the chart display and select Visible Items»Digital Display from the shortcut menu to display the digital values.
4. Save the VI as Temperature Control.vi.
Lesson 6 Case and Sequence Structures

Block Diagram

5. Modify the block diagram as follows. The FALSE case of the Case structure is empty.

![Block Diagram Image]

a. Place the Greater? function located on the Functions > Comparison palette. This function returns TRUE if the temperature exceeds High Limit. Otherwise, the function returns FALSE.

b. Place the Beep VI located on the Functions > Graphics & Sound > Sound palette. This VI sounds a beep if the selector terminal of the Case structure receives TRUE.

c. (MacOS) Provide values for the Beep VI input terminals.

6. Save the VI because you will use this VI later in the course.

7. Display the front panel, enter 80 in High Limit, and run the VI.

If the VI returns a temperature greater than High Limit, Warning turns on, the VI executes the TRUE case, and a beep sounds. If the temperature is less than High Limit, Warning turns off, the VI executes the FALSE case, and no beep sounds.

8. Close the VI.

End of Exercise 6-2
B. Sequence Structures

A Sequence structure, shown at left, contains one or more subdiagrams, or frames, which execute in sequential order. A Sequence structure executes frame 0, then frame 1, then frame 2, until the last frame executes. The Sequence structure does not complete execution or return any data until the last frame executes.

The sequence selector identifier at the top of the Sequence structure, shown at left, contains the current frame number and range of frames in the center and decrement and increment arrow buttons on each side. For example, in the sequence selector identifier shown at left, 0 is the current frame number and [0..2] is the range of frames. Click the decrement and increment arrow buttons to scroll through the available frames.

Use the Sequence structure to control the execution order when natural data dependency does not exist. A node that receives data from another node depends on the other node for data and always executes after the other node completes execution. Within each frame of a Sequence structure, as in the rest of the block diagram, data dependency determines the execution order of nodes.

The tunnels of Sequence structures can have only one data source, unlike Case structures. The output can emit from any frame, but data leave the Sequence structure only when all frames complete execution, not when the individual frames complete execution. As with Case structures, data at input tunnels are available to all frames.

Sequence Locals

To pass data from one frame to any subsequent frame, use a sequence local terminal, shown at left. An outward-pointing arrow appears in the sequence local terminal of the frame that contains the data source. The terminal in subsequent frames contains an inward-pointing arrow, indicating that the terminal is a data source for that frame. You cannot use the sequence local terminal in frames that precede the first frame where you wired the sequence local. Right-click the structure border and select Add Sequence Local from the shortcut menu to create a sequence local.
The following example shows a three-frame Sequence structure. A sequence local in frame 1 passes the value that the Thermometer VI returns to frame 2, as indicated by the arrow pointing into frame 2. This value is not available in frame 0, as indicated by the dimmed square.
Exercise 6-3    Time to Match VI

Objective: To use the Sequence structure.

Complete the following steps to build a VI that computes the time it takes to generate a random number that matches a number you specify.

Front Panel

1. Open the Auto Match VI, which you built in Exercise 4-3.
2. Modify the front panel as follows.
   a. Change Number to Match, Current Number, and # of iterations to I32 representation.
   b. Change Time to Match to DBL representation and 3 digits of precision.
3. Save the VI as Time to Match.vi.
Lesson 6 Case and Sequence Structures

Block Diagram

4. Modify the block diagram as follows.

![Block Diagram Image]

a. Place a Sequence structure located on the Functions » Structures palette.
b. Right-click the structure border and select Add Frame After from the shortcut menu to add a frame.
c. Place the Tick Count (ms) function located on the Functions » Time & Dialog palette. This function reads the current value of the operating system clock and returns the value in milliseconds.

5. Save the VI.

6. Display the front panel, enter a number in Number to Match, and run the VI.

In frame 0, the VI executes the While Loop while Current Number does not match Number to Match. In frame 1, the Tick Count (ms) function reads the operating system clock. The VI subtracts the new value from the initial time read and returns the elapsed time in seconds.

Note If Time to Match is always 0.000, the VI might be running too quickly. Either run the VI with execution highlighting enabled or increase the numeric constant wired to the Multiply function in frame 0 to a large value, such as 1000000.

7. Close the VI.

End of Exercise 6-3
C. Formula and Expression Nodes

Use the Formula Node and Expression Node to perform mathematical operations within the LabVIEW environment. For more advanced functionality, you can link to the mathematics applications HiQ and MATLAB to develop equations. HiQ and MATLAB are software packages that help you organize and visualize real-world math, science, and engineering problems.

Expression Nodes

Use Expression Nodes to calculate expressions, or equations, that contain a single variable. Expression Nodes are useful when an equation has only one variable but is otherwise complicated. Expression Nodes use the value you pass to the input terminal as the value of the variable. The output terminal returns the value of the calculation.

Formula Nodes

The Formula Node is a convenient text-based node you can use to perform mathematical operations on the block diagram. Formula Nodes are useful for equations that have many variables or are otherwise complicated and for using existing text-based code. You can copy and paste the existing text-based code into a Formula Node rather than recreating it graphically on the block diagram.

Create the input and output terminals of the Formula Node by right-clicking the border of the node and selecting Add Input or Add Output from the shortcut menu. Type the equation in the structure. Each equation statement must terminate with a semicolon (;).

The Formula Node can perform many different operations. Refer to the LabVIEW Help for more information about functions, operations, and syntax for the Formula Node.
Exercise 6-4  Formula Node Exercise VI

Objective: To use the Formula Node.

Complete the following steps to build a VI that uses the Formula Node to perform a complex mathematical operation and graphs the results.

Front Panel

1. Open a new VI and build the following front panel.

![Waveform Graph](image)

Block Diagram

2. Build the following block diagram.

![Block Diagram](image)

a. Place the Formula Node located on the **Functions » Structures** palette.

b. Create the x input terminal by right-clicking the left border and selecting **Add Input** from the shortcut menu.

c. Create the y and a output terminals by right-clicking the right border and selecting **Add Output** from the shortcut menu. You must create output terminals for temporary variables like a.

Note When you create an input or output terminal, you must use a variable name that exactly matches the one in the equation. Variable names are case sensitive.
d. Type the following equations in the Formula Node, where ** is the exponentiation operator. Refer to the LabVIEW Help for more information about syntax for the Formula Node.

\[ a = \tanh(x) + \cos(x); \]
\[ y = a^{3} + a; \]

3. Save the VI as Formula Node Exercise.vi.

4. Display the front panel and run the VI. The graph displays the plot of the equation \( y = f(x)^{3} + f(x) \), where \( f(x) = \tanh(x) + \cos(x) \).

   During each iteration, the VI divides the iteration terminal value by 15.0. The quotient is wired to the Formula Node, which calculates the function value. The VI plots the array as a graph.

5. Close the VI.

End of Exercise 6-4
Exercise 6-5  Expression Node Exercise VI

Objective:  To use the Expression Node.

Complete the following steps to build a VI that performs the same operation as the Formula Node Exercise VI you completed in Exercise 6-4.

Front Panel

1. Open a new VI and build the following front panel.

![Waveform Graph](image)

Block Diagram

2. Build the following block diagram.

![Block Diagram](image)

a. Place the Expression Node located on the **Functions»Numeric** palette.

b. Type the following equation in the Expression Node. Refer to the *LabVIEW Help* for more information Expression Node syntax.

\[(\tanh(x)+\cos(x))^{*3}+(\tanh(x)+\cos(x))\]

3. Save the VI as **Expression Node Exercise.vi**.

4. Run the VI. The graph displays the plot of the equation.

5. Close the VI.

End of Exercise 6-5
D. Avoiding Overusing Sequence Structures

To take advantage of the inherent parallelism in LabVIEW, avoid overusing Sequence structures. Sequence structures guarantee the order of execution and prohibit parallel operations. For example, asynchronous tasks that use I/O devices, such as PXI, GPIB, serial ports, and DAQ devices, can run concurrently with other operations if Sequence structures do not prevent them from doing so. Sequence structures also hide sections of the block diagram and interrupt the natural left-to-right flow of data.

When you need to control the execution order, consider establishing data dependency between the nodes. For example, you can use error I/O to control the execution order of I/O.

Also, do not use Sequence structures if you want to update an indicator from different frames of the Sequence structure. For example, a VI used in a test application might have a Status indicator that displays the name of the current test in progress. If each test is a subVI called from a different frame, you cannot update the indicator from each frame, as shown by the broken wire in the following block diagram.

Because all frames of a Sequence structure execute before any data pass out of the structure, only one frame can assign a value to the Status indicator.

Instead, use a Case structure and a While Loop, as shown in the following example.
Each case in the Case structure is equivalent to a Sequence structure frame. Each iteration of the While Loop executes the next case. The **Status** indicator displays the status of the VI for each case. The **Status** indicator is updated in the case before the one that calls the corresponding subVI because data pass out of the structure after each case executes.

Unlike a Sequence structure, a Case structure can pass data to end the While Loop during any case. For example, if an error occurs while running the first test, the Case structure can pass FALSE to the conditional terminal to end the loop. However, a Sequence structure must execute all its frames even if an error occurs.
Summary, Tips, and Tricks

- A Case structure has two or more subdiagrams, or cases. Only one subdiagram is visible at a time, and the structure executes only one case at a time. If the selector terminal is Boolean, the structure has a TRUE case and a FALSE case. If the selector terminal is an integer, string, or enumerated type value, the structure can have up to $2^{31} - 1$ cases.

- Inputs are available to all subdiagrams, but subdiagrams do not need to use each input. If at least one output tunnel is not defined, all output tunnels on the structure appear as white squares.

- A Sequence structure contains one or more subdiagrams, or frames, which execute in sequential order. A Sequence structure executes frame 0, then frame 1, then frame 2, until the last frame executes. The Sequence structure does not complete execution or return any data until the last frame executes.

- To pass data from one frame to any subsequent frame, use a sequence local terminal. Right-click the structure border and select Add Sequence Local from the shortcut menu to create a sequence local.

- Formula Nodes are useful for equations that have many variables or are otherwise complicated and for using existing text-based code. Each equation statement must terminate with a semicolon (;).

- Use Expression Nodes to calculate expressions, or equations, that contain a single variable.

- To take advantage of the inherent parallelism in LabVIEW, avoid overusing Sequence structures. When you need to control the execution order, consider establishing data dependency between the nodes.

- Do not use Sequence structures if you want to update an indicator from different frames of the Sequence structure. Instead, use a Case structure and a While Loop.