Exercise 3, Mon 21.11.

Problem1.

The factorial $n!$ is defined for a positive integer n as

$$n! \equiv n * (n-1) \ldots * 2 * 1.$$

The factorial of 0 is defined as 1. In this example, we will create a VI which performs this function, using a For Loop with a shift register.

1. Open a new blank VI and place a Case structure on the block diagram. The case structure can be found on the **Programming≫Structures** palette. We will use the Case structure to determine whether or not the user has entered a nonnegative integer.
2. Place a numeric control to the left of the Case structure and change the representation to I32. Label this control n. Use Figure 6.27 as a guide in the development of your VI.
3. Navigate to the **Programming≫Comparison** palette and find the Greater Than or Equal to 0? function. Connect the input of this function to n and connect the output to the selector terminal of the Case structure. The True frame of the case structure will execute when the input is a value greater than or equal to zero. The False frame will execute in all other cases.
4. Click either the increment or decrement button on the selector terminal to view the False case. Go to the **Programming≫Dialog & User Interface** palette and place a One Button Dialog function within the Case structure. Create a constant on the message input to this node with a message prompting the user to enter a nonnegative input.

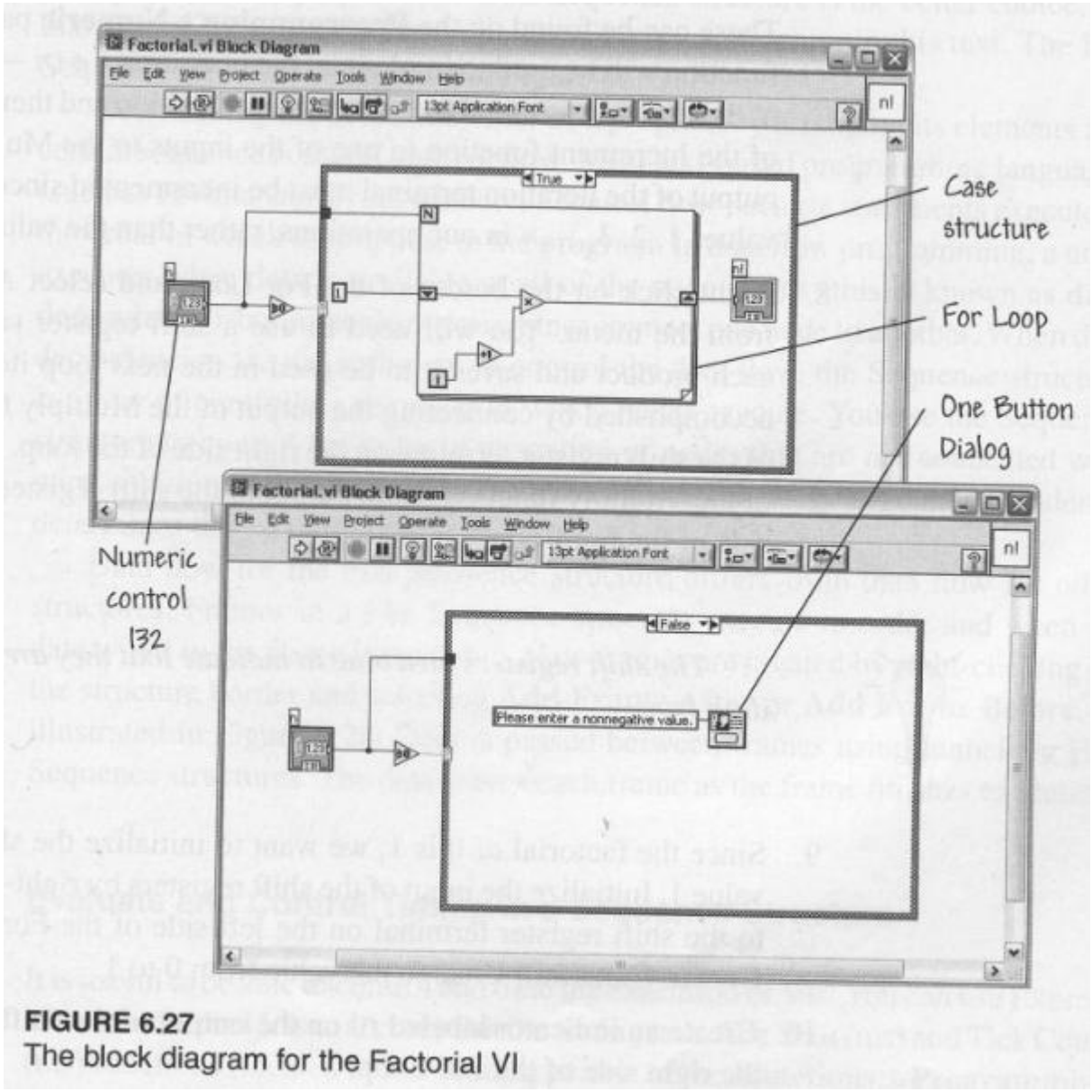


FIGURE 6.27
The block diagram for the Factorial VI

5. Now switch to the True case by clicking either the increment or decrement button on the selector terminal. Place a For Loop within the Case structure. The For Loop is used because we know that we need to perform the multiplication operation exactly n times.
6. Wire the input n to the count terminal of the For loop so that the loop will execute n times. When you do this, a tunnel will automatically appear on the border of the Case structure.
7. Place a Multiply function and an Increment function within the For Loop. These can be found on the **Programming≫Numeric** palette. The Multiply function will be used to implement the operation $n * (n - 1)$. Wire the output of the iteration terminal to the increment function and then connect the output of the Increment function to one of the inputs to the Multiply function. The output of the iteration terminal must be incremented since we want to use the values $1, 2, 3, \ldots n$ in our operations, rather than the values $0, 1, 2, \ldots n - 1$.
8. Right-click on the border of the For Loop and select **Add Shift Register** from the menu. You will need to use a shift register so that the VI stores each product and saves it to be used in the next loop iteration. This can be accomplished by connecting the output of the Multiply function to the input of the shift register terminal on the right side of the loop. Wire the other input of the Multiply function to the output of the shift register terminal on the left side of the For Loop.

The shift registers turn blue to indicate that they are handling an integer data type.

9. Since the factorial of 0 is 1, we want to initialize the shift register with the value 1. Initialize the input of the shift registers by right-clicking on the input to the shift register terminal on the left side of the For Loop and selecting **Create Constant**. Change the value from 0 to 1.
10. Create an indicator labeled n! on the output of the shift register terminal on the right side of the For Loop.

You have now created a VI which calculates factorials. Experiment with different input values. Try running the VI with highlight execution turned on so that you can see what values appear on the output of the iteration terminal and shift register with each loop iteration. Compare your VI to the factorial VI found on the **Mathematics≫Elementary & Special Functions≫Discrete Math** palette.

-Return the VI you created

Problem2.

In this example, we will use shift registers to compute the running average of a sequence of random numbers. Since the Random Number function in LabVIEW provides random numbers from 0 to 1, we expect that the average will be 0.5. How many random numbers will it take to obtain an average near 0.5?

Figure 6.18 shows a VI that computes the running average of several random numbers. The length of the random number sequence is input via the slide control, as depicted on the front panel in Figure 6.18. Using the block diagram in the figure as a model, develop your own VI to compute the running average of a sequence of random numbers. The formula that is coded to compute the average is

$$Ave_i = \frac{i}{i+1} Ave_{i-1} + \frac{1}{i+1} RN_i,$$

where $i = 0, 1, \cdots, N-1$, Ave_i is the computed average at the ith iteration, and RN_i is the current random number from the random number function. If you aren't familiar with computing a running average, just concentrate on the programming aspects of the VI shown in Figure 6.18 and do not worry about the formula. The main point is to understand how to use the shift registers in conjunction with For Loops and While Loops!

In this example, the shift register is used to pass the value of the variable Ave_{i-1} from one iteration to the next. You should notice as you run the code that for small values of N (e.g., $N = 3$) the average is generally not close to the expected value of $Ave_N = 0.5$; however, as you increase N, the average gets closer and closer to the expected value. Try it out!

-Return your VI

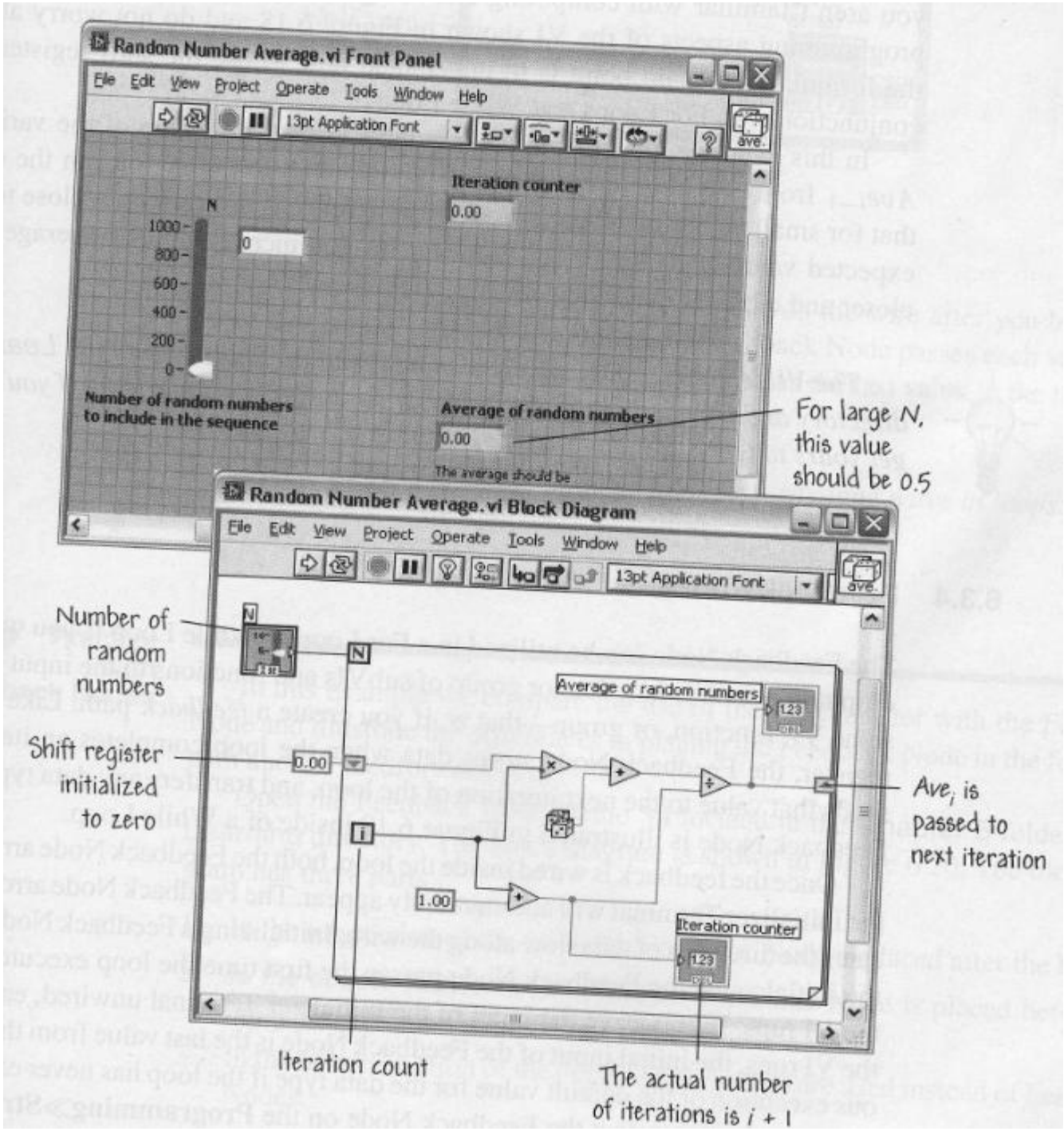
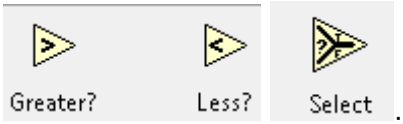


FIGURE 6.18
Computing a running average of a sequence of random numbers

Problem3.

Use a For Loop to generate 100 random numbers. Determine the most current maximum and minimum number as the random numbers are being generated. Display the running maximum and minimum values as well as the current random number on the front panel.

Useful functions from the Comparison palette can be: 

.

To slow down the For loop you should add, for example, the Time Delay Express VI in the For loop (you can find it from Programming >> Timing >> Time Delay. Delay of 0.1 s should be suitable to visualize how the values update.

-Return your VI

Problem4.

Create a time trial program to test the execution times of the Formula Node and the native LabView Math Functions. Use a For loop to run your function N times (N=1e6 should be sufficient, depending on the computer). Make a case structure so that the user can select between Formula Node and native LabView code. Sequence structure is needed to get the Tick counts before and after the code executes. An example for timing your VIs or functions can be found from Help >> Find Examples >> Fundamentals >> Loops and structures >> Timing Template (data dep).vi (when saving your VI don't overwrite the example; use Save As). Try the timing on the following function:

Y=2*X+exp(-X)+1;

where X is the input and Y is the output.

Is there any difference between the execution times of Formula Node and the native LabVIEW code? Try also to take the indicator, showing the result of the calculation, outside the For loop. Does this make the execution faster?

-Return your VI