



LabVIEW Programming

Intro, Examples, Palettes, Structures,
Concurrency, Timing, Troubleshooting

What is LabVIEW?

- LabVIEW stands for Laboratory Virtual Instrument Engineering Workbench
- LV is a graphical programming language. This means that it uses graphical icons (instead of text) to program.
- LV is created by National Instruments

Why use LabVIEW?

- LV is specifically designed for scientists and engineers (designed for ease in taking measurements, analyzing data, and presenting results)
- NI has written free drivers to control popular instruments. This makes laboratory automation much easier and less time consuming for the user.
- NI also has nice equipment that is almost guaranteed to work with LV, so many users use LV for this reason.

Why use LabVIEW?

- As mentioned before, LV makes it really easy to present data (in particular, charts).
- It also has many knobs and buttons to choose from to make Graphical User Interfaces (GUI) easily.
- This makes creating Virtual Instruments very easy compared to other languages. (*Note: the programs are saved as .vi files*)

Why use LabVIEW?

- Ease of interfacing over different communication links such as NI's General Purpose Interface Bus (GPIB) cables, USB, RS-232, ethernet, etc. Note that GPIB used to be the gold standard for communicating with instruments because of their reliability, but USB has taken over since it's much cheaper.
- Parallel processing occurs naturally with how LabVIEW code is written

LabVIEW's Biggest Disadvantage

LV's biggest disadvantage (in my opinion) is that you have to connect all of the items with wires, which means you spend a lot of time cleaning up your wires and making things look pretty instead of actually programming.

This can be really tedious, especially when you're dealing with strings. Text-based code is far more superior when dealing with strings!

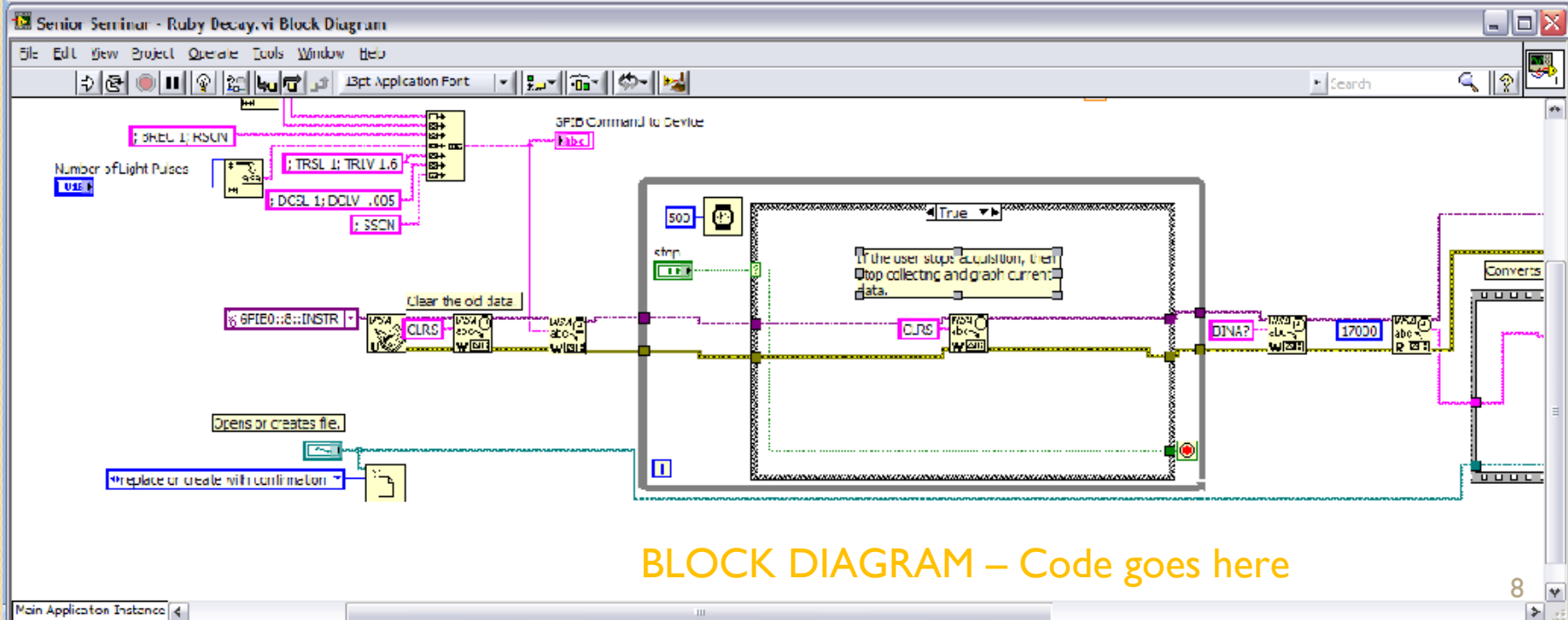
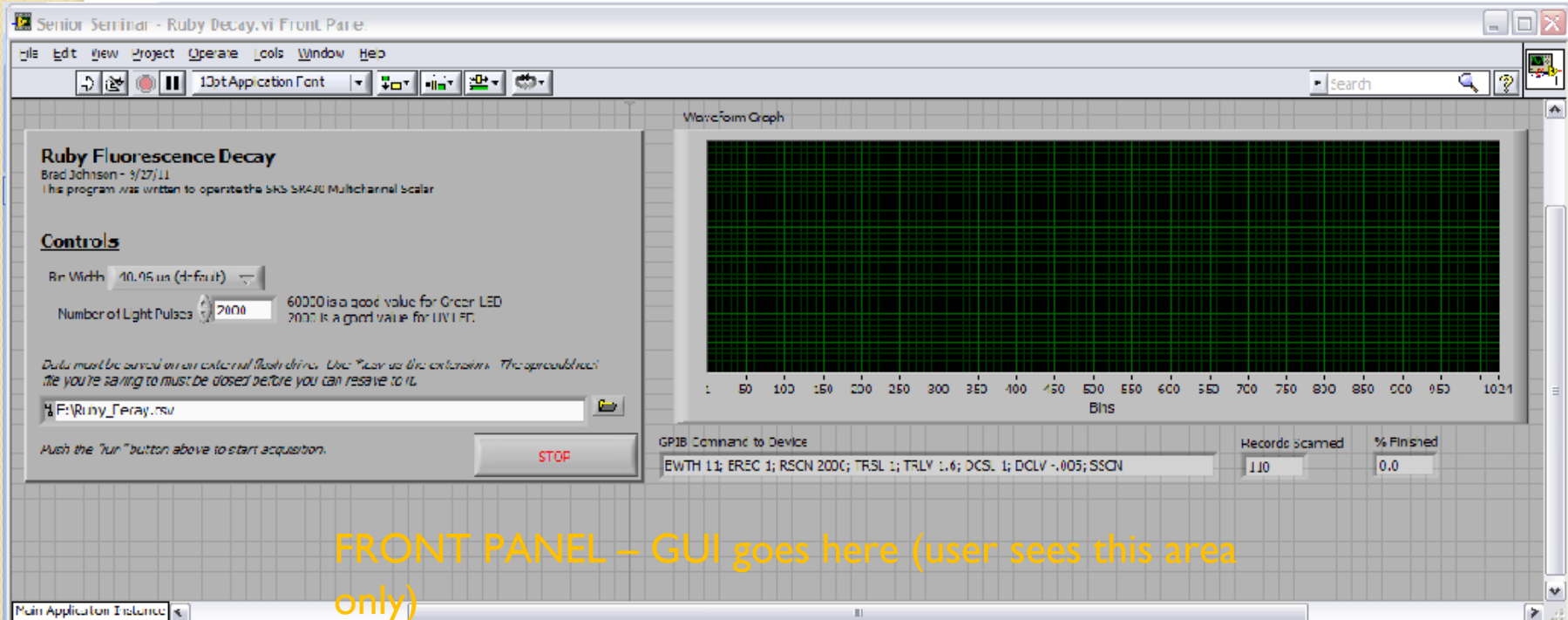
However, I find reading other people's code or reading old code that I've written is much easier and faster than text-based languages. So this disadvantage is an advantage for reading code. Some text-based code is so massive and overwhelming that it becomes obsolete with time and upgrades.



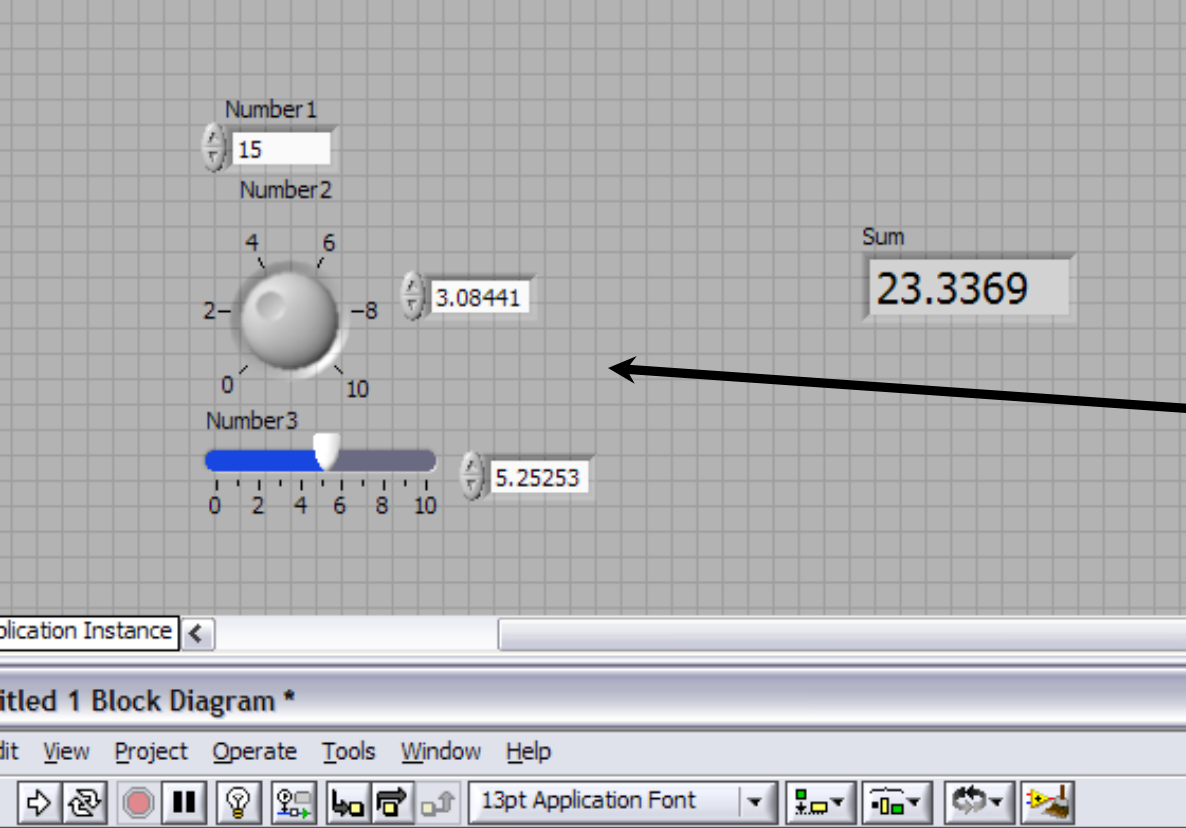
LabVIEW Example

This example is a LabVIEW program that controls a multi-channel scalar that will be used for Senior Seminar.

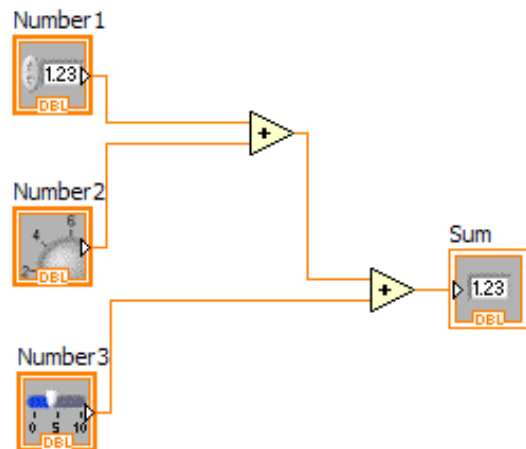
Students will excite a ruby crystal and measure the fluorescence decay.



Easier Program

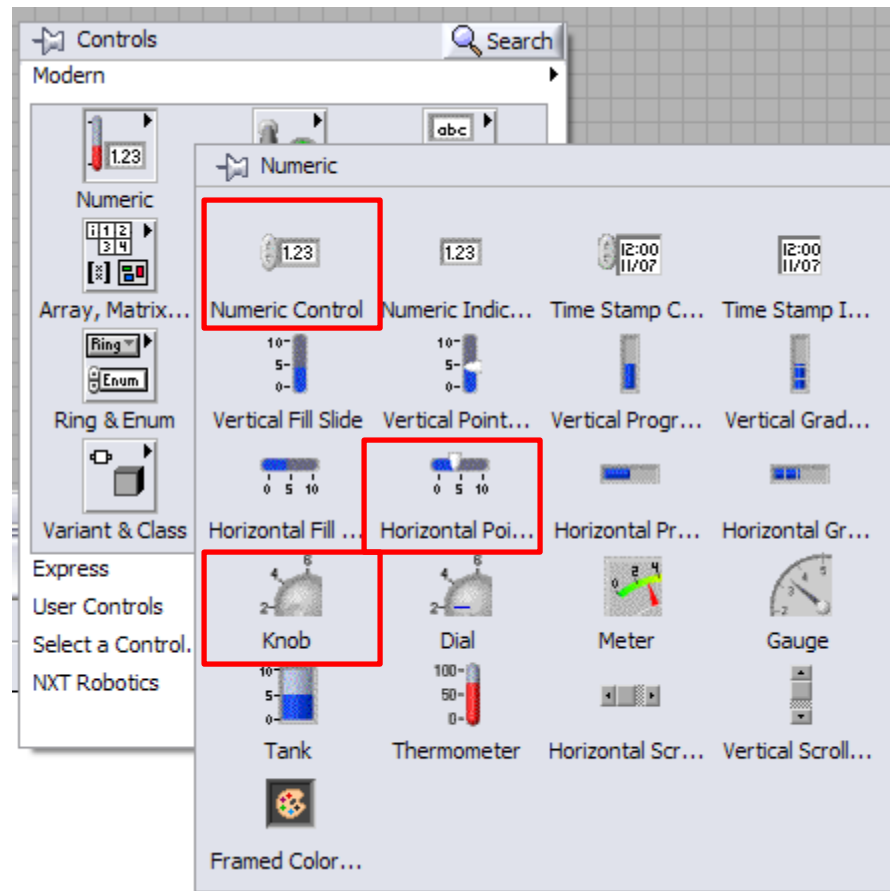


Notice that you can make several different kinds of numbers

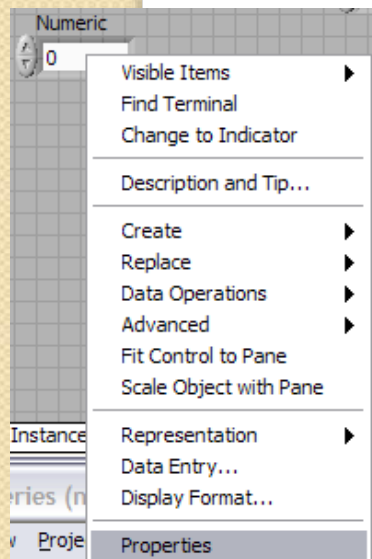


Adds the 3 numbers with
“Sum” being the result

The numbers were taken from the Numeric Palette.



LabVIEW has lots of palettes to choose from. You should scroll through all of the different palettes to see what is available.



If you right-click on the number and click on properties, you can change the characteristics of the number.

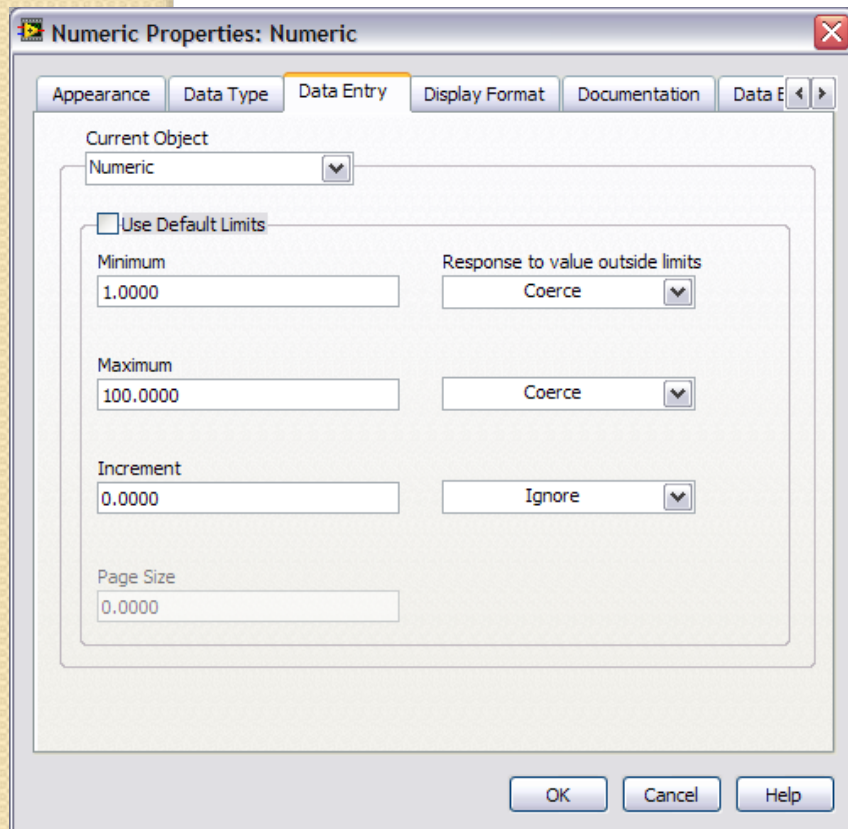
Some of these characteristics include . . .

- the maximum and minimum values of the number

- the increment of the arrows beside the number

- the type of number (integer, double, etc.)

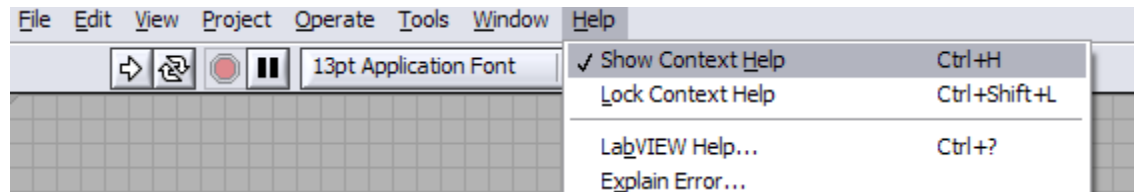
- the display format of the number (scientific notation, SI notation, regular) (number of decimal places).



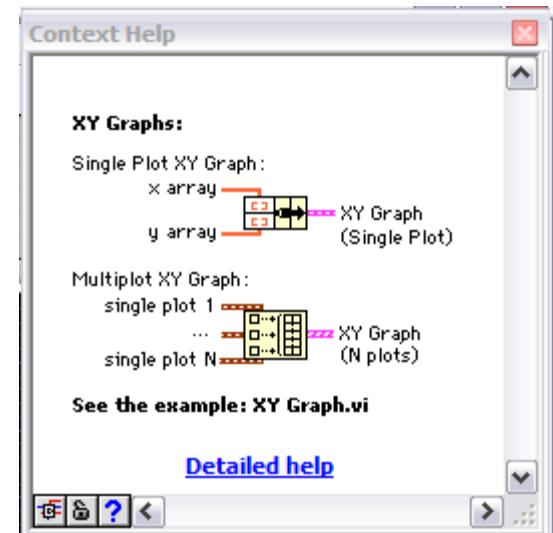
You can also create it so that if the maximum number is 100 and the user inputs 430, then the number will coerce to 100. This makes it so that the user sees the value change after s/he types in a number that is too big. This makes the code more “user friendly” since users will see that they can’t input certain values. It is also important to make your code “foolproof” so that the user can’t crash the program or put in illegal values.

LabVIEW Help and Examples

Before I go further, I want to mention that you can get help with each VI by choosing Help > Show Context Help (ctrl+h).



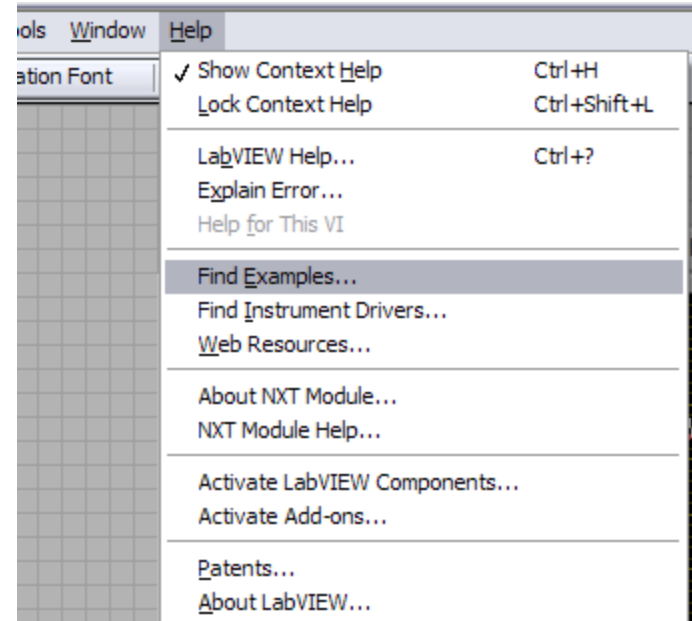
This will explain some VI and will usually let you find out more information about the VI by going to Detailed Help.



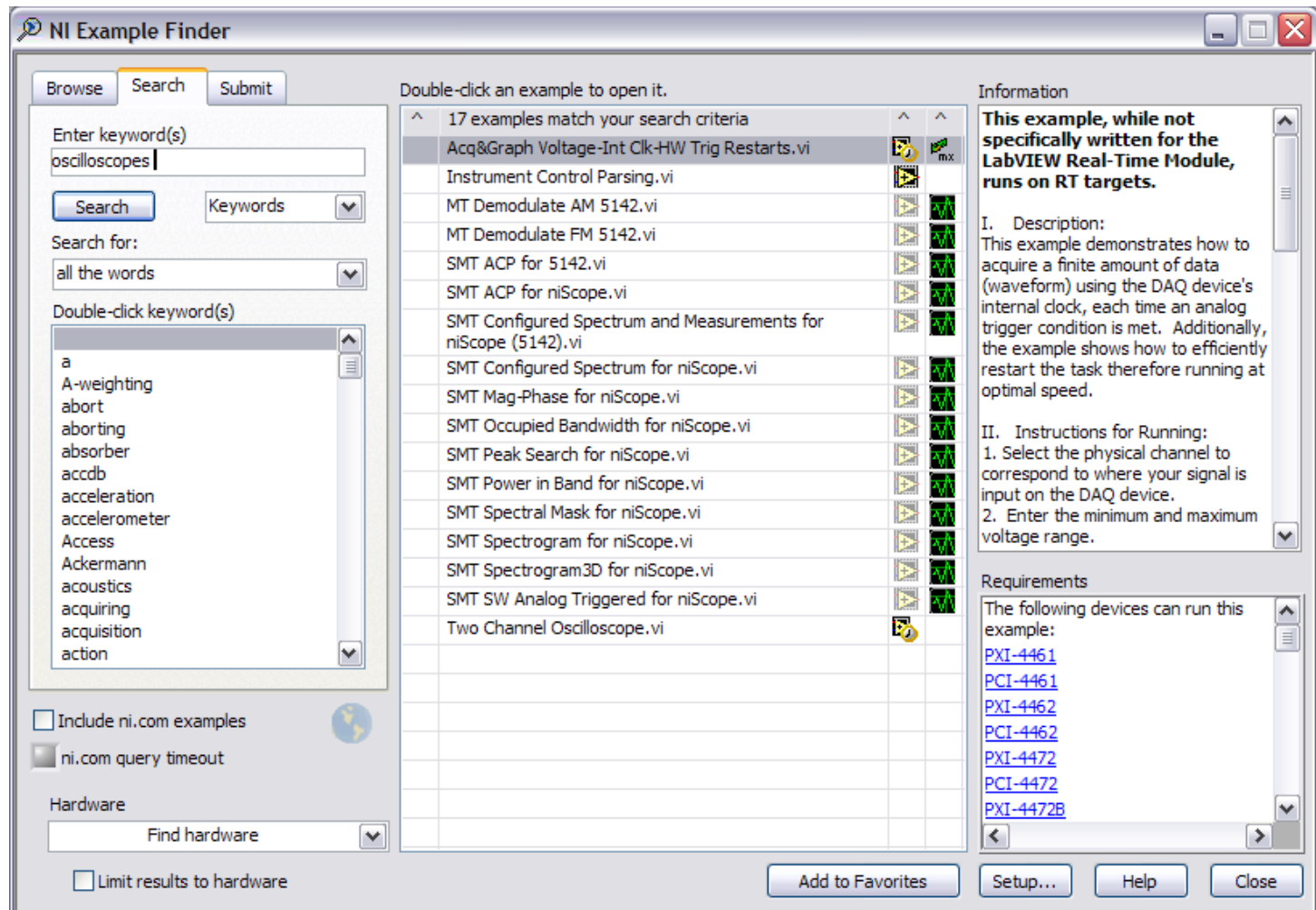
LabVIEW Help and Examples

You can almost find an example for anything you want to do by going to Help > Find Examples.

I use this resource more than any others when I'm trying to figure out how to do something.



LabVIEW Help and Examples

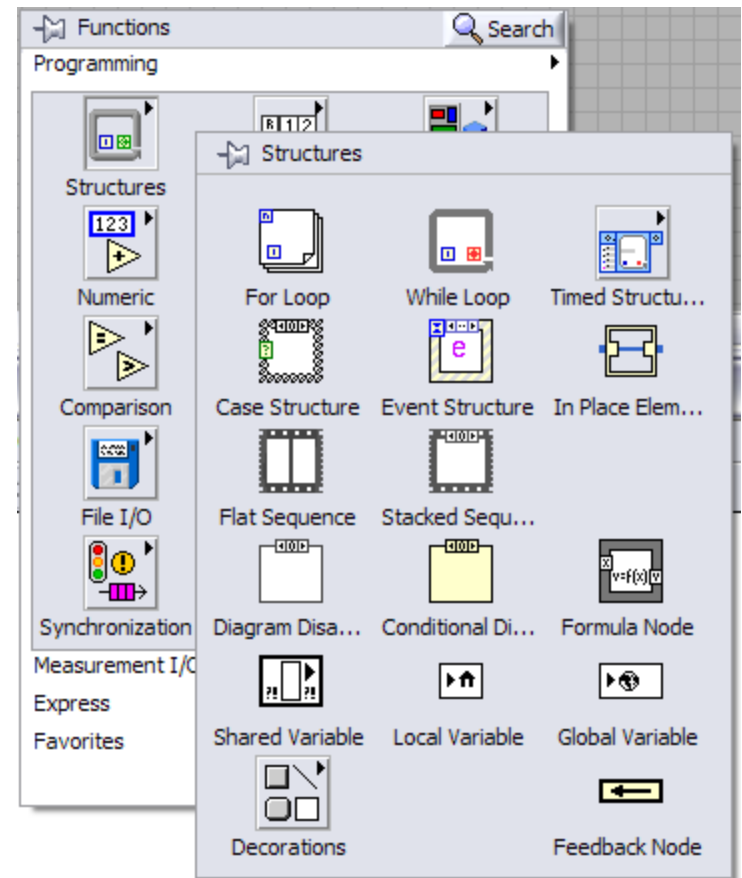


Structures

(Ch.6) from LV for Everyone

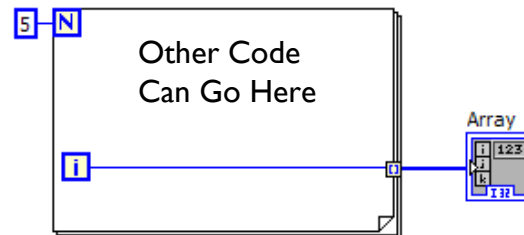
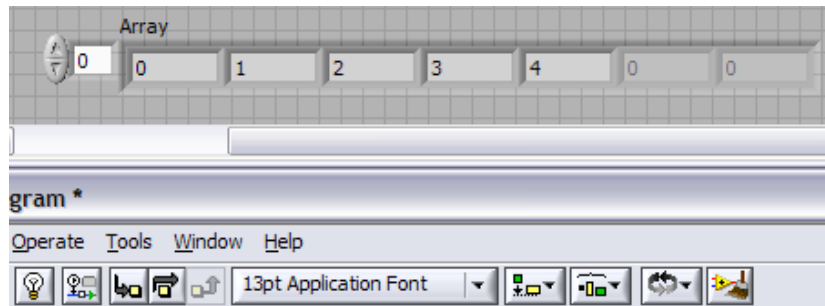
Structures are used often in LabVIEW and each structure is completely different.

We'll use For loop, While loop, Case Structure, Event Structure, Flat Sequence and Stacked Sequence, Diagram Disable, and Formula Node.



For Loop

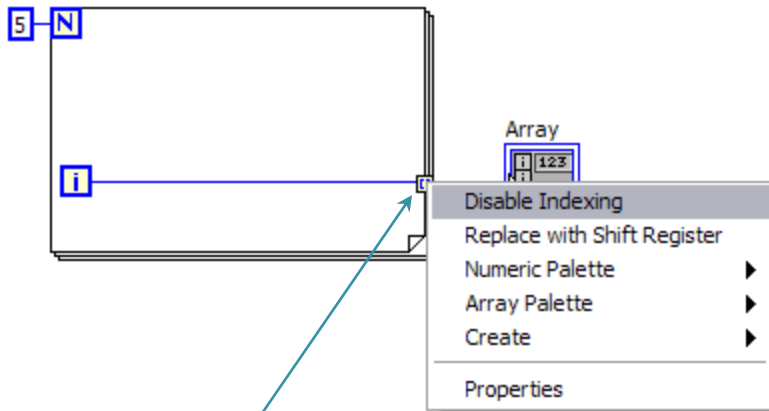
As you probably know, For loops are used when you know the exact number of iterations you want to perform.



In this example, the 5 represents the number of iterations you want to perform and the box represents the For loop. *i* is the value of the current iteration (starts at 0). This program made an array of 5 numbers (0 through 4). For loops are a good way to make arrays. Notice that the numbers after 4 are grayed out, meaning they don't really exist.

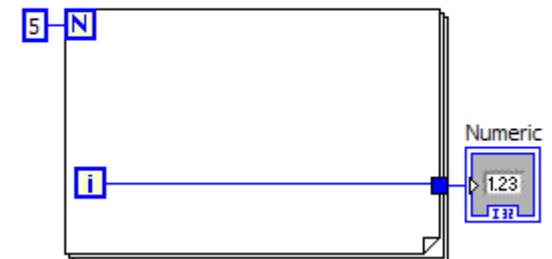
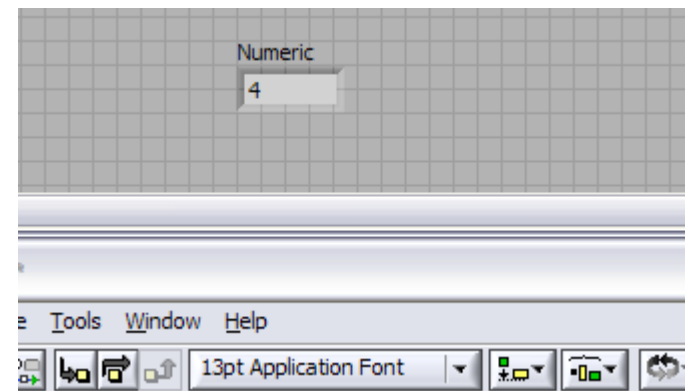
For Loop

- You can disable indexing if you only want the value of the last iteration.



This is achieved by right-clicking on the little square and selecting “Disable Indexing.”

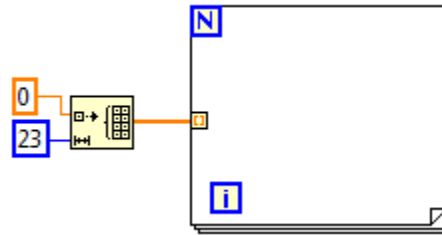
Notice that by disabling indexing, we only take the value from the last iteration, so we no longer have an array. New value is 4.



For Loop

A neat trick with For loops is if you want the For loop to have as many iterations as the number of elements in some array, then you simply tie an array into the left side of the for loop with indexing enabled.

If you do this, then you shouldn't wire a number into the N of the for loop. **Important: If you do wire a number into N, then the for loop will iterate the smaller number between the number and array size to iterate.**



The first icon creates an array of 23 zeros. We will work with arrays later on in the course. Since the array has 23 elements, the for loop will iterate 23 times.


Note: If you were to wire a 100 into the “N”, then the for loop would iterate the smaller of the two numbers – so 23 times.

For Loop

Note: If you input a 0 or negative number into the N terminal of a For loop, the For loop will not implement any code.






CLAD Exam Question

Which combination of words correctly completes the following statement? The _____ indicates the total number of times the loop will execute and the _____ returns the number of times the loop has executed minus one.

- a. count terminal,  ; conditional terminal, 
- b. conditional terminal,  ; iteration terminal, 
- c. count terminal,  ; iteration terminal 
- d. conditional terminal,  ; count terminal, 

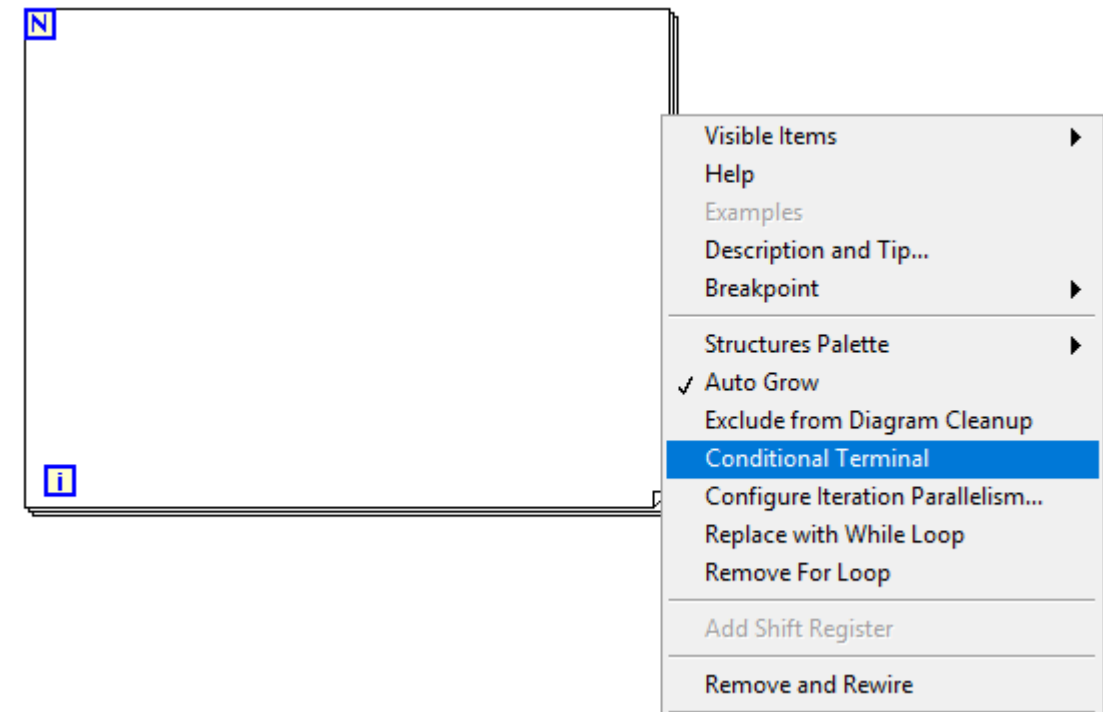
CLAD Exam Question

Which combination of words correctly completes the following statement? The _____ indicates the total number of times the loop will execute and the _____ returns the number of times the loop has executed minus one.

- a. count terminal,  ; conditional terminal, 
- b. conditional terminal,  ; iteration terminal, 
- c.  count terminal,  iteration terminal
- d. conditional terminal,  ; count terminal, 

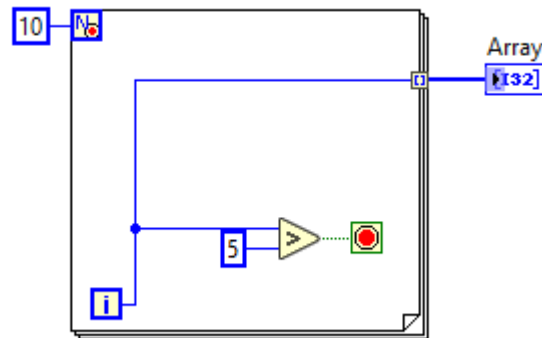
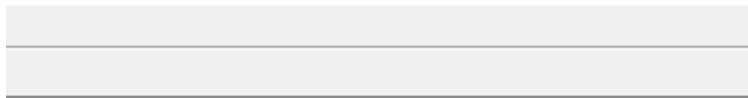
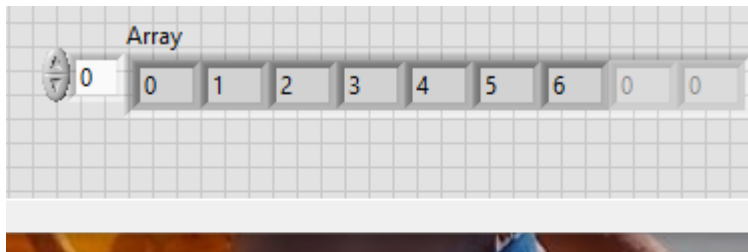
Conditional For Loops

You can create a Conditional For Loop by right-clicking on the bottom right corner and selecting “Conditional Terminal.”



Conditional For Loops

The following shows an example of a Conditional For Loop



Notice that the For loop looks slightly different with the N and it now has a stop sign.

Conditional For Loops allow a For loop to break out if some condition is met. In this example, if the iteration (i) is greater than 5, then it stops the For loop. Notice that the For loop iterates 7 times and not the 10 times that is fed into the Number of Iterations (N).

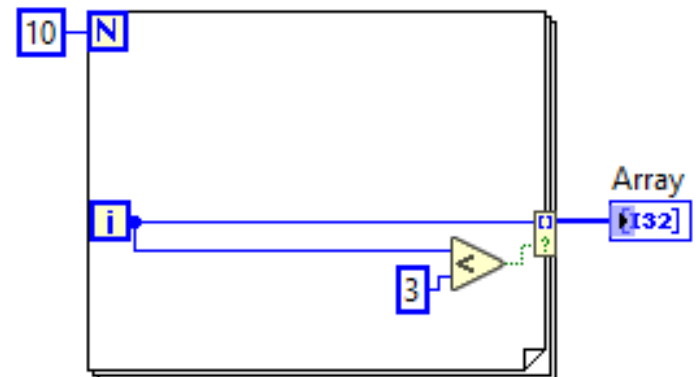
Conditional Terminals with Structures

A new feature to LabVIEW is to allow conditional terminals with structures. Notice the Question mark at the terminal. This means that only items that are less than 3 will be allowed to pass through.

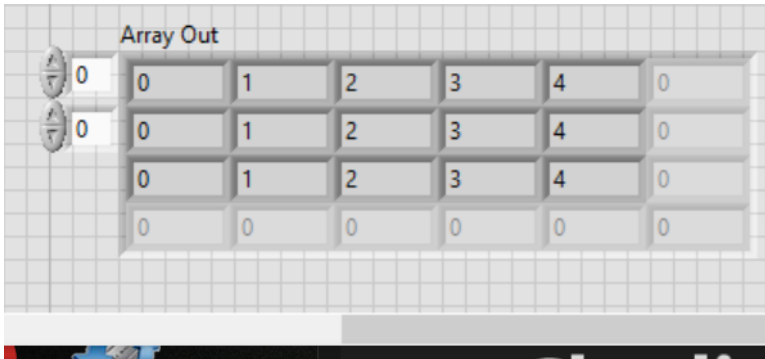
This feature could be really valuable.

To make a conditional terminal, right-click on the terminal and choose “Conditional.”

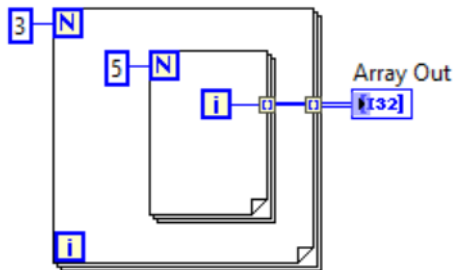
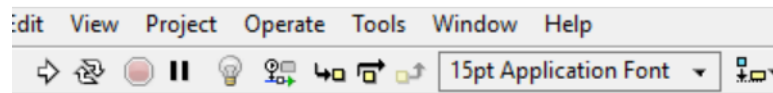
These conditional terminals work with other structures also such as While loops.



Nested Tunnel Mode



Lab Exam - Tunnel mode - concatenate.vi Block Diagram *

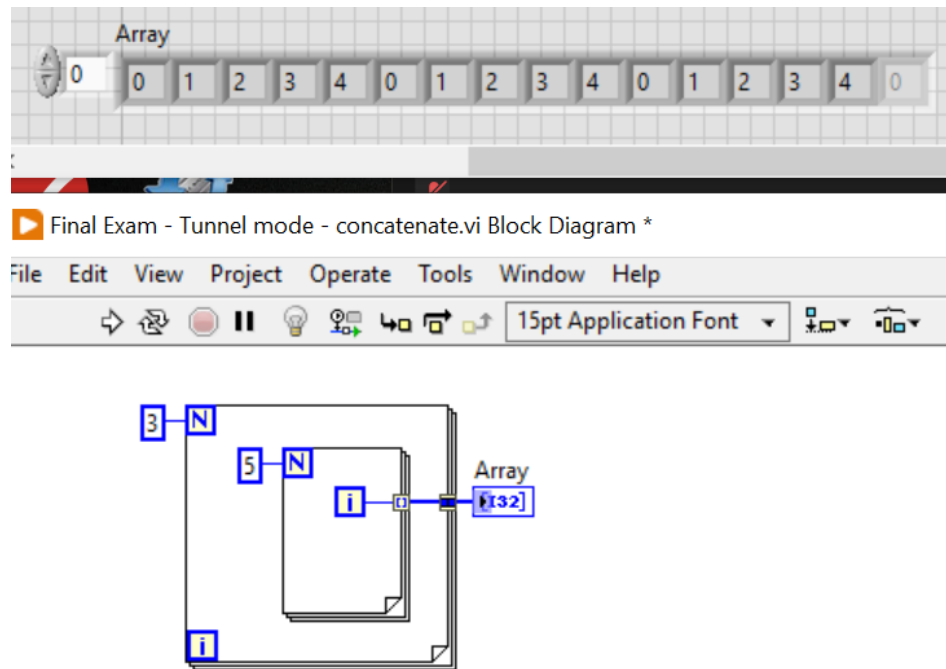


If I have 2 nested For loops, the inner loop creates {0,1,2,3,4} and the outer loop repeats this 3 times creating the 2D array of...

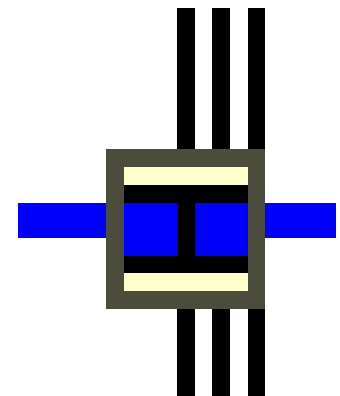
{0,1,2,3,4},
{0,1,2,3,4},
{0,1,2,3,4}}

Concatenating Tunnel Mode

If I wanted to have the 3 rows concatenated, then I can right-click on the tunnel and choose “concatenating.” This produces a 1D array of {0|2340|2340|234} instead of 2D.

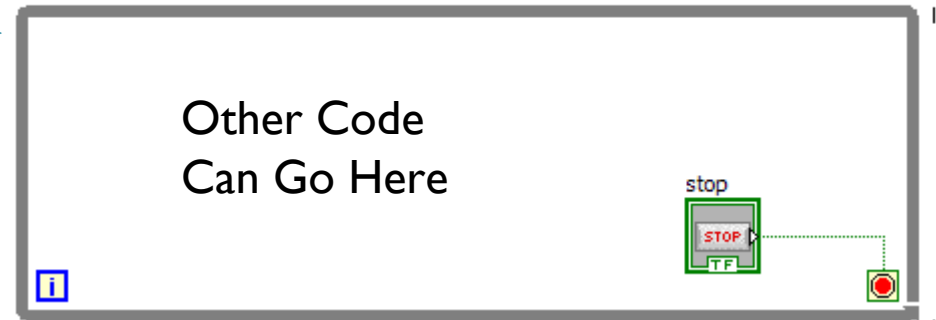


Zoomed in...



While Loop

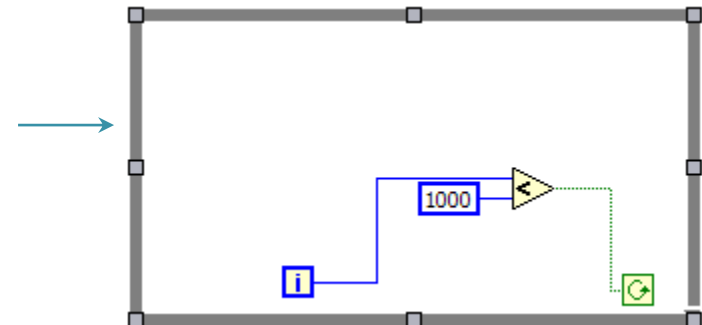
A while loop continuously implements the code inside of the loop until some kind of Boolean conditional stops the loop and it always implements **at least one** iteration (like a do-while in C). The Boolean condition in this code is the stop button. So the program will keep running until a user hits the stop button.



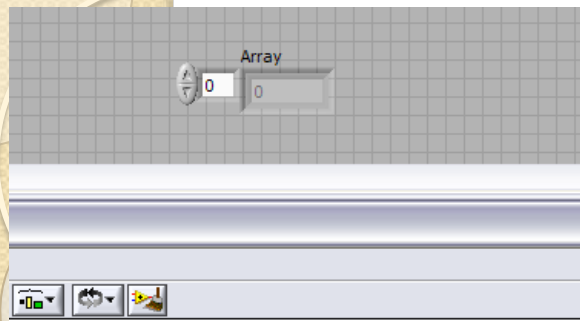
By the way, green lines represent Boolean

The Boolean condition in this while loop is a comparison between the current iteration and 1000. The code runs while $i < 1000$. Stops when $i \geq 1000$.

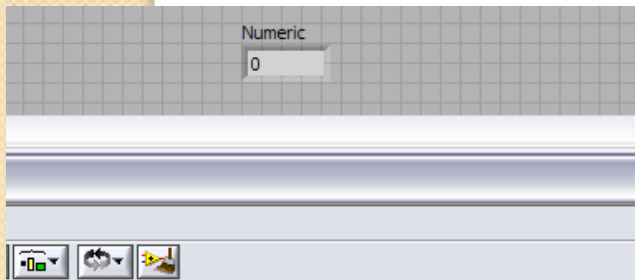
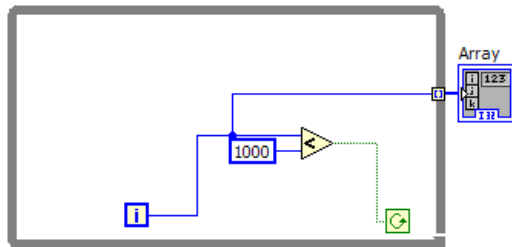
Blue lines represent integers, Orange lines represent floats or doubles, Pink lines represent strings.



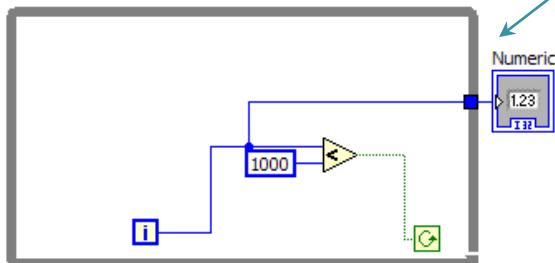
While Loop



You can also create an array of values by Enabling Indexing on while loops just like you did for the For Loop example



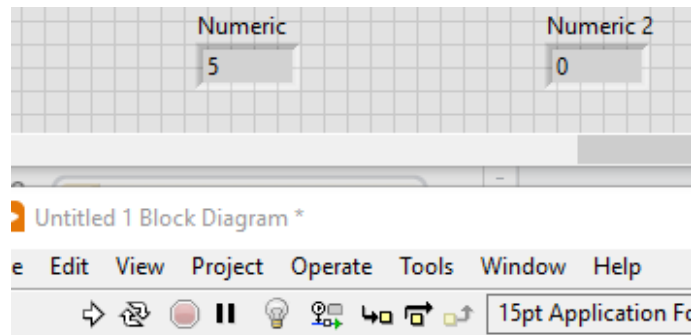
Or you can disable indexing, just like the For Loop. While loops default to indexing disabled and For loops default to indexing enabled.



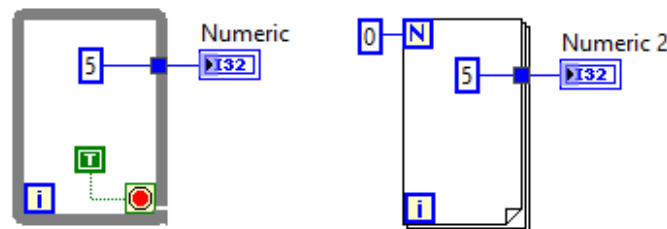
While Loop

While loops in LabVIEW act like Do-While loops in C because it always **executes for at least one iteration.**

Many programs in LabVIEW use While loops as the main loop in the program because While loops keep the program running until some condition is met (such as pushing the stop button). In contrast, For loops execute for a set number of iterations and then the program is finished.

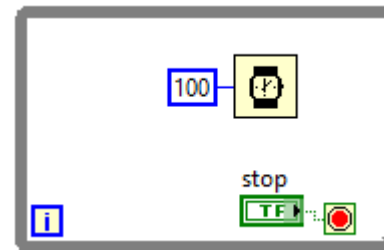
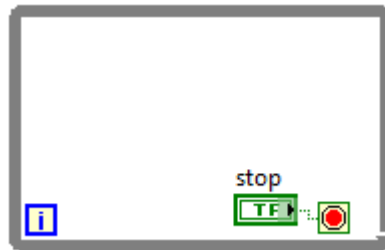


Notice that the While loop runs at least once and displays a 5, but the For loop doesn't run at all and displays a 0.



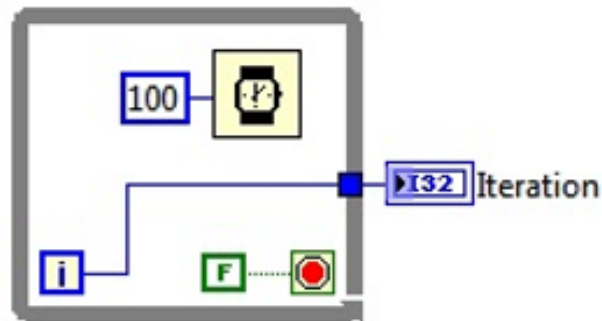
Adding a Wait (ms) Delay to While Loops

Often times, data only needs to get polled every so often. For example, the first program is just waiting for the user to hit the stop button. The way this is implemented, the while loop will run as fast as possible causing our CPU to bog down when it really doesn't need to. A fix to this is to add a Wait (ms) delay. This way the while loop only polls every 100 ms and allows our CPU to do other tasks. Note that the code inside of a while loop implements at the beginning of the 100 ms and then it waits for the 100 ms to be up before switching to the next iteration.



CLAD Question

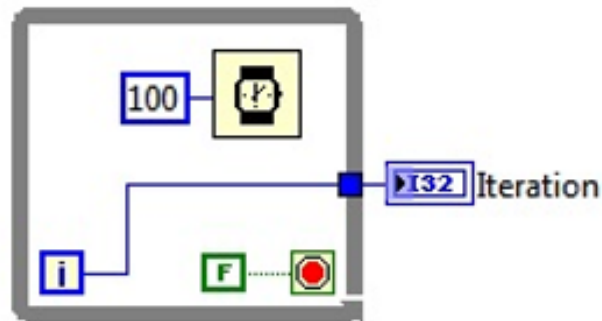
Q6: Which of the following statements is TRUE about the following code segment?



- A** The loop will execute once and the indicator Iteration will have a value of one
- B** The loop will execute once and the indicator Iteration will have a value of zero
- C** The loop will execute continuously and the program will have to be aborted
- D** The loop will not execute and the indicator Iteration will have a value of zero

CLAD Question

Q6: Which of the following statements is TRUE about the following code segment?

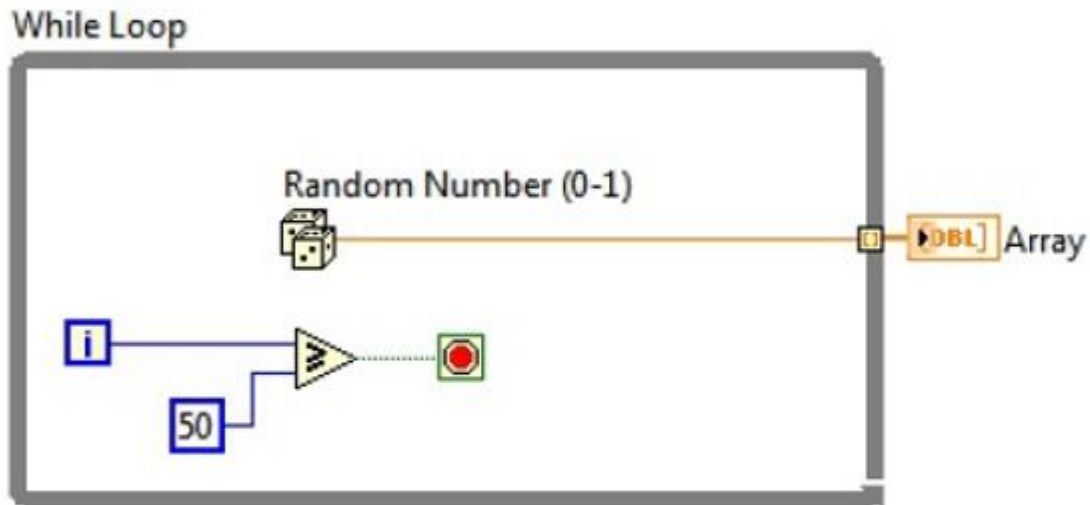


- A** The loop will execute once and the indicator Iteration will have a value of one
- B** The loop will execute once and the indicator Iteration will have a value of zero
- C** The loop will execute continuously and the program will have to be aborted
- D** The loop will not execute and the indicator Iteration will have a value of zero

CLAD Question

Q5: Which of the following statements is TRUE regarding the execution of the following code?

The loop will iterate:



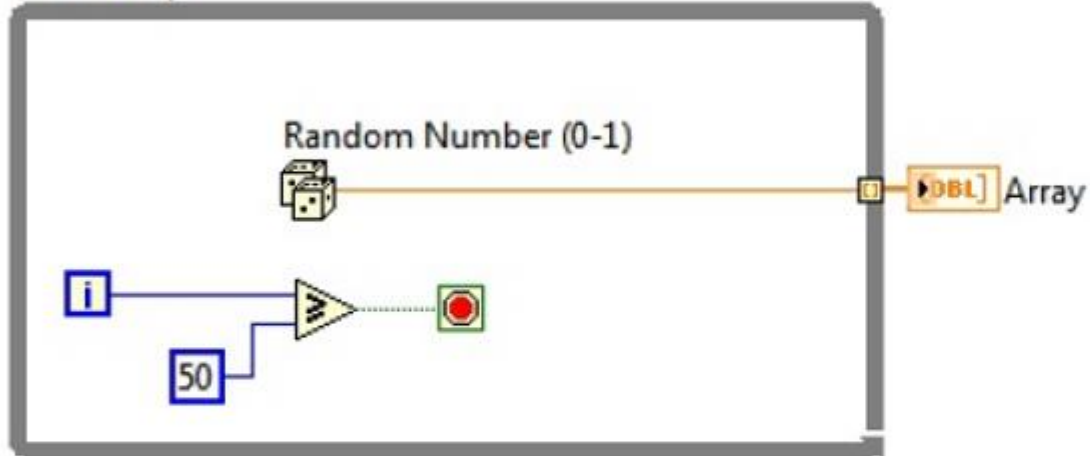
- A** 51 times
- B** 50 times
- C** 49 times
- D** a random number of times

CLAD Question

Q5: Which of the following statements is TRUE regarding the execution of the following code?

The loop will iterate:

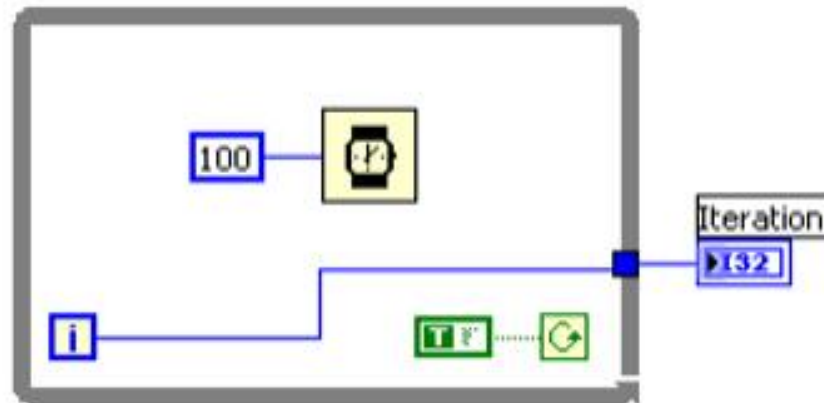
While Loop



- ☒ A 51 times
- ☐ B 50 times
- ☐ C 49 times
- ☐ D a random number of times

CLAD Question

Which of the following statements is true about the following block diagram?



- a. The loop will execute once and the iteration terminal, **i**, will output a value of one
- b. The loop will execute once and the iteration terminal, **i**, will output a value of zero
- ☒ c. The loop will execute infinitely and the program will have to be aborted
- d. The loop will not execute and the iteration terminal, **i**, will return a null value

CLAD Question

The Wait function can be added to While Loops:

- a. To free up available memory.
- b. To allocate memory used by the CPU.
- c. To allow the processor time to complete other tasks.
- d. To reserve which processor the code is running on.

CLAD Question

The Wait function can be added to While Loops:

- a. To free up available memory.
- b. To allocate memory used by the CPU.
- ☒ c. To allow the processor time to complete other tasks.
- d. To reserve which processor the code is running on.

Shift Registers and Formula Nodes

Number of Months: 4

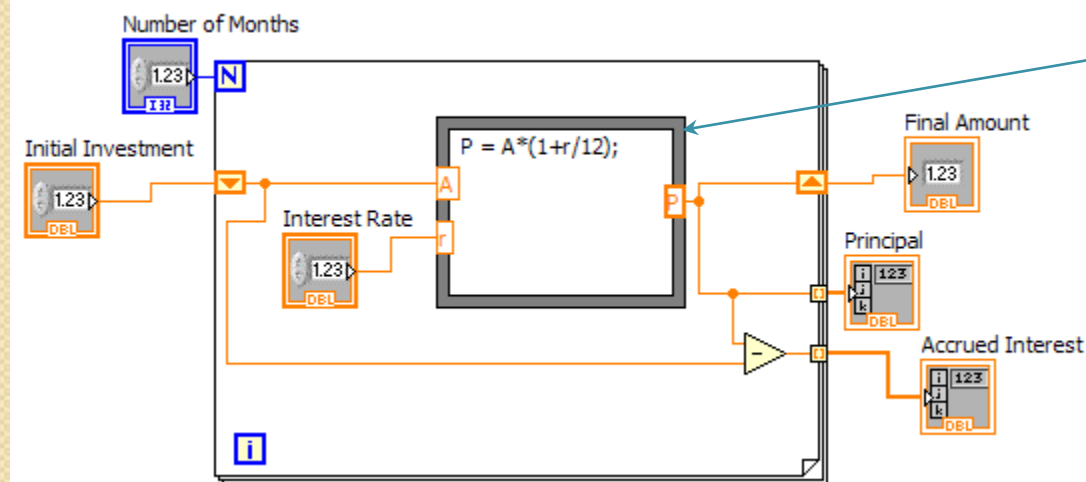
Initial Investment: 1000

Interest Rate: 0.13

Principal: 0, 1010.83, 1021.78, 1032.85, 1044.04, 0

Accrued Interest: 0, 10.8333, 10.9507, 11.0693, 11.1892, 0

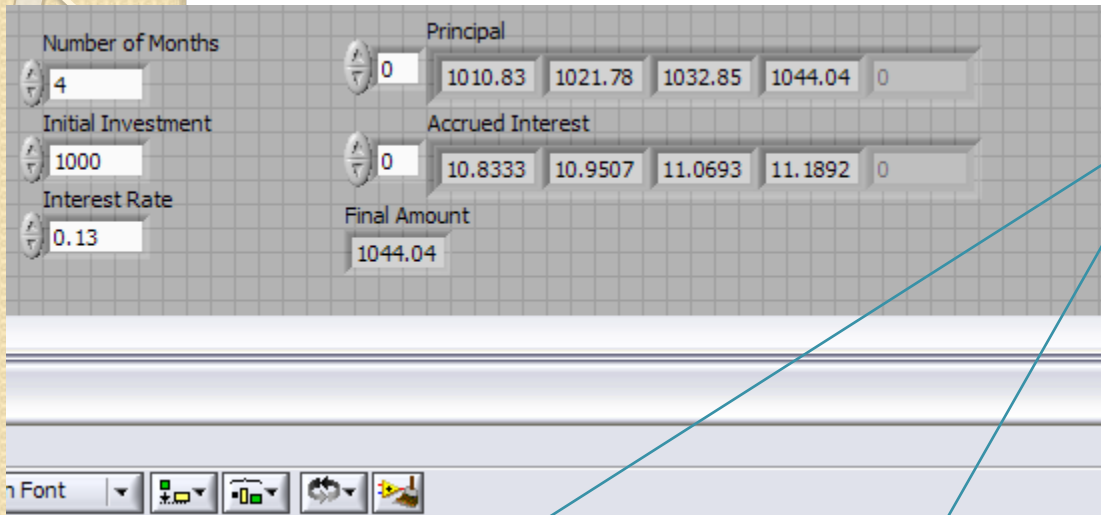
Final Amount: 1044.04



The **formula node** allows us to easily create a formula so that we don't have a million icons. I find that these make the code easier to read as well, especially for formulas. It uses C syntax.

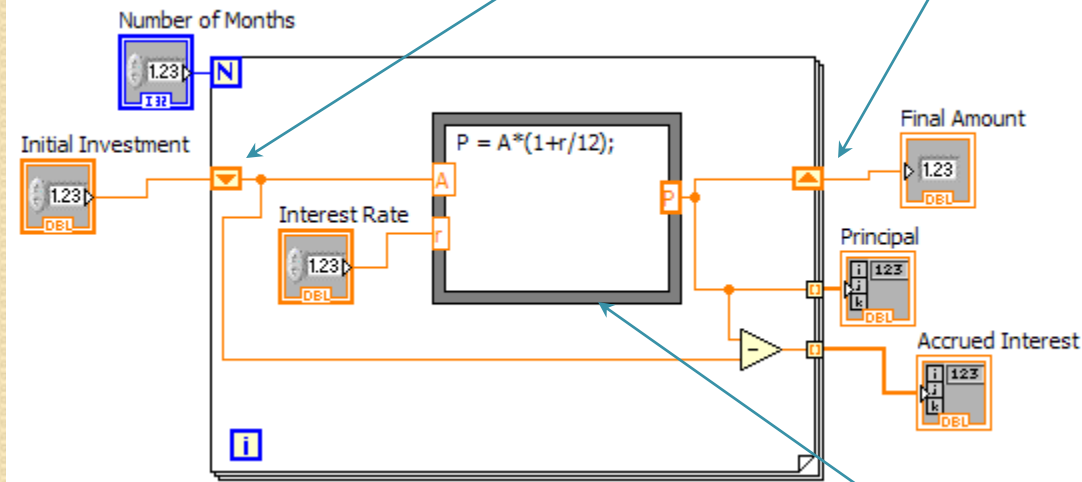
The letters on the left-hand side of the formula node are inputs and the letters on the right-hand side are outputs.

Shift Registers (Ch.6, p.195 LVFE) and Formula Nodes



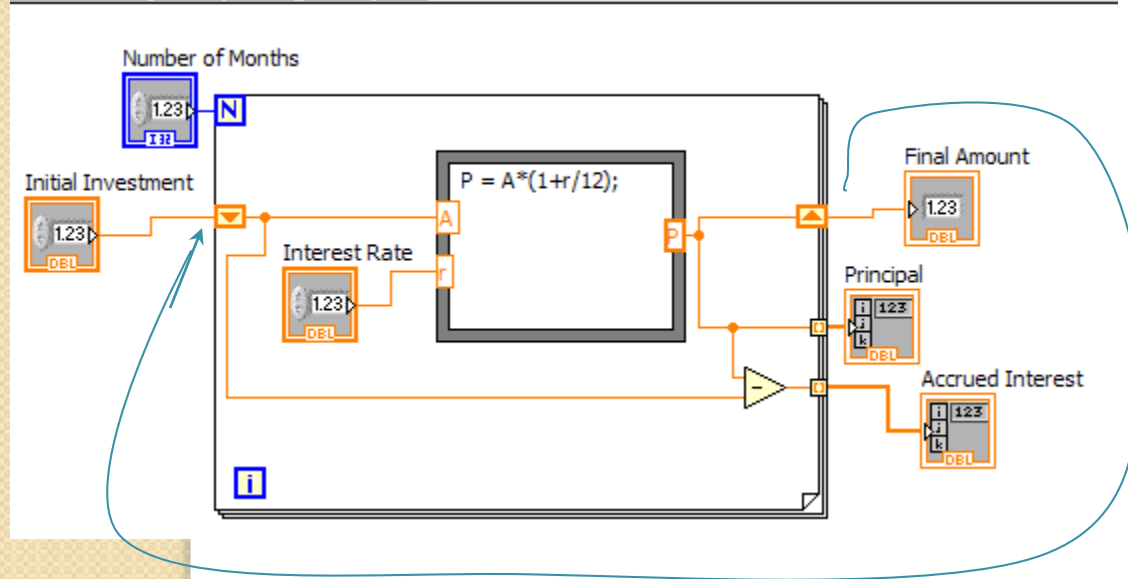
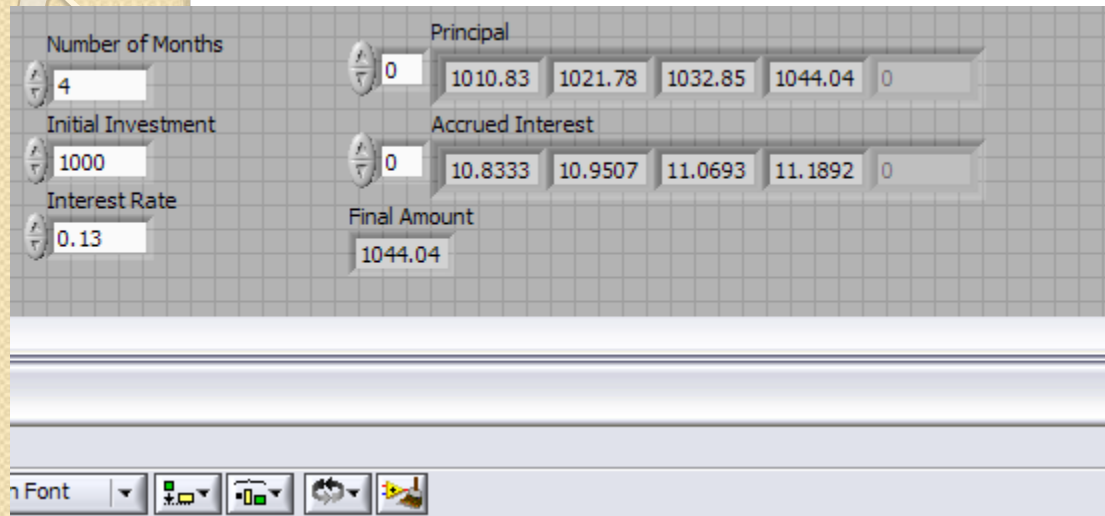
A shift registers saves the value for the current iteration and uses it for the next iteration.

In this example, we are computing the principal for an investment of \$1000 with an interest rate of 13% compounded monthly for 4 months.



As you can see, the accrued interest increases a little bit each month (~12 cents) since you add the accrued interest to the principal each month. So you make about \$11 + additional \$0.12 each month.

Shift Registers and Formula Nodes



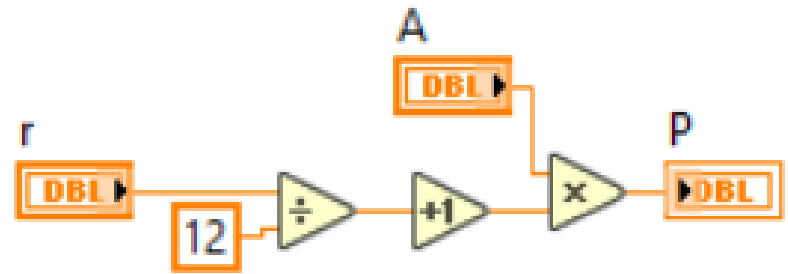
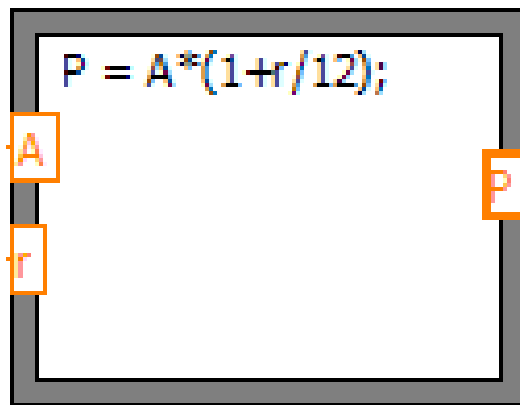
So the shift registers allow us to take the new principal (with the accrued interest) and re-apply it to the formula node for the next iteration. For this example, each iteration represents a month.

We initialized the shift register with an initial investment of \$1000. Then after 1 month, we'll have \$1010.83, then we apply this amount to the next month. After the 2nd month, we'll have \$1021.78 and we'll apply this to the following month and so on.

Shift registers make this possible and are used often.

Formula Nodes

Formula Nodes make it easier to read math equations and typically take less space. It uses C type code. For example, x^2 doesn't exist. Must use `pow(x,2)`.



These two equations create the same output, but the Formula Node is much easier to read.

CLAD Question

Formula nodes accept which of the following operations?

- a. Basic programming language instructions *Input* and *Print*
- b. Embedding of SubVIs within the Formula Node
- c. Pre and post increment (++) and decrement (--) as in the C language
- d. The use of nested Formula Node structures

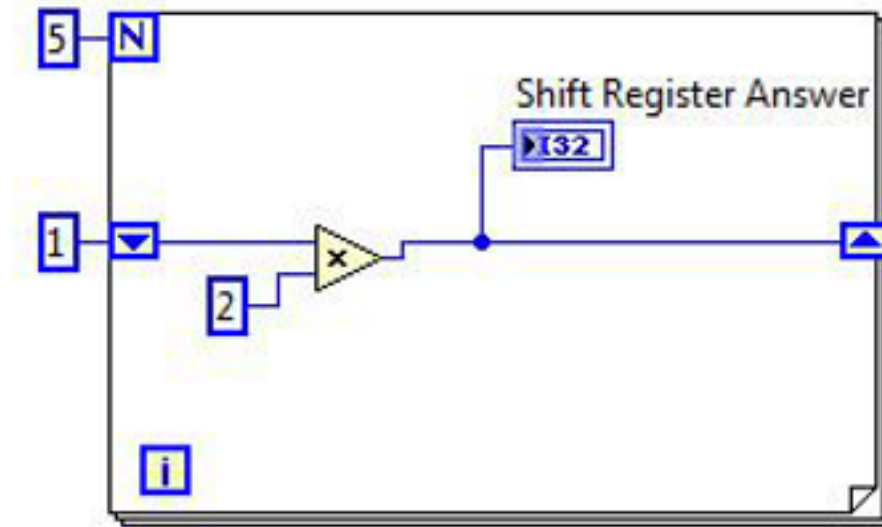
CLAD Question

Formula nodes accept which of the following operations?

- a. Basic programming language instructions *Input* and *Print*
- b. Embedding of SubVIs within the Formula Node
- c. Pre and post increment (++) and decrement (--) as in the C language
- d. The use of nested Formula Node structures

CLAD Question

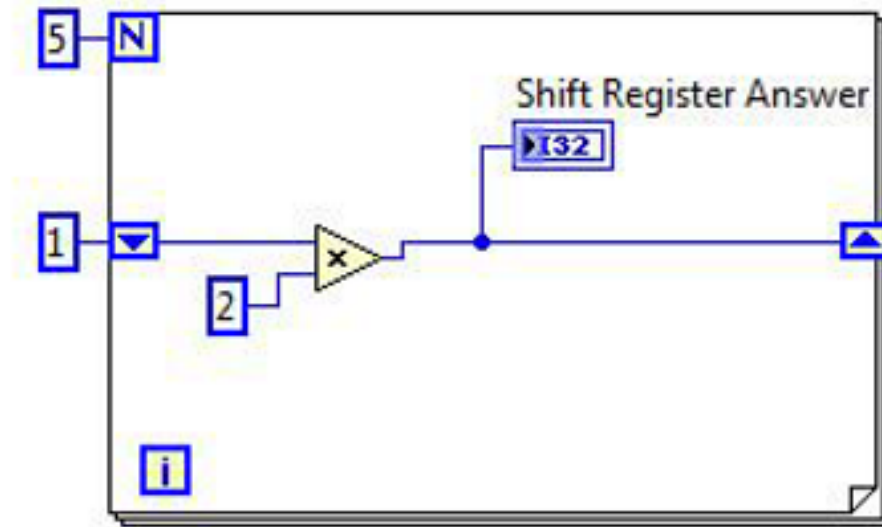
What is the value in **Shift Register Answer** after the following code has executed?



- a. 16
- b. 24
- c. 32
- d. 10

CLAD Question

What is the value in **Shift Register Answer** after the following code has executed?

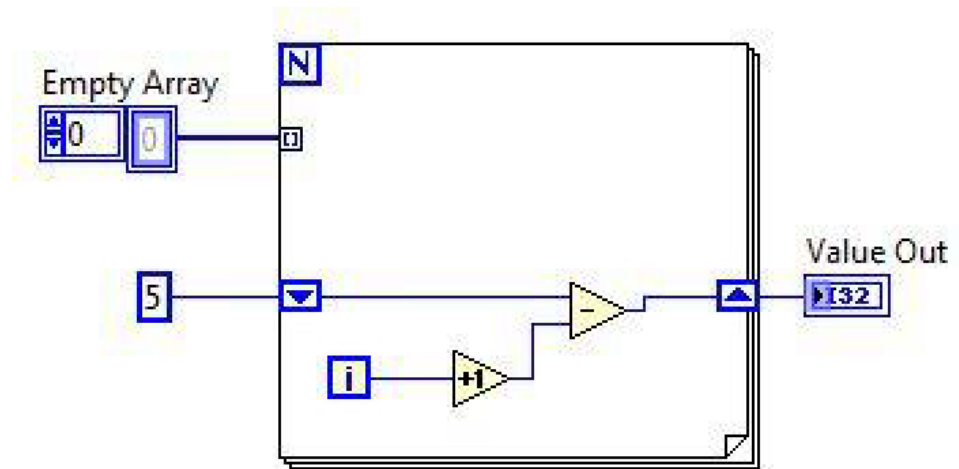


- a. 16
- b. 24
- c. 32
- d. 10

Iteration	Answer
0	2
1	4
2	8
3	16
4	32

CLAD Question

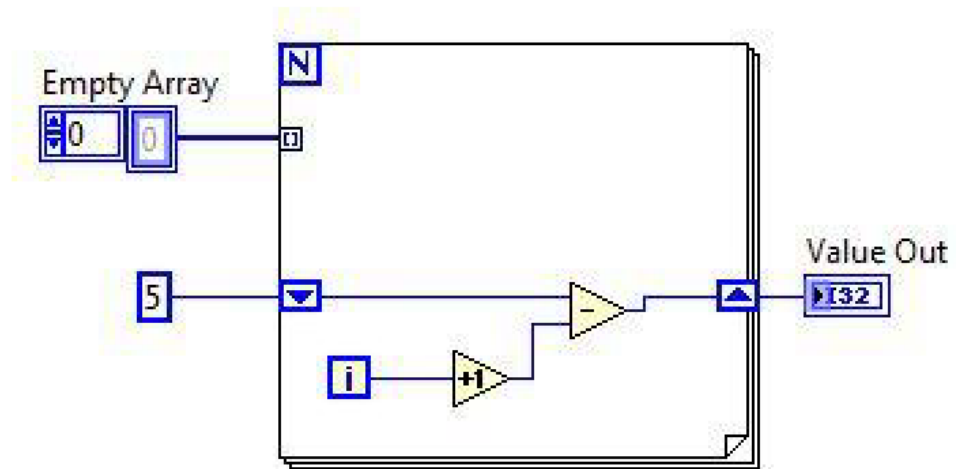
5. What value does the **Value Out** indicator display after the VI executes?



- a. 0
- b. 4
- c. 5
- d. 6

CLAD Question

5. What value does the **Value Out** indicator display after the VI executes?

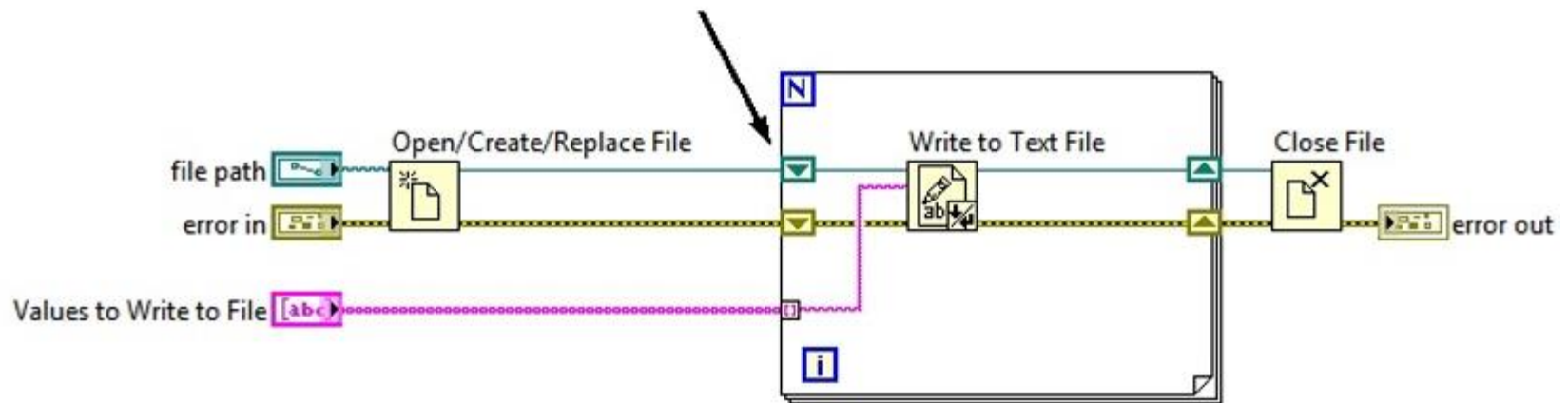


- a. 0
- b. 4
- c. 5
- d. 6

The For loop has 0 iterations since an empty array is fed into the structure. Therefore, the 5 just passes through to the output.

CLAD Question

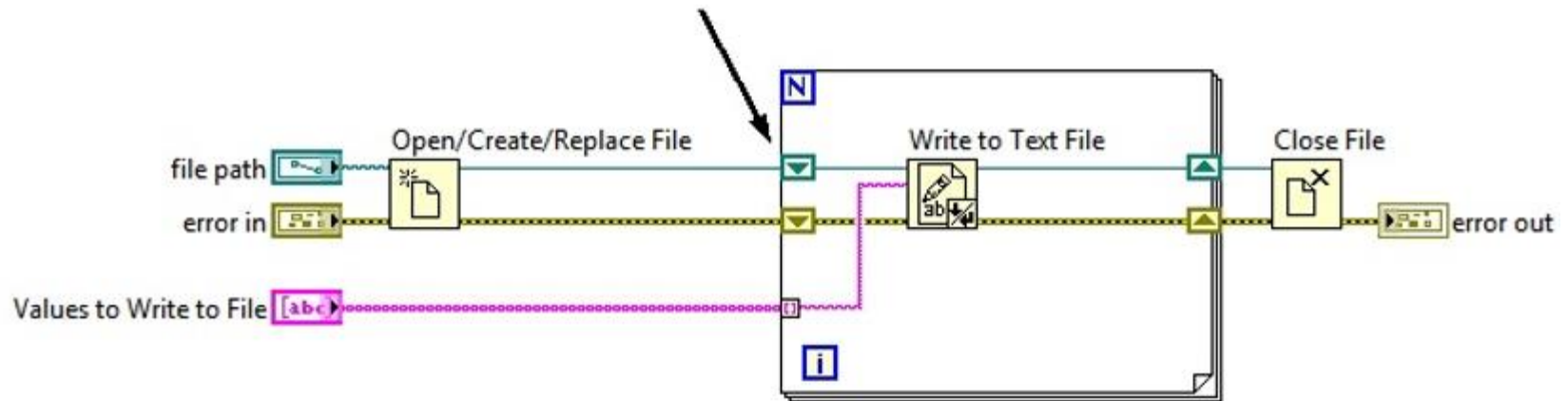
Q7: Why is a shift register used for the file refnum input on the For Loop?



- A** If the Values to Write to File array is empty, the For Loop will run zero times, but the refnum value will be passed through to the shift register output
- B** The Write to Text File function modifies the value of the refnum on each For Loop iteration
- C** It is the only way to prevent the For Loop from building an array at the refnum output terminal
- D** No reason. A regular tunnel would provide identical functionality to the shift register

CLAD Question

Q7: Why is a shift register used for the file refnum input on the For Loop?



A

If the Values to Write to File array is empty, the For Loop will run zero times, but the refnum value will be passed through to the shift register output

B

The Write to Text File function modifies the value of the refnum on each For Loop iteration

C

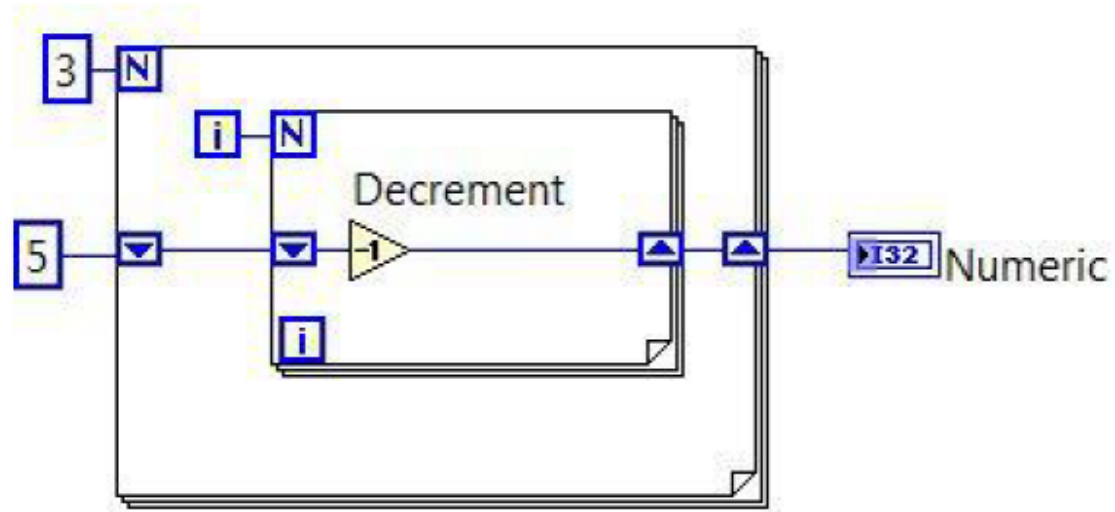
It is the only way to prevent the For Loop from building an array at the refnum output terminal

D

No reason. A regular tunnel would provide identical functionality to the shift register

CLAD Question

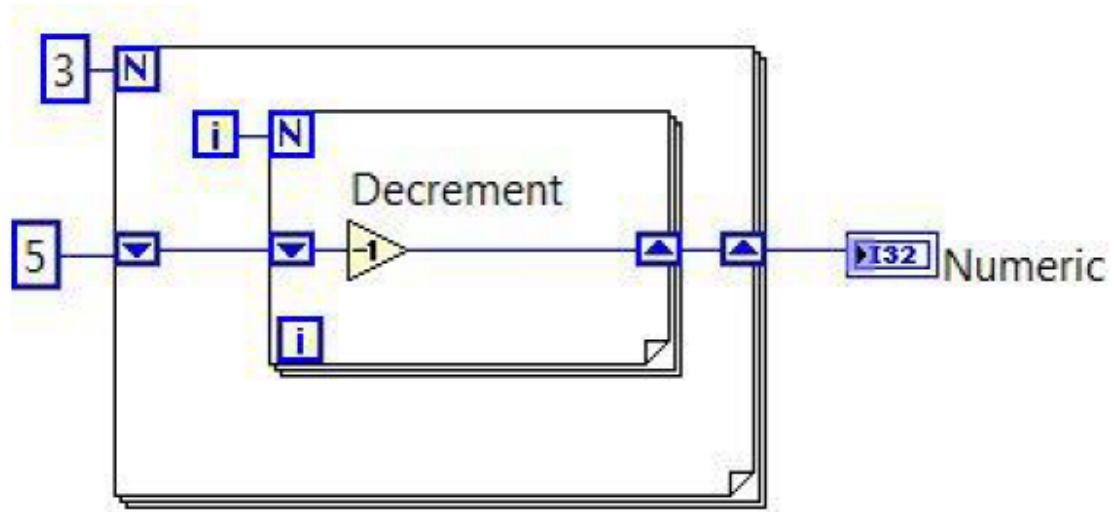
Q15: What value will be displayed in the **Numeric** indicator when the VI completes execution?



- A 0
- B 1
- C 2
- D 4

CLAD Question

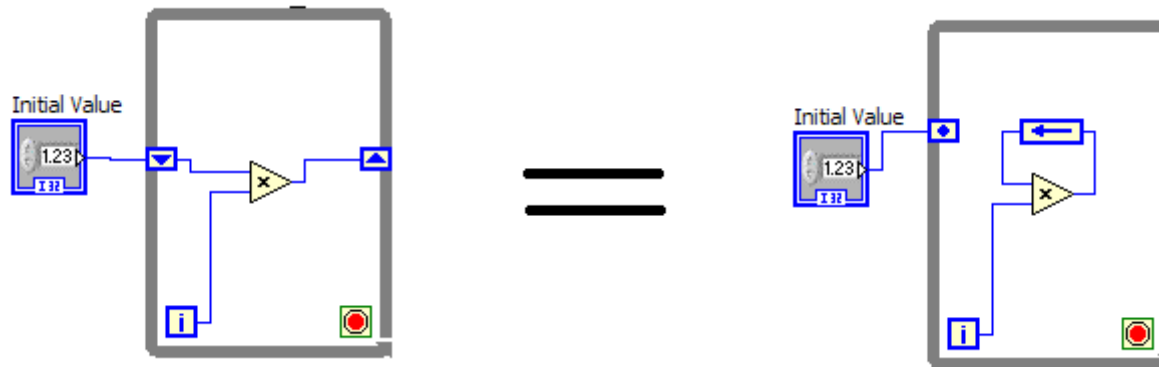
Q15: What value will be displayed in the **Numeric** indicator when the VI completes execution?



- A 0
- B 1
- C 2**
- D 4

Outer i = 0: 5 passes through. Value = 5.
Outer i = 1: 5 decrements 1 time. Value = 4.
Outer i = 2: 4 decrements twice. Value = 2.

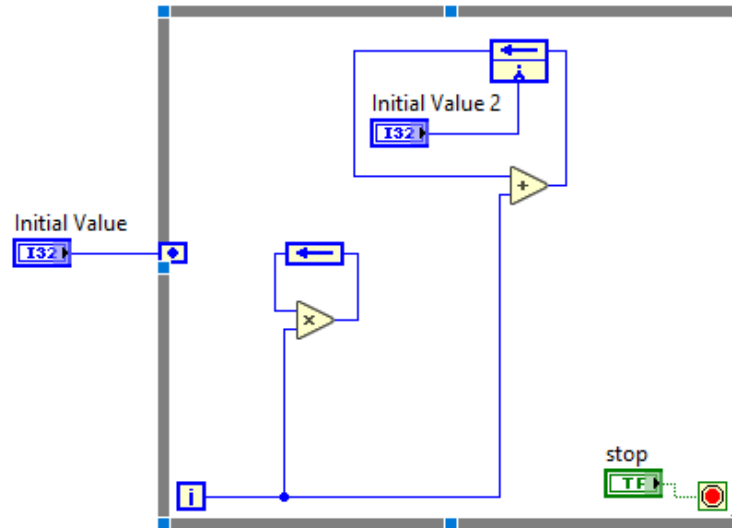
Feedback Nodes (very similar to a shift register)



FYI, these do the exact same thing. Note: I used to use Shift Registers instead of Feedback Nodes because I learned about Shift Registers first.

The feedback node has one major advantage – you don't have a wire going from one side of the while loop to the other side of the while loop! I've written many programs where the while loop takes up about 2 computer screens in width and I had wires going everywhere since I used a Shift Register instead of a Feedback Node.

Feedback Nodes (very similar to a shift register)



Notice there are 2 types of initialization with Feedback Nodes. The multiplication uses “Move Initializer One Loop Out” where the addition has the initializer directly connected to it.

The disadvantage to “Move Initializer One Loop Out” is that if you’re using several of them, then it may be hard to tell which initialization diamond goes with the appropriate feedback node (they are always at the same height is how you can tell).

The advantage to “Move Initializer One Loop Out” is that for a large sized while loop, the programmer would know that there exists a Feedback Node in the code. It makes it more apparent.

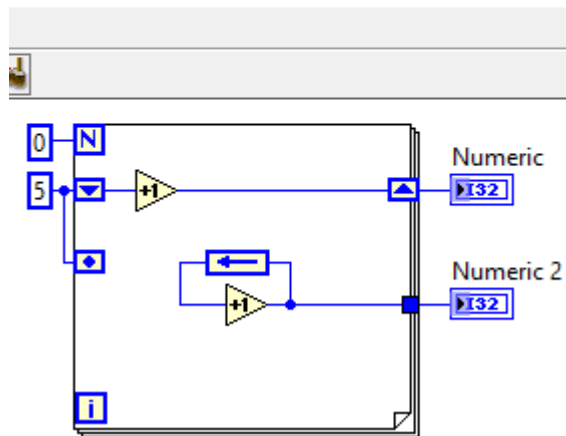
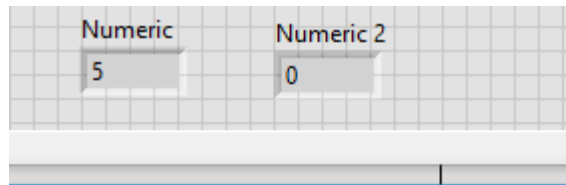


Some Differences Between Feedback Nodes and Shift Registers

I. Feedback nodes must be initialized, whereas Shift Registers do not have to be initialized

Some Differences Between Feedback Nodes and Shift Registers

2. The initialized values of Feedback Nodes do not pass through For Loops that have 0 iterations.

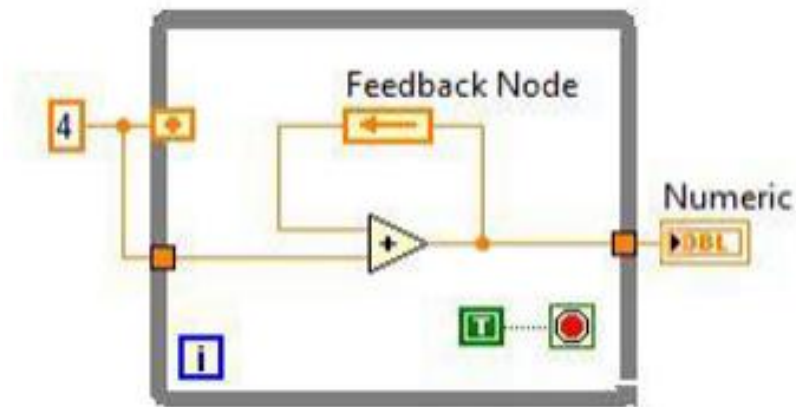


The shift register will pass the initialized 5 through the For loop even though the For loop has 0 iterations.

Notice that the Feedback Node does not do this.

CLAD Question

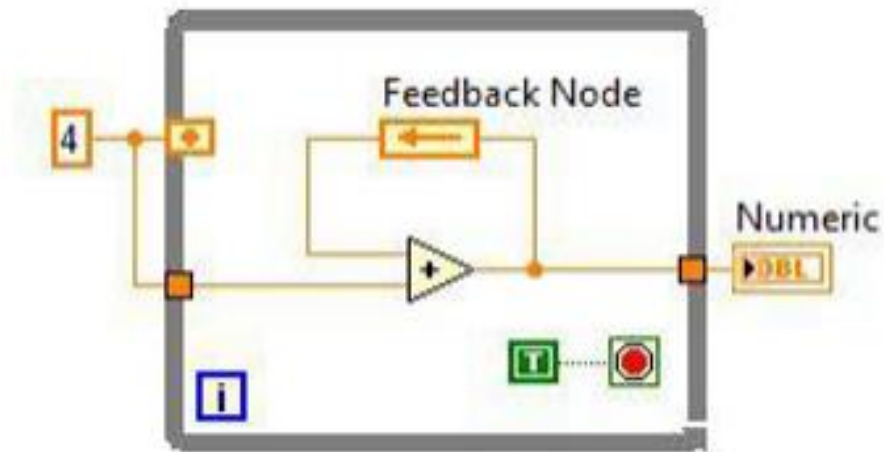
What value does the **Numeric** indicator display after this code executes?



- a. 0
- b. 4
- c. 8
- d. The While Loop iterates indefinitely

CLAD Question

What value does the **Numeric** indicator display after this code executes?



- a. 0
- b. 4
- c. 8
- d. The While Loop iterates indefinitely

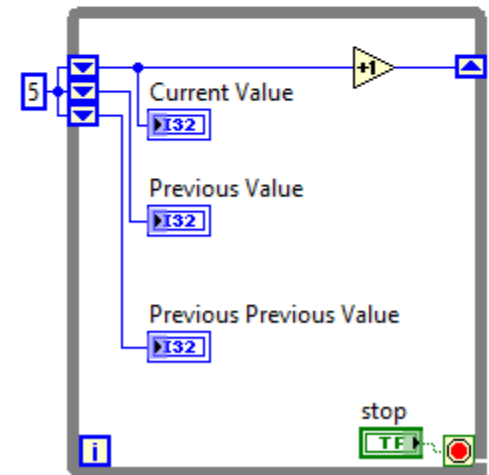
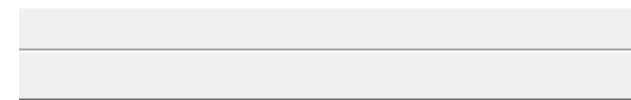
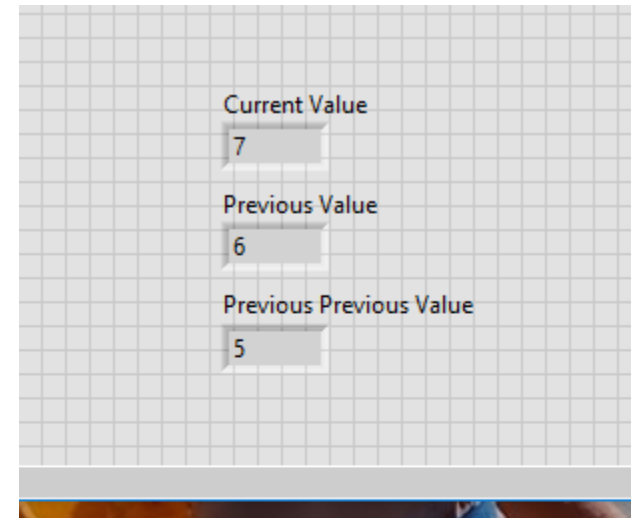
True fed into Stop, so while loop iterates one time.
 $4+4 = 8$.

Previous Previous Value of a Shift Register

You can drag down a shift register so that it has multiple icons (see image of left side of the While loop).

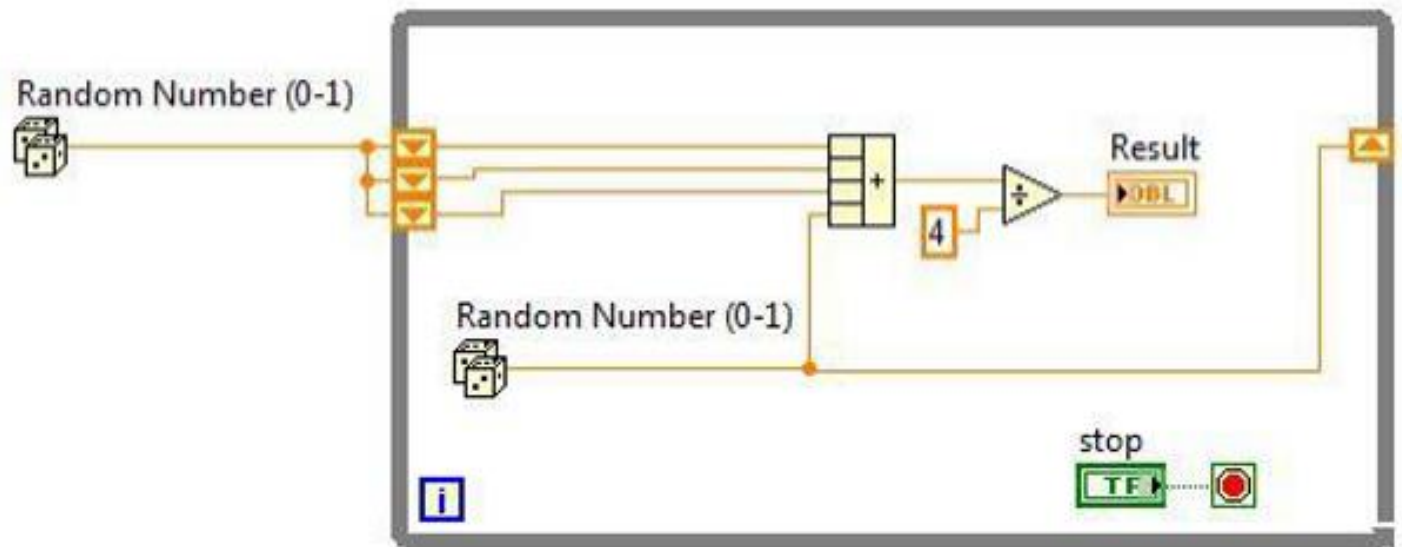
All of these shift register are initialized to 5 and then we increment the value. The top shift register is the current value, the 2nd shift register down is the previous value, and the 3rd shift register down is the previous-previous value.

So we start with 5,5,5 for the shift register on the left-hand side. Then we increment on the first iteration so it becomes 6,5,5 (*the blue 5 drops off*). Then after the next iteration, it becomes 7,6,5 (*the maroon 5 (hee hee) drops off*).



CLAD Question

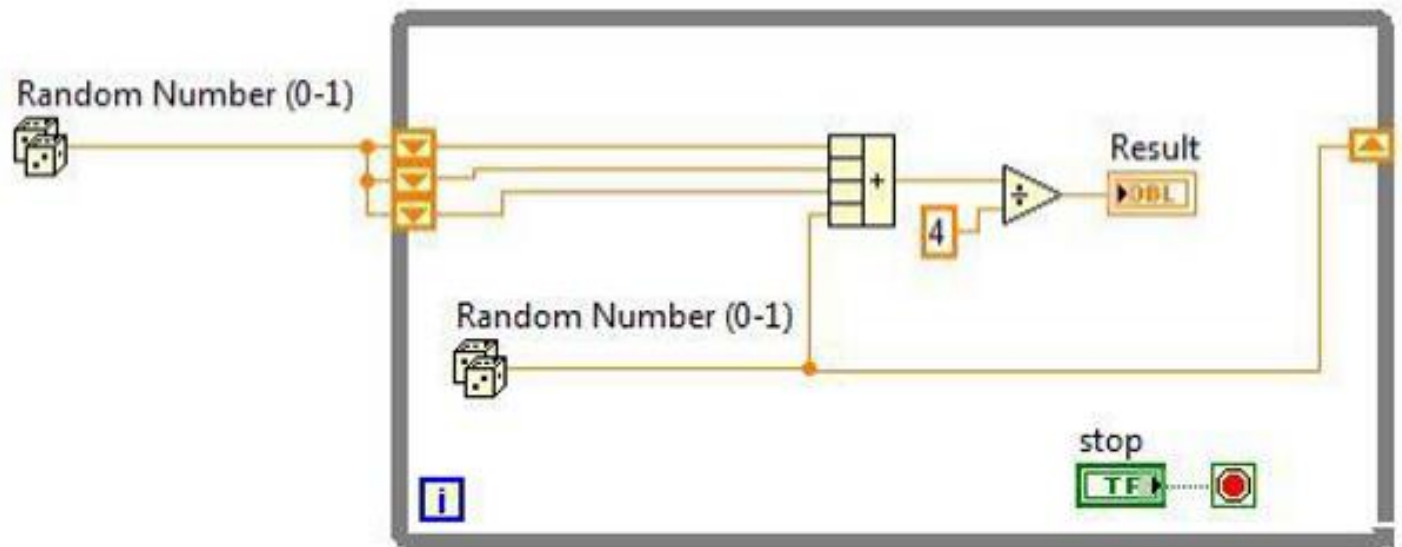
Which of the following accurately describes the output that results from execution of the following loop?



- a. The last three values from the Random Number function will be displayed.
- b. A running average of all measurements will be displayed.
- c. An average of the last four measurements will be displayed.
- d. None of the above

CLAD Question

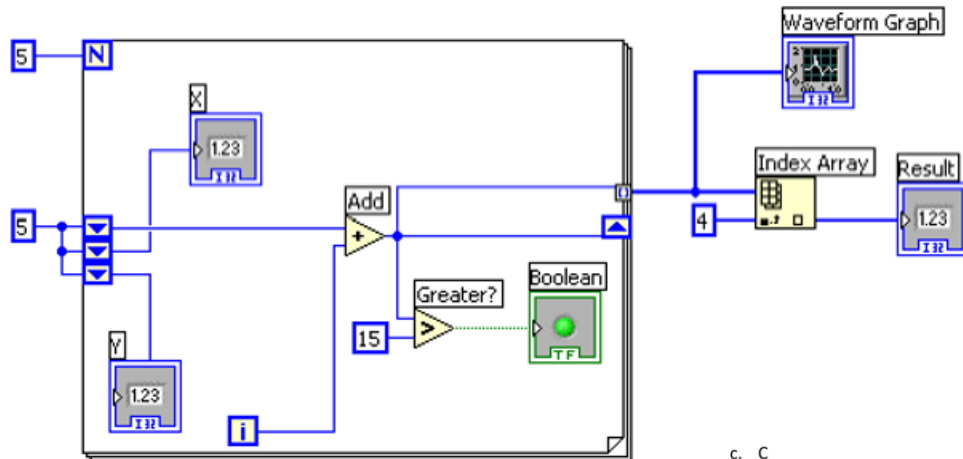
Which of the following accurately describes the output that results from execution of the following loop?



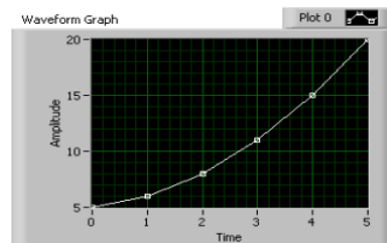
- a. The last three values from the Random Number function will be displayed.
- b. A running average of all measurements will be displayed.
- ☒ c. An average of the last four measurements will be displayed.
- d. None of the above

CLAD Question

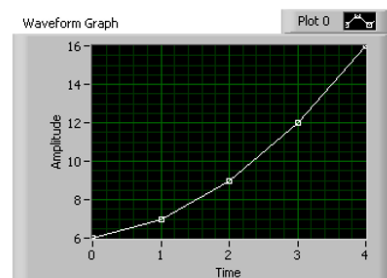
Which of the graphs below matches the output in the Waveform Graph indicator after the following code has executed?



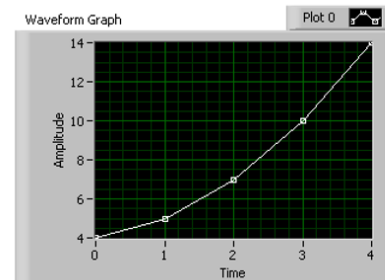
a. A



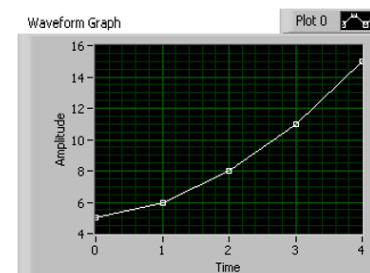
b. B



c. C

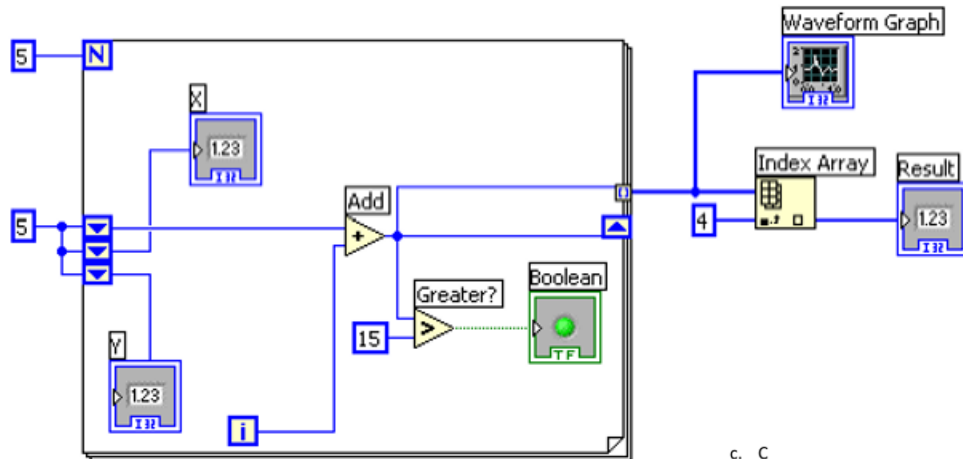


d. D



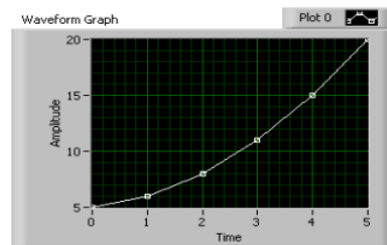
CLAD Question

Which of the graphs below matches the output in the Waveform Graph indicator after the following code has executed?

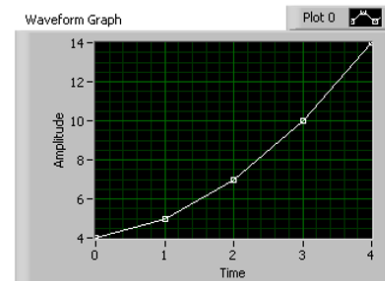


i	Value
0	5+0=5
1	5+1=6
2	6+2=8
3	8+3=11
4	11+4=15

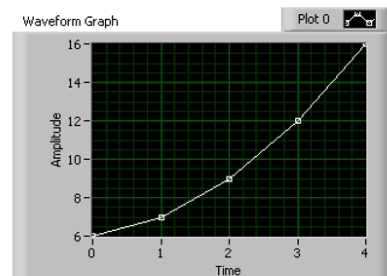
a. A



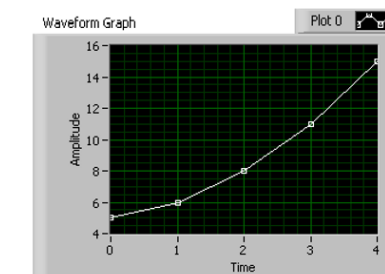
c. C



b. B

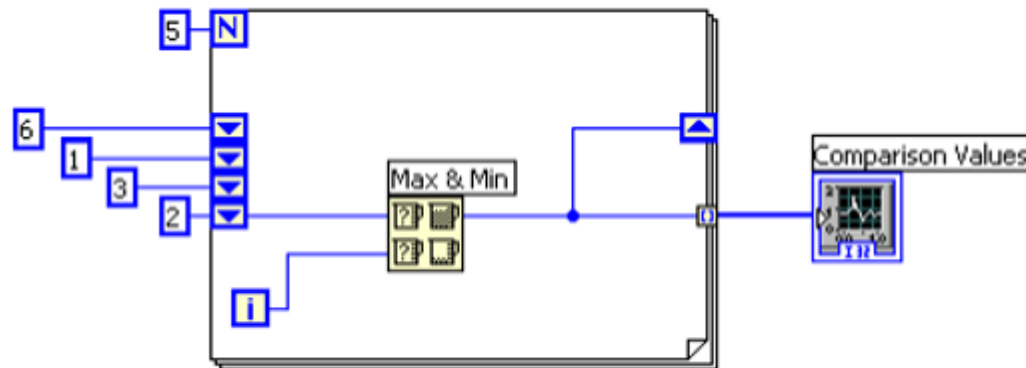


d. D



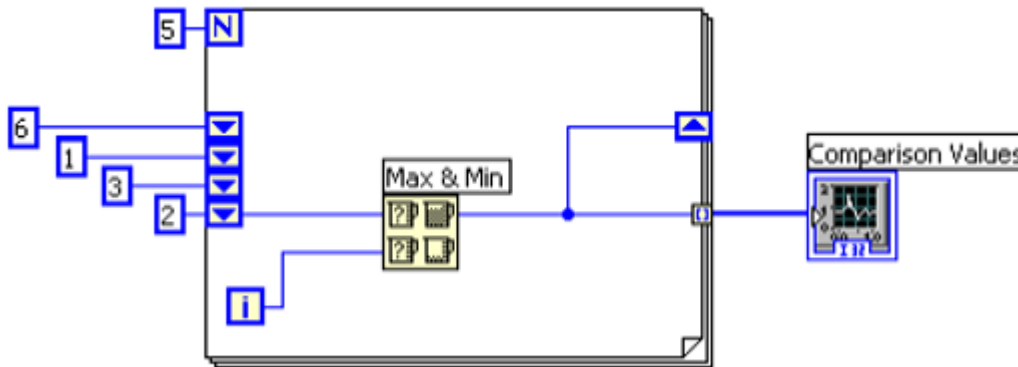
CLAD Question (modified)

What is the value of “Comparison Values” after this VI executes?



CLAD Question (modified)

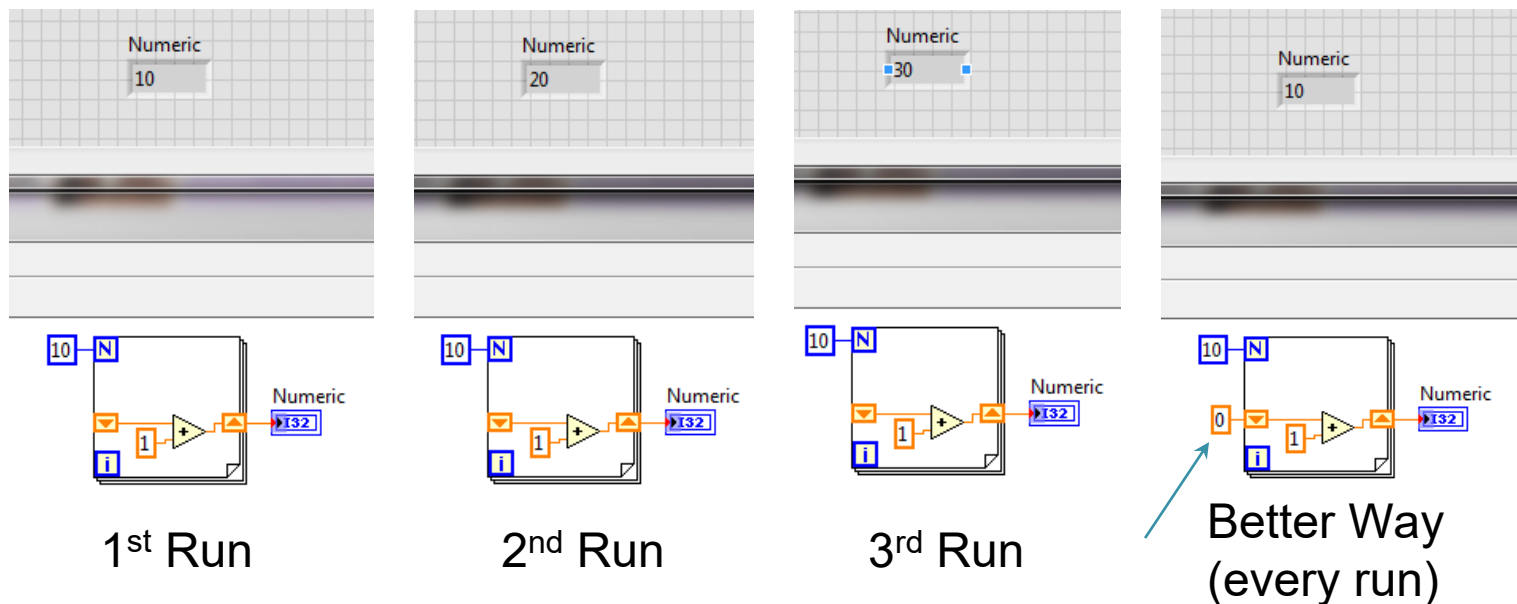
What is the value of “Comparison Values” after this VI executes?



i	Value
0	2 vs 0: max = 2
1	3 vs 1: max = 3
2	1 vs 2: max = 2
3	6 vs 3: max = 6
4	2 vs 4: max = 4

Uninitialized Shift Registers and Feedback Nodes

It's really important to initialize almost all shift registers/feedback nodes because shift registers/ feedback nodes save the old values from previous runs into memory. So if you run the program twice, it will take the value of the last run and apply it to the next run.

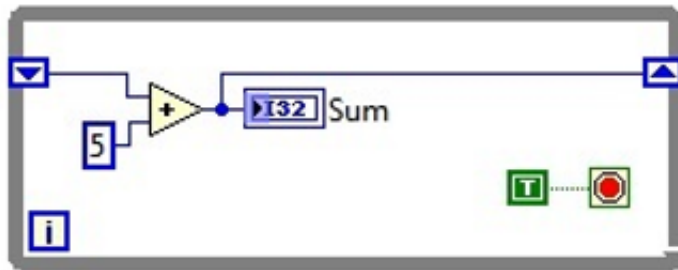


Notice that the 1st run gives the correct value. The 2nd run uses the 10 from the first run to start adding from (instead of 0). The better way to do this is to initialize the shift register with a 0 so you get the same answer with every run.

CLAD Question

Q8: The following SubVI is loaded into memory and then run. At some later time it is still in memory and is run a second time.

What is the value is displayed in the **Sum** indicator when the SubVI completes execution the second time it is run?

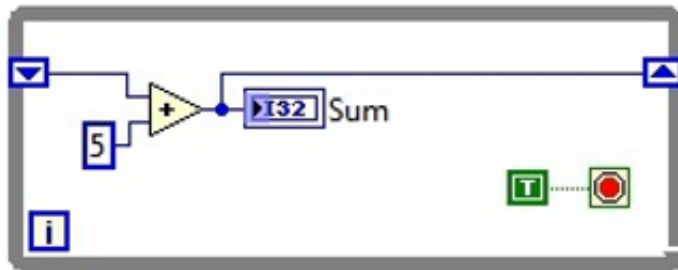


- A 5
- B 10
- C There is no way to know
- D 0

CLAD Question

Q8: The following SubVI is loaded into memory and then run. At some later time it is still in memory and is run a second time.

What is the value is displayed in the **Sum** indicator when the SubVI completes execution the second time it is run?



- A 5
- B 10**
- C There is no way to know
- D 0

Since the shift register is uninitialized, the value of the shift register stays stored in memory for the next run of the VI.

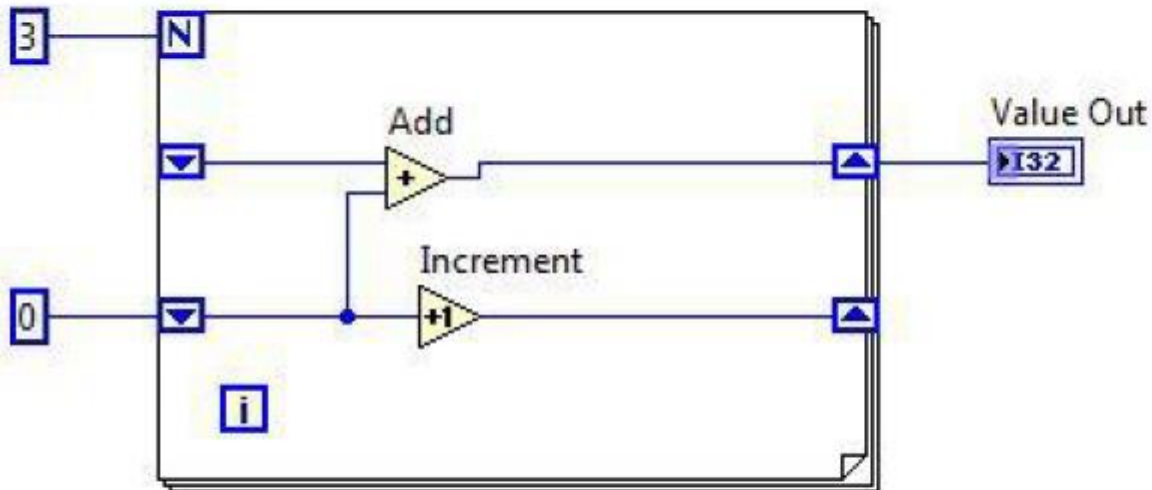
First run: SR defaults to 0: $0+5 = 5$

2nd run: SR uses previous value of 5: $5+5 = 10$.

CLAD Question

Q6: The VI is open and run twice without being closed or modified.

What value is displayed in the **Value Out** indicator after the second execution of the VI?

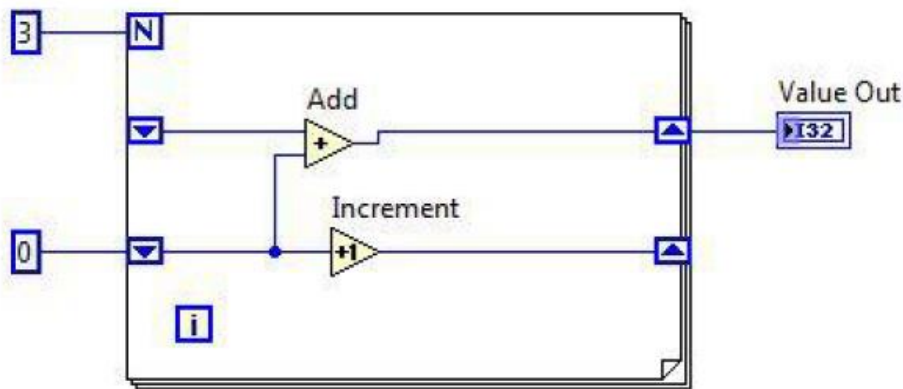


- A 3
- B 4
- C 5
- D 6

CLAD Question

Q6: The VI is open and run twice without being closed or modified.

What value is displayed in the **Value Out** indicator after the second execution of the VI?



Notice the uninitialized shift register. We assume it defaults to 0.

- A 3
- B 4
- C 5
- D 6

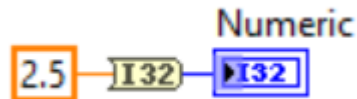
i	Value Top	Value Bottom
0	0+0=0	0+1=1
1	0+1=1	1+1=2
2	1+2=3	2+1=3
0	3+0=3	0+1=1
1	3+1=4	1+1=2
2	4+2=6	2+1=3

Coercion Dot

A coercion dot converts one type of data into another type. In this example, it converts a double into an integer.



Evidently, coercion dots use memory for the conversion. This is minimal unless it's converting a large array. We can avoid the coercion dot by converting to the correct type (in this case 32 bit integer).



By the way, when LabVIEW rounds a number that ends in .5, it rounds to the nearest **even** integer (which is strange to me). So 2.5 would round to 2 (not 3!).

2.51 rounds to 3.

2.49 rounds to 2.