LabVIEW for Physicists

Daniel Janse van Rensburg

June 2022

Contents

Chapter 1

Introduction

1.1 Welcome

Welcome to the LabVIEW course, for 3rd year physics students, at UJ.
In your first semester of 3^{rd} year, you have been exposed to a programming language called `C++`, an *Object-Oriented* text based computer language.
With this mighty tool in hand, you made a little microcontroller do your bidding, specifically an `ATmega328` found on the Arduino boards.

You may have noticed how terse statements in `C++` can get, sometimes obfuscating the programmers intent, see listing 1.1 on page 2 for an extreme example. In this course however, we will not be dealing with text based statements, instead you will make a computer bend to your will by means of pictures and threads!

Before we jump into the details, let us first compare two programs performing the same task, one written in `C`, and one assembled in `LabVIEW`.
Suppose we want to capture two decimal numbers from a user in order to compute the sum of said numbers:

C Programming

In listing 1.2, on page 3, the first line imports the required libraries for `C` to handle our input and output, or `IO`. We then have to declare what type of variables we are storing and give them names, in this case `numberA & numberB`. In this case, the values are initialised to 0, not required for this example, but good practice regardless. The numbers are then captured from the user on line 7 & 8. Finally, the results are printed so that the user may see that their system can indeed do basic arithmetic.

```
char*l="ustvrtsuqqqqqqqqyyyyyyyy}{|~z|{}"
"   76Lsabcddcba .pknbrq  PKNBRQ ?A6J57IKJT576,+-48HLSU";
#define F getchar()&z
#define v X(0,0,0,21,
#define Z while(
#define _ ;if(
#define P return--G,y^=8,
B,i,y,u,b,I[411],*G=I,x=10,z=15,M=1e4;X(w,c,h,e,S,s){int t,
 o,L,E,d,O=e,N=-M*M,K
=78-h<<x,p,*g,n,*m,A,q,r,C,J,a=y?-x:x;y^=8;G++;d=w||s&&s>=h
 &&v 0,0)>M;do{_ o=I[
p=O]){q=o&z^y _ q<7){A=q--&2?8:4;C=o-9&z?q["& .$  "]:42;do{
 r=I[p+=C[l]-64]_!w|p
==w){g=q|p+a-S?0:I+S _!r&(q|A<3||g)||(r+1&z^y)>9&&q|A>2){_
 m=!(r-2&7))P G[1]=O,
K;J=n=o&z;E=I[p-a]&z;t=q|E-7?n:(n+=2,6^y);Z n<=t){L=r?l[r
 &7]*9-189-h-q:0 _ s)L
+=(1-q?l[p/x+5]-l[O/x+5]+l[p%x+6]*-~!q-l[O%x+6]+o/16*8:!!m
 *9)+(q?0:!(I[p-1]^n)+
!(I[p+1]^n)+l[n&7]*9-386+!!g*99+(A<2))+!(E^y^9)_ s>h||1<s&s
 ==h&&L>z|d){p[I]=n,O
[I]=m?*g=*m,*m=0:g?*g=0:0;L-=X(s>h|d?0:p,L-N,h+1,G[1],J=q|A
 >1?0:p,s)_!(h||s-1|B
-O|i-n|p-b|L<-M))P y^=8,u=J;J=q-1|A<7||m||!s|d|r|o<z||v
 0,0)>M;O[I]=o;p[I]=r;m?
*m=*g,*g=0:g?*g=9^y:0;}_ L>N){*G=O _ s>1){_ h&&c-L<0)P L _!
 h)i=n,B=O,b=p;}N=L;}
n+=J||(g=I+p,m=p<O?g-3:g+2,*m<z|m[O-p]||I[p+=p-O]);}}}Z!r&
 q>2||(p=O,q|A>2|o>z&
!r&&++C*--A));}}}Z++O>98?O=20:e-O);P N+M*M&&N>-K+1924|d?N
 :0;}main(){Z++B<121)*G
++=B/x%x<2|B%x<2?7:B/x&4?0:*l++&31;Z B=19){Z B++<99)putchar
 (B%x?l[B[I]|16]:x)_
x-(B=F)){i=I[B+=(x-F)*x]&z;b=F;b+=(x-F)*x;Z x-(*G=F))i=*G
 ^8^y;}else v u,5);v u,
1);}}

```

Listing 1.1: This listing is executable C code see http://www.nanochess.org

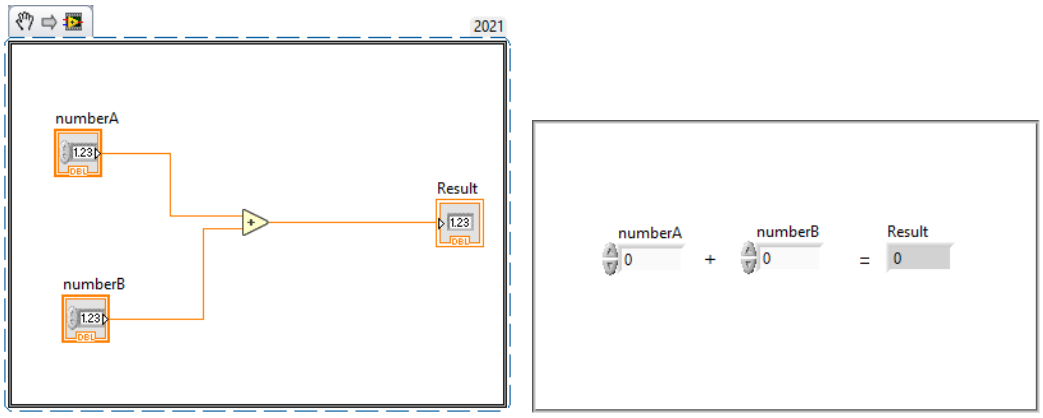
```c
#include<stdio.h>

int main() {

  double numberA = 0;
  double numberB = 0;
  scanf("%f", &numberA);
  scanf("%f", &numberB);

  printf("%d + %d = ",numberA, numberB, numberA + numberB);
  return 0;
}
```

Listing 1.2: A simple C program to add two user typed numbers



(a) A LabVIEW block diagram. (b) A LabVIEW front panel.

Figure 1.1: A LabVIEW program to sum two numbers.

LabVIEW Programming

Without having any idea what LabVIEW is, I am sure you are able to follow what figure 1.1a, on page 3, is attempting to convey. In figure 1.1b you will find the *Graphical User Interface*, or GUI, for the summing program. The two figures are one program developed simultaneously, we will dive into the details throughout the course.

Before you even ask, yes it is really that simple. You will find that creating programs in LabVIEW is fun and rewarding, so relax and take it easy. By the end of this course, you will be able to create complex programs to help you solve problems in physics, as well as bridge the gap between the computer world and the real world.

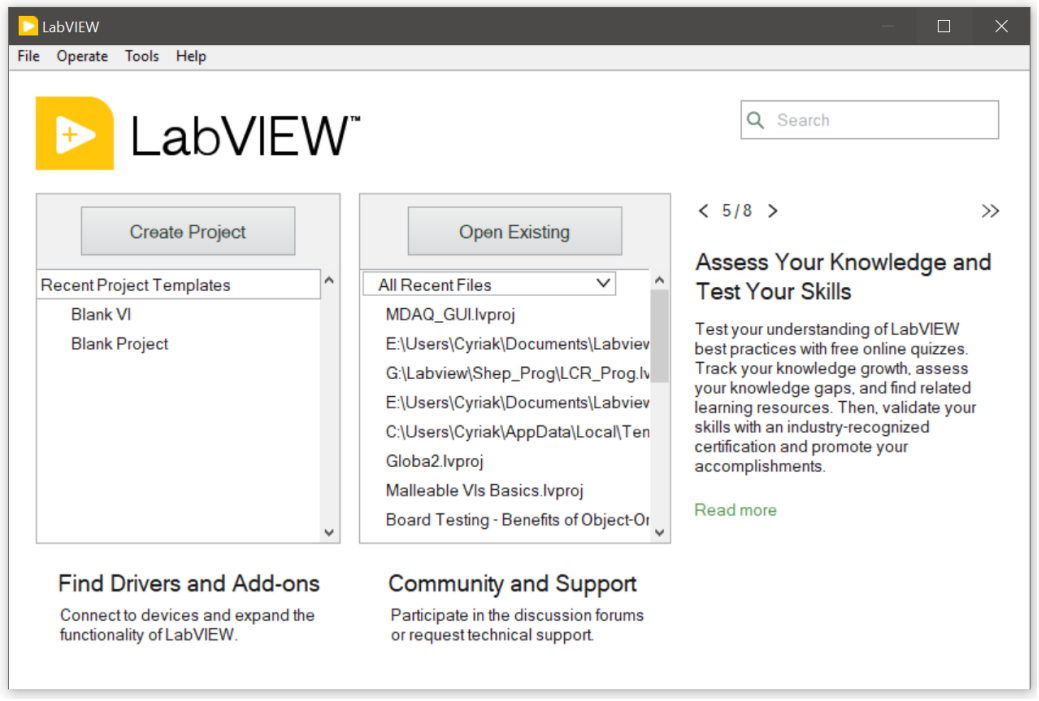


Figure 1.2: The window LabVIEW greats you with on startup.

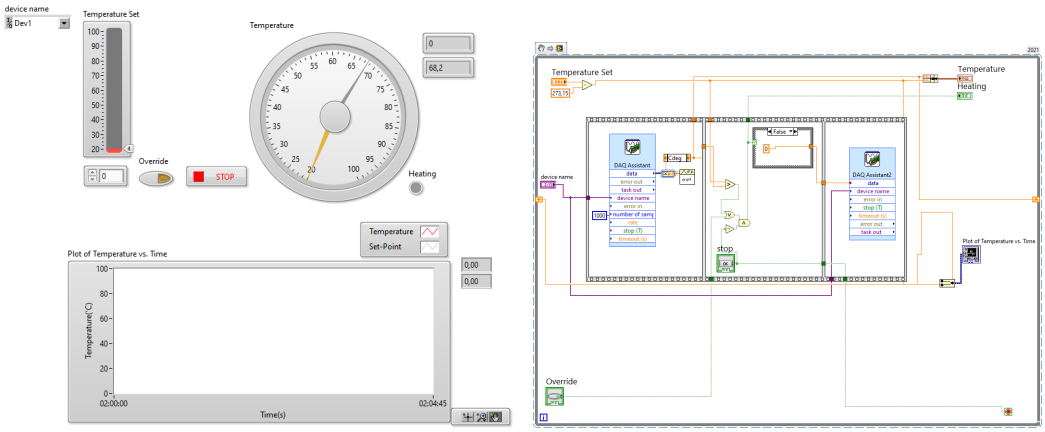
1.2 The LabVIEW Interface

1.2.1 Introduction

LabVIEW usually greats you with a launcher, figure 1.2. From here you may open your recent projects, start a fresh project, or open a virtual instrument, known as a VI. For now, from the "File" drop down menu, select "New VI" or press `Ctrl+N` on your keyboard.

Two windows will pop up, a "Front Panel" and a "Block Diagram". You will need to familiarise yourself with the LabVIEW interface, this is best done by exploration, trail and error. Simply mousing over any button should give you some clue as to what the button does or what is contained in the menus. You probably would not break your computer or the LabVIEW installation by playing around with the interface.

In the next few sections we will go over what is meant by a VI, the difference between the block diagram and the front panel, how to cycle between these views, and how to place objects on these windows.



(a) A more useful front panel. (b) The logic behind the front panel.

Figure 1.3: A program which turns a kettle on and off to achieve a set temperature.

1.2.2 The VI

As stated before, VI is short for virtual instrument. The idea is that you create an interface which may be operated by a human, usually with a mouse and keyboard. This interface is called the "Front Panel" in LabVIEW. You then glue the elements of the front panel together in the "Block Diagram", this is where the LabVIEW magic happens, or the LabVIEW logic execution engine, if you do not believe in magic.

Figure 1.3a, on page 5, shows one of the first front panels I have ever created along with the block diagram in figure 1.3b. By the end of this course you will be able to understand exactly what is going on there so do not let it intimidate you. Assert dominance over your computer, lest it assert dominance over you.

The Front Panel

Figure 1.4 shows an empty front panel. `Right clicking` anywhere on the grey grid will open a menu containing controls, known as the "controls palette". Do not be overwhelmed, given some time you will get a feel for where the most important controls are. As mentioned, and will be repeated throughout this book, explore the interface on your own.

Here is a little secret, simply press `Ctrl+H`. This will open a floating window called "Context Help", your new life-line in LabVIEW. Hover over

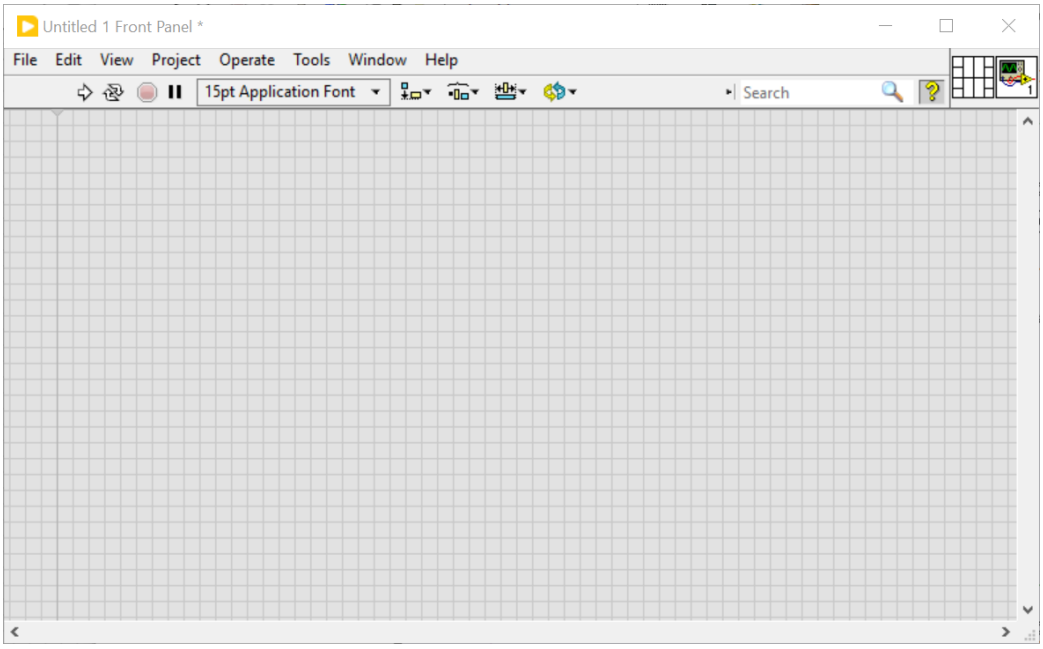


Figure 1.4: A barren wasteland of a front panel, not very useful.

anything and it will give you some information about what you are looking at. Pressing "Detailed help" takes you to the LabVIEW help files. You will spend a great deal of time reading these documents as you progress through your LabVIEW journey so you should know where to find it.

If you figured out how to place down controls in LabVIEW, that is great, do not let the interface intimidate you. If you have not done so yet, we will go through the details in section 1.4.2.

The Block Diagram

The sibling panel to figure 1.4 may be found in figure 1.5. As before, you may `Right click` on the white background to open the "functions palette". This is where you will spend most of your time in LabVIEW, other than the help files that is. You thread together small function blocks to build the logic of your program. If you are reading this and can't find the block diagram panel, simply press `Ctrl-E`, this will switch from the front panel to the block diagram and vice versa. This is probably the most important hot-key in LabVIEW so learn it, you will often have to flip between the two windows.

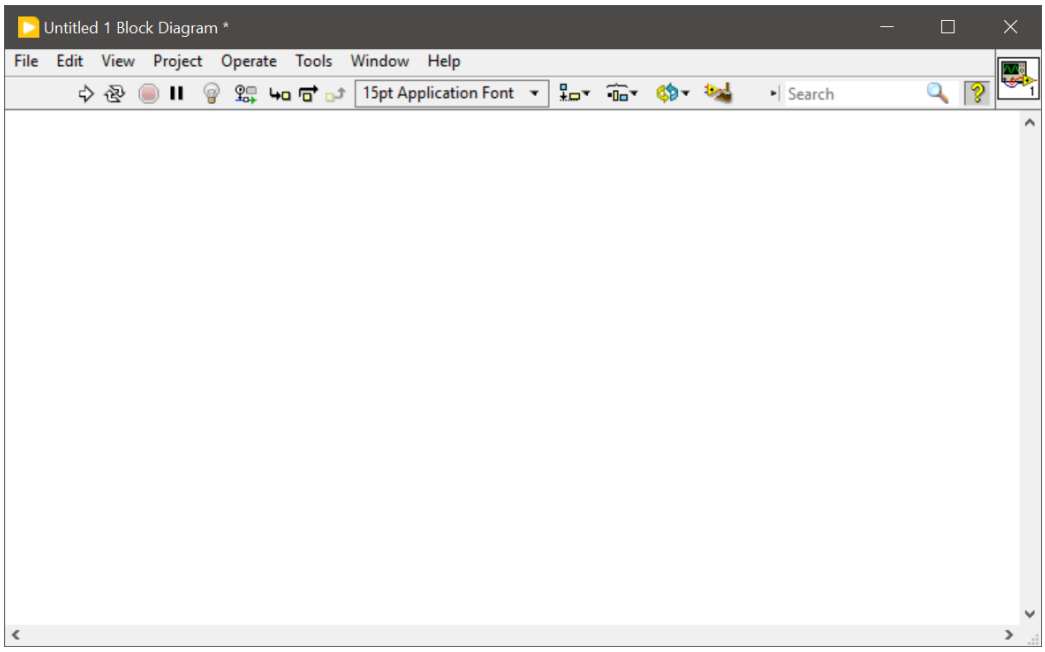


Figure 1.5: The void, even interstellar space has more going on than this panel.

1.3 LabVIEW variables and you

1.3.1 Overview

From your knowlage of mathematics, you understand the concept of a variable. This concept is of fundamental importance to all fields of computer science so let us take a slight detour before we dig further into LabVIEW.

Variables are named containers in which we may store information. It is then possible to read from, or write to, this container using it's name. At any moment in time, there are several named variables in your head, for example: `cellphoneNumber`, `currentDate`, `amountOfCoffeeHadToday`, etc.

With these examples, it is clear that some variables are not of the same type, i.e. saying "I had 3 June 2022 coffees today" makes no sense. You should know by now, or if you do not you will learn very soon, that computers are incredibly stupid and they will happily add 3 June 2022 to 072 543 7711 and give you a result. (The answer is 29 May 2045 by the way).

In general, it is your job to tell the computer what type a variable is. This

is not strictly true as many programming languages can infer the data type from your input, however for LabVIEW and programming languages such as `C` and `C++`, you are responsible.

1.3.2 Variable Types

Boolean

Perhaps the most fundamental variable type is the boolean, or bool for short. It consists of either yes or no, true or false, `1` or `0`, you get the picture. In LabVIEW, bools form the basis of nearly all your decision making code. It is represented as green icons, seen in figure 1.6a, on page 9.

Floating Point

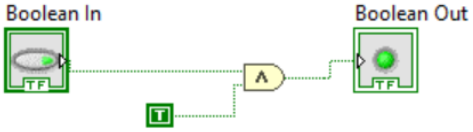
Floating point variables hold decimal numbers, in an application this may be the value of a magnetic field, the temperature of a thermocouple, basically any number that you can think of that would fit into 64bits of memory (more on that if you do computational physics in honours physics). All floating point values are represented in Labview as orange icons, seen in figure 1.6b, on page 9.

Integer

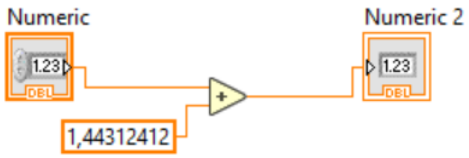
The integer type is self explanatory. This is the only type of number which may be represented in LabVIEW as exact numbers and is ideal for comparison and counting. It is worth mentioning briefly that integers exist as two types namely signed and unsigned. Negative numbers are contained in signed integers, unsigned integers are strictly positive. The nuances of signedness are best left for the computational physics coarse in honours. Integer types are represented as blue icons in LabVIEW, see figure 1.6c on page 9.

Strings

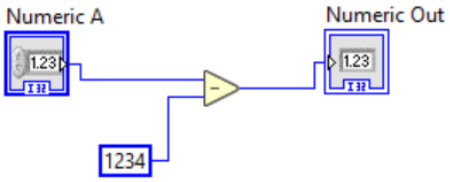
This line that you are reading is a string. Strings are made up of characters, itself a type of variable. Characters are not as important in LabVIEW as in other programming languages so they do not deserve their own subsection. Strings may be used to convey information to users, it may also be used to store information on your hard-disk. Strings are pink in LabVIEW, reference 1.6d on page 9.



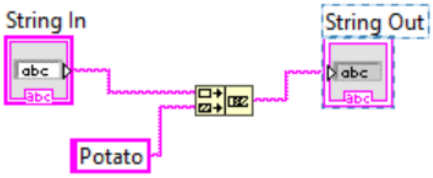
(a) Boolean controls and constants.



(b) Floating point controls and constants.



(c) Integer data types.



(d) String data types

Figure 1.6: Fundamental LabVIEW data types.

Arrays

Arrays are homogenous sections of sequential memory. Any of the above data types may be formed into an array. This variable type allows you to reference a collection of data points as a single item. Unless you derive joy from naming thousands of individual variables, you should use arrays when working with more than a handful of data points. Arrays are indexed from 0 in LabVIEW, important if you have programmed in Fortran before. Arrays follow the colour of their contents in LabVIEW, i.e. a blue array contains only integers.

Aggregate Types

Variable types may be grouped together to form a new variable type, usually with some relationship among one another. In LabVIEW, these types are known as clusters, similar to structs from the `C` family of languages. We will leave clusters for now, they are only really useful once your programs start getting big.

1.4 Your First LabVIEW program

1.4.1 Problem statement

We must create a program in which a user gives three integer values, `numberA`, `numberB`, and `NumberC`. The program needs to compute and output the sum of `numberA` and `numberB` as well as indicate if this sum is larger than `numberC`.

The rest of this section will show you in detail how such a program is made in LabVIEW, however it will not explain the details of the program in depth. This is left for later sections in chapter 2.

1.4.2 Implimentation

Front Panel

You may think of the front panel as your scratch pad. What information does the user need to provide for our program and how may the computer display the outcome of a process?

Anywhere on the front panel, `right click` to show the control palette. Mouse over the top left folder called "Numeric", a new window will pop up. Drag your mouse into this window and select "Numeric Control" by clicking once on the icon. You will notice that your mouse cursor now changed to a little grab hand with an outline of the proposed control. Simply click where you would like this control to live.

The name "Numeric" is not helpful to us. If the name is highlighted, white text in a black box, you may type a new name and it would replace the old one. Call this control "numberA". If the name is not highlighted, simply `double click` on the name until you see the black box with white text and then type "numberA". Once you have typed the name, `left click` anywhere on the background to make the change permanent.

Figure 1.7 shows how selected text looks like and what you should have on your front panel after you have renamed the control. Repeat what you have just done two more times to create "numberB" and "numberC".

For our resulting number, open the controls palette, go to "Numeric" and select the "Numeric Indicator" icon. Rename this to "Result".

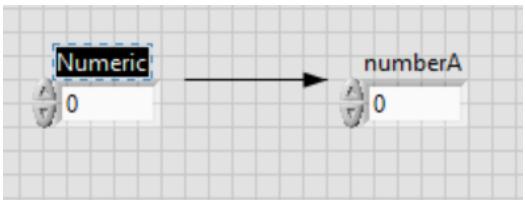


Figure 1.7: What you should have after renaming the control.

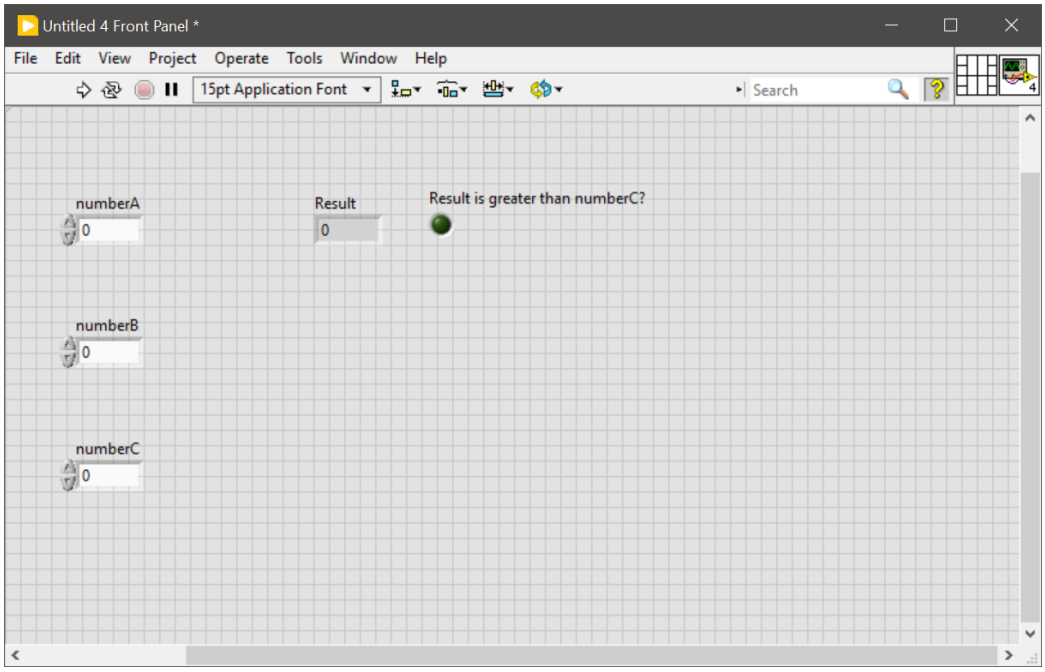


Figure 1.8: Roughly what your front panel should look like..

Finally, open the control palette again, this time go to the "Boolean" folder and select the "Round LED". You should place this close to your result indicator. You will start to notice that grouping controls together with regards to input and output makes it easier to understand the intent of your program. Lastly, rename this indicator to "Result is greater than numberC?".

Your front panel should now look like the panel in figure 1.8 on page 11.

If you are not satisfied with your layout, you may `left click` and hold on the edges of a control and drag it to where you would like to to be. This might take some getting used too. Your cursor will change from a cross to a pointer when you hover over a part of a control which is allowed to be moved.

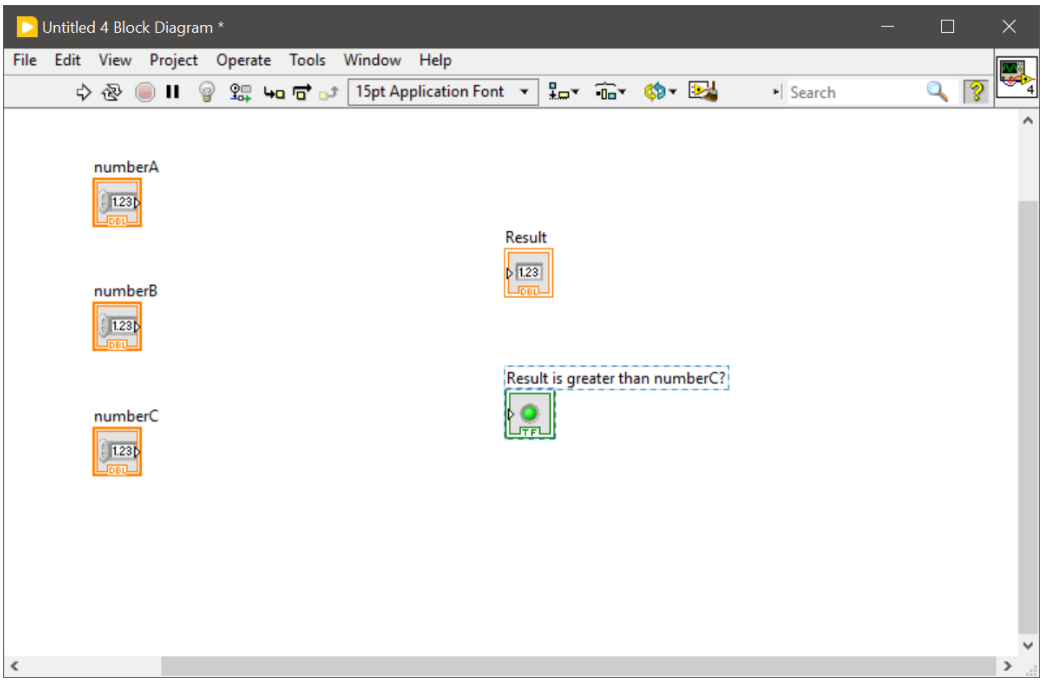


Figure 1.9: More or less what your block diagram should look like.

Block Diagram

While looking at your front panel, press `Ctrl-E` to flip to your block diagram. You would notice that what you have done on the front panel is reflected in the block diagram. Your inputs have little white arrows pointing out of the icon to the right. The outputs, also known as indicators in LabVIEW, have white arrows pointing into the icon on the left.

You should move these icons so that the interface flows from left to right, that is, inputs on the left and outputs on the right. You move the icons like you have moved the front panel elements, just click, hold, and drag the icons to where you think they should live. See figure 1.9 for inspiration.

You are now ready to build the logic of your program. From our problem statement, we need to sum numberA and numberB. Both are numerical values so we go to the "Numeric" folder of the function palette. The first icon you see is the add function. Place it somewhere in the middle of your inputs and outputs.

Hover your mouse cursor over the newly placed function, you will notice

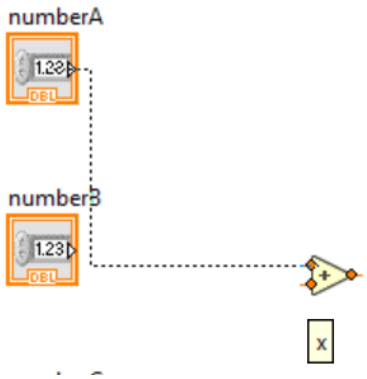


Figure 1.10: A ghostly wire follows your cursor to your desired terminal.

that little orange terminals are highlighted. If you hover over one of your number inputs, you will also see a little terminal, conveniently placed next to the white arrow. `Left click` on the terminal of numberA, moving your mouse around the screen you will see a black striped line following your cursor. Then `left click` on one of the terminals of the add function. You would see something like figure 1.10 before you click. This will create an orange line, also known as a wire, connecting your input to the function.

Connect numberB to the add function, then connect the output of the add function to the result indicator. You should now have a block diagram that looks like figure 1.11 on page 14.

From the function palette, select the "comparison" folder and look for the function that says "greater than". It is the triangle with the ¿ symbol on it. Place this somewhere below the add function, but still between the controls and the indicators.

You should now wire the output from this function to the terminal of your LED. The function compares two inputs and sends a true value to the LED if the one input is greater than the other, but how do we know which input is which? The "context help" window will show you which input is which, see figure 1.12. If you do not see this window, just press `Ctrl+H` on your keyboard.

Wire in the value from numberC into the comparison function and connect the other input to the output of your summing function. You may do this by starting the wire at the input of the comparison function and `Left clicking` on the wire leading from the summing function to your result in-

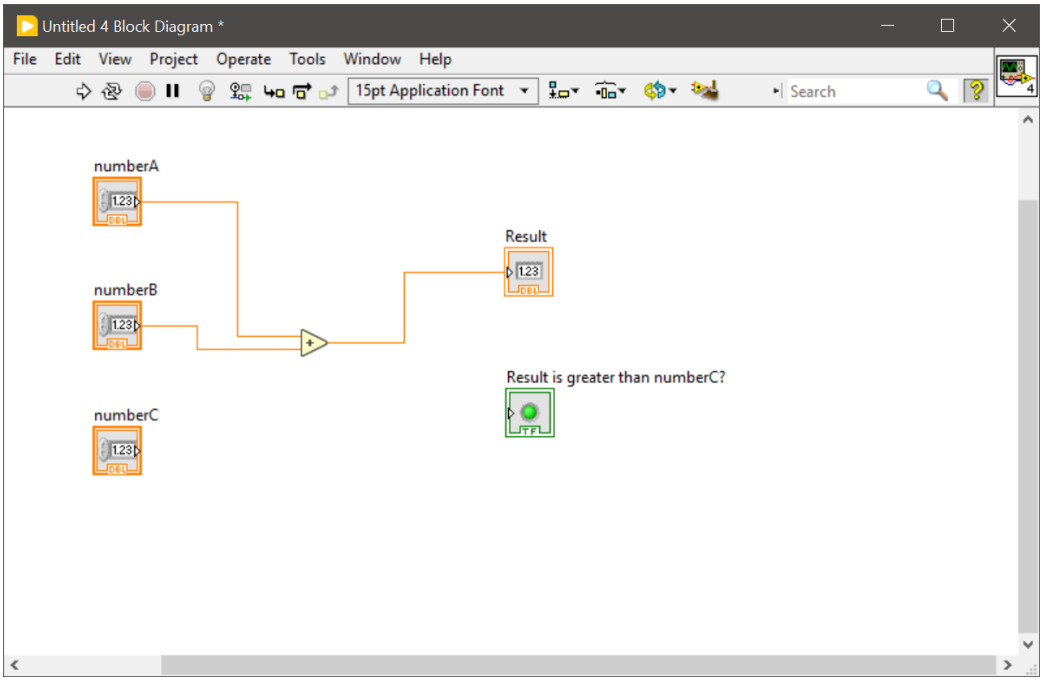


Figure 1.11: The summing logic wired correctly, your block diagram should look like this.

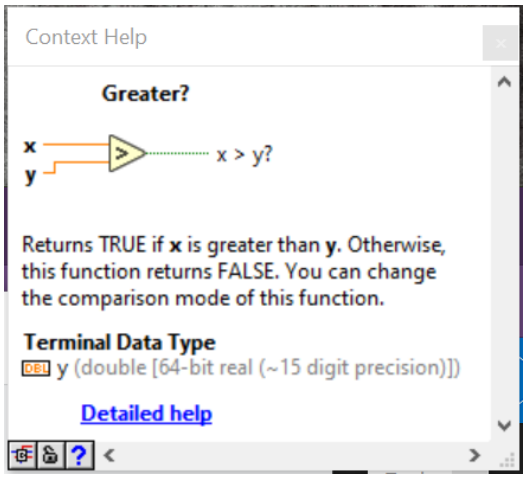


Figure 1.12: The context help for the comparison function, the top terminal is x and the bottom one is y.

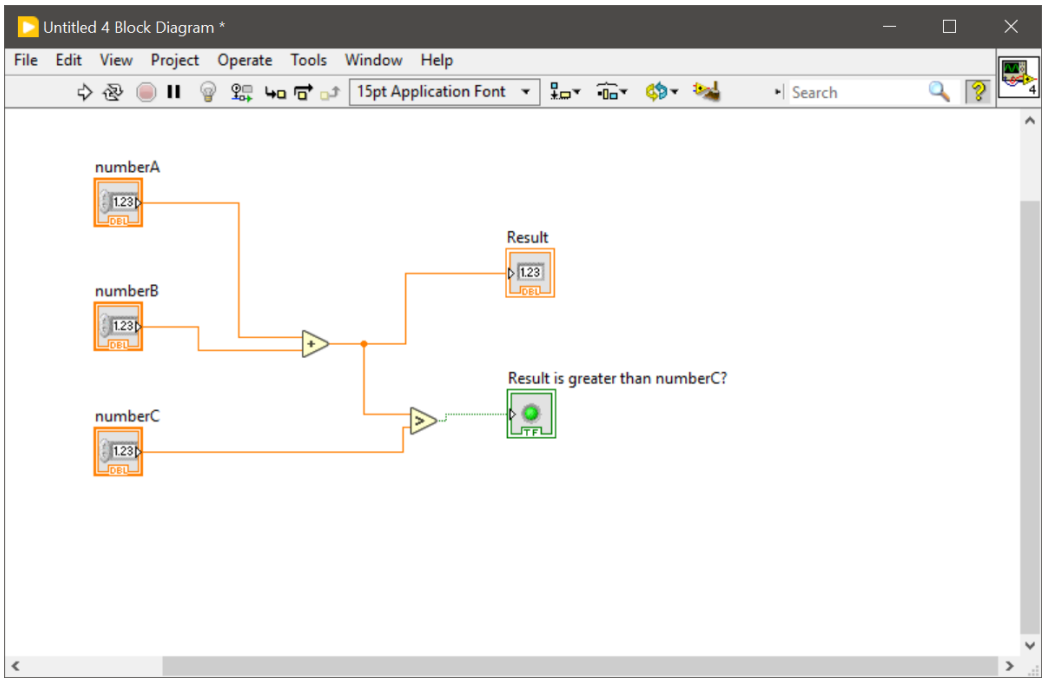


Figure 1.13: The final wired block diagram.

dicator. It is also possible to `Right click` on the wire leading from the summing function and selecting the "Create wire branch" option from the menu. You then wire this into the comparison function.

Finally, your block diagram should look like the one in figure 1.13. You may also move around the wire pieces by `left click`ing on them and dragging them about.

1.4.3 Testing your program

You may now go back to your front panel. You can click in the centre of the controls and type in a value. Do this for all three number controls and press the play button, the one found near the top left of your window.

Does your results make sense? Is the summing correct? Is the light on? Should it be on? These are all the questions that you should have when testing your program. Figure 1.14 shows what happens when the sum of numberA and numberB is greater than numberC. Figure 1.15 shows when numberC is smaller than the sum. If your program does the opposite of what

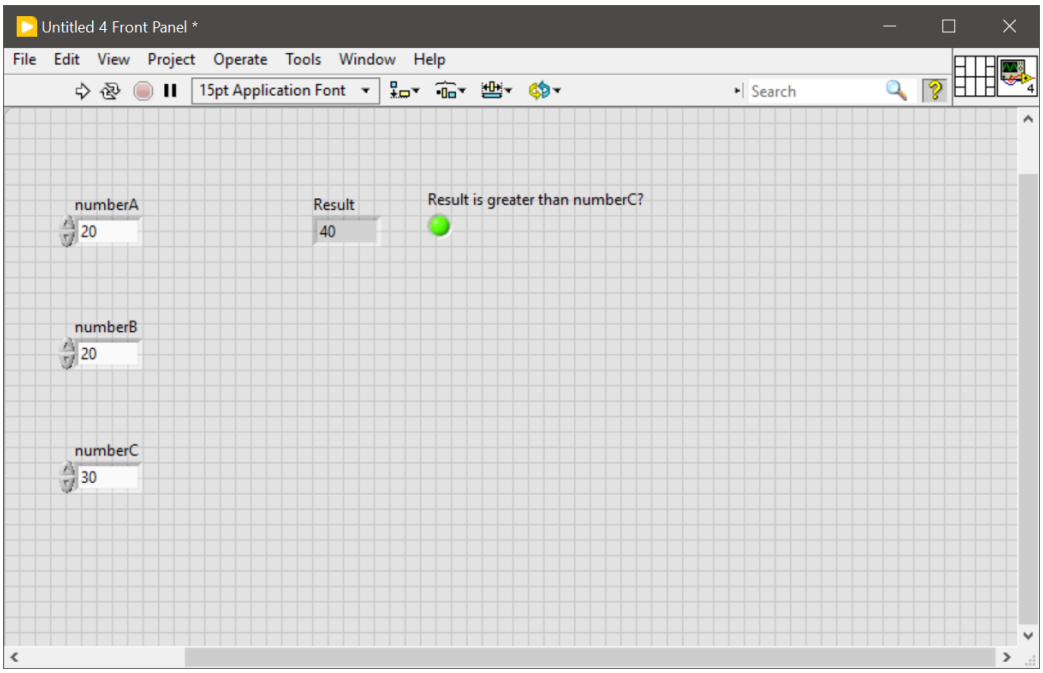


Figure 1.14: The sum is 40 which is greater than 30 so the LED is on.

it is supposed to do, make sure that you have wired in the correct terminals and have selected the correct comparison function. It is worth noting that the VI executes once and stops. After you have typed in new values, you will need to run the VI again.

You may now save your VI by clicking "File" on the task bar and selecting save from the menu. Name it something like "helloworld.vi".

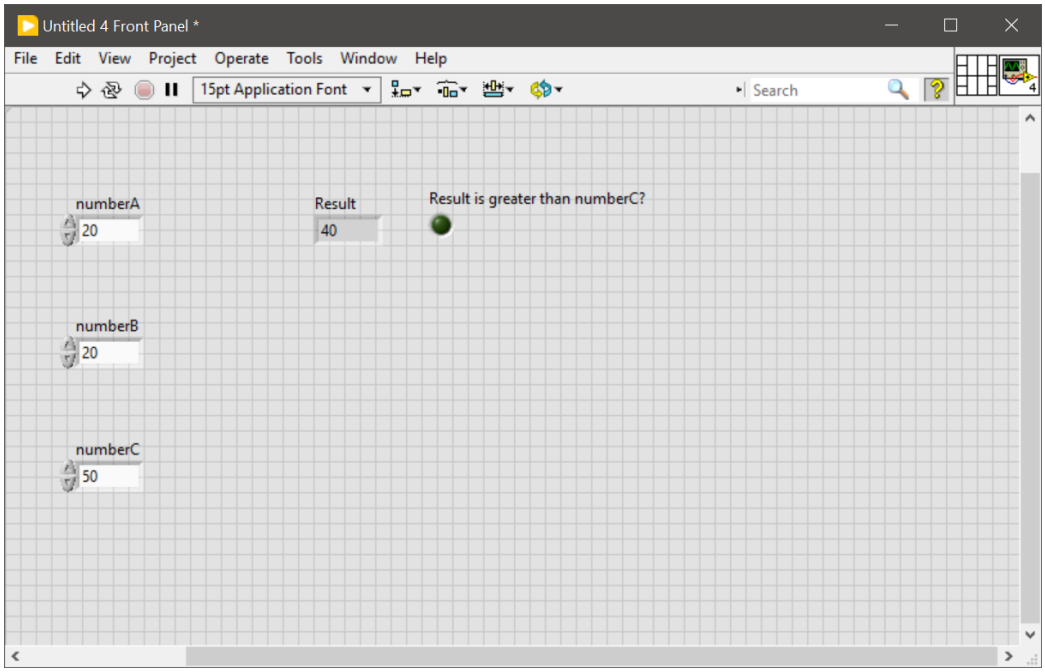


Figure 1.15: Changing numberC to 50 turns the LED off.

Chapter 2

The Second Chapter