Intro to LabVIEW

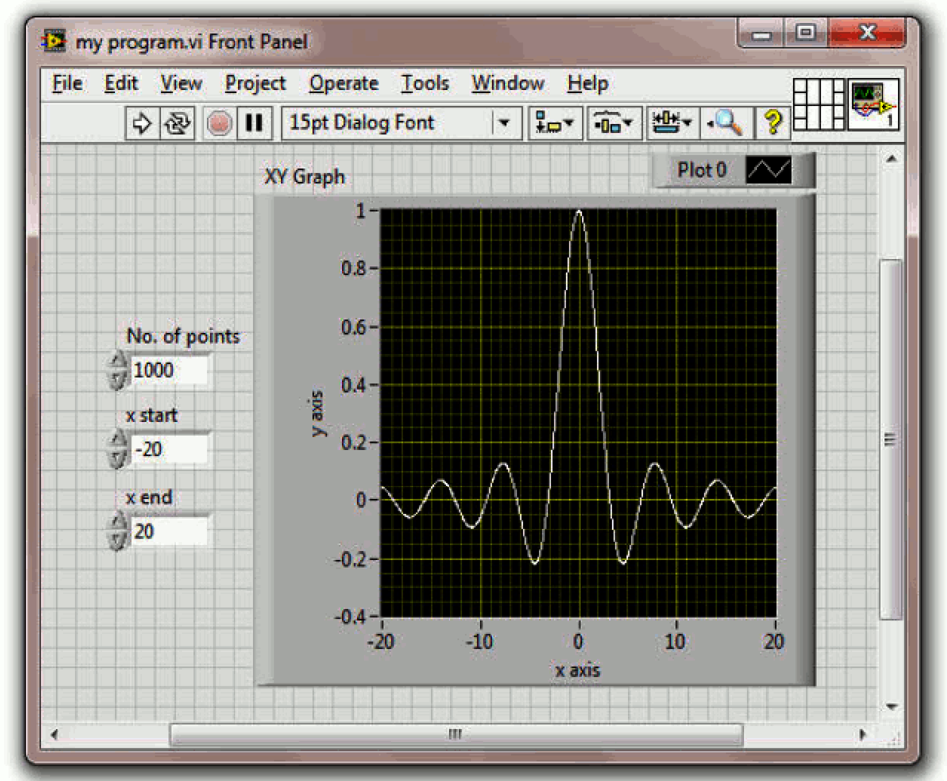
(Materials developed by Christophe Salzmann at École polytechnique fédérale de Lausanne EPFL)

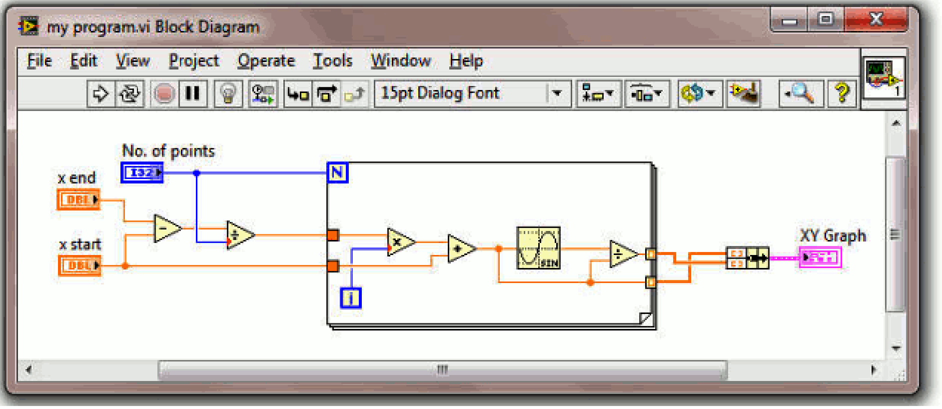
LabVIEW ?

- LabVIEW is a graphical programming environment, it is targeted mainly at measurements and control, but not exclusively
- LabVIEW runs on wide range of hardware, from FPGA to multi-core CPUs
- LabVIEW is programmed (mainly) in G a graphical language
- LabVIEW is the acronym for:
  Laboratory **V**irtual **I**nstrumentation **E**ngineering **W**orkbench

Why use LabVIEW ?

You will be a lot more efficient

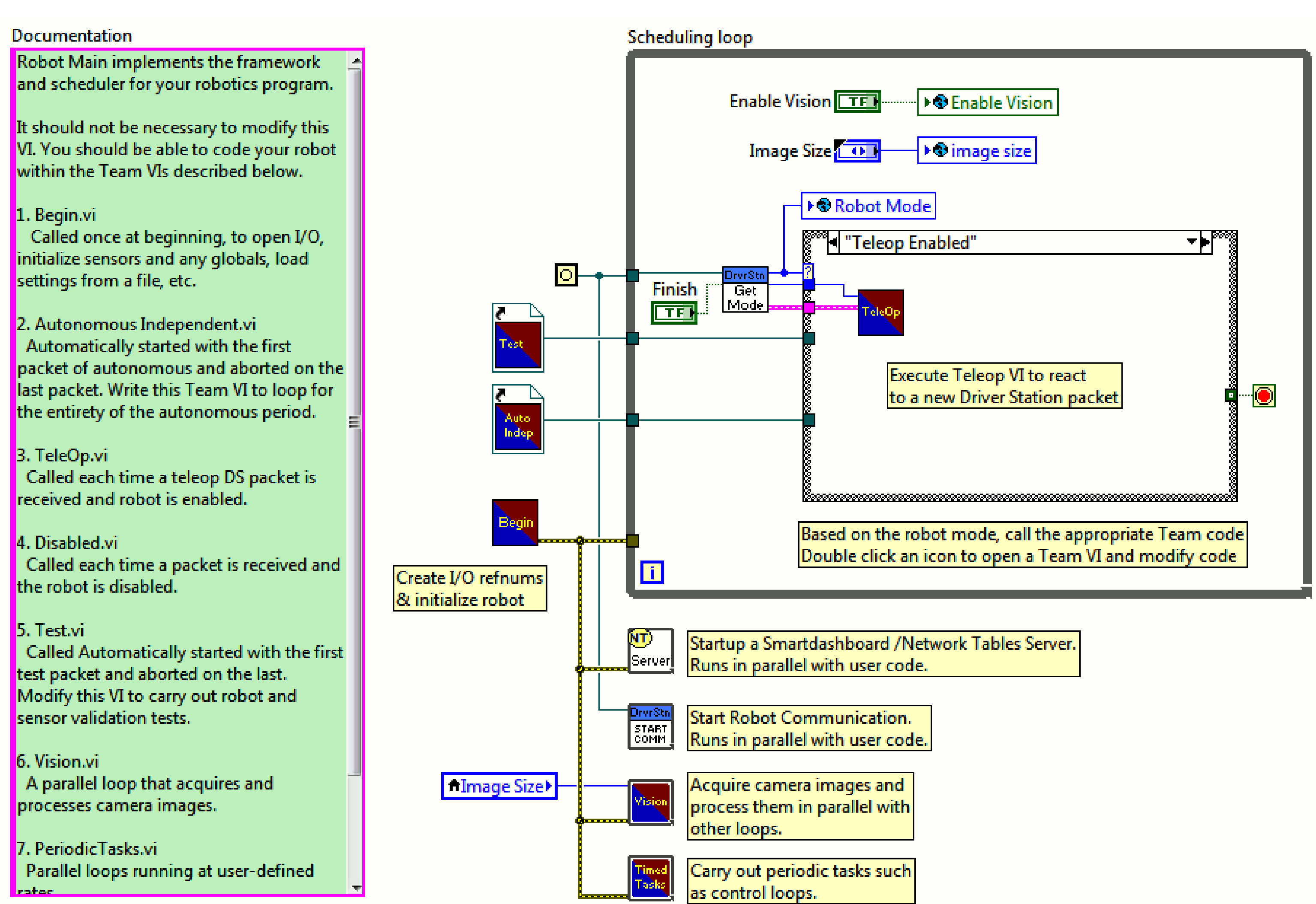




LabVIEW large examples

- CERN LHC
- SALT telescope
- Honeywell
- LA remote lab
- First Robotics

…

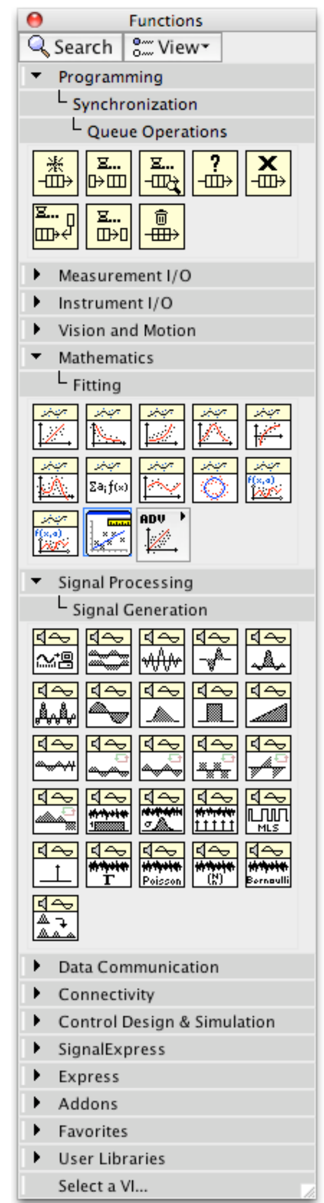


Why LabVIEW ?

- LabVIEW has hundreds of built-in functions (engineering + scientific)
- And a lot of additional libraries

LabVIEW Application Builder
LabVIEW Real-Time Module
LabVIEW FPGA Module
LabVIEW PDA Module
LabVIEW Touch Panel Module
NI Vision Development Module
NI SoftMotion Development Module for LabVIEW
Express VI Development Toolkit
LabVIEW VI Analyzer Toolkit
Report Generation Toolkit for Microsoft Office
Internet Toolkit
LabVIEW Real-Time Module
LabVIEW FPGA Module
LabVIEW Statechart Module
LabVIEW Microprocessor SDK
NI LabVIEW Embedded Module for ADI Blackfin Processors
LabVIEW DSP Module
LabVIEW Real-Time Execution Trace Toolkit
LabVIEW PID Control Toolkit
LabVIEW DSP Test Integration Toolkit for TI DSP

Digital Filter Design Toolkit
Advanced Signal Processing Toolkit
Math Interface Toolkit
Modulation Toolkit
Order Analysis in LabVIEW
NI Sound and Vibration Analysis Software
Spectral Measurements Toolkit
NI Vision Builder for Automated Inspection
NI Motion Assistant
Database Connectivity Toolkit
NI Modulation Toolkit
NI Requirements Gateway
LabVIEW Control Design and Simulation Module
LabVIEW Simulation Interface Toolkit
LabVIEW System Identification Toolkit
LabVIEW PID Control Toolkit
LabVIEW Datalogging and Supervisory Control Module
LabVIEW Statechart Module
Vision Development Module
NI SoftMotion Development Module for LabVIEW

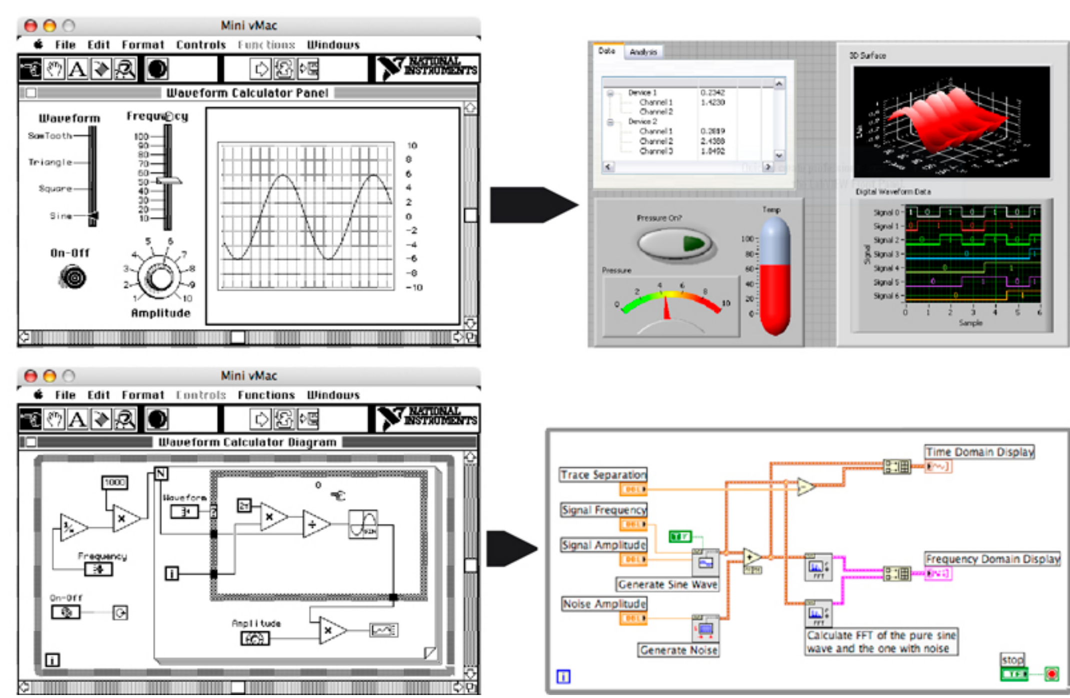


LabVIEW history (short)

1986 - LabVIEW 1

..

2014 - LabVIEW 2014



Same concepts among versions, if you know LabVIEW 1 you know LabVIEW X and vice-versa

LabVIEW

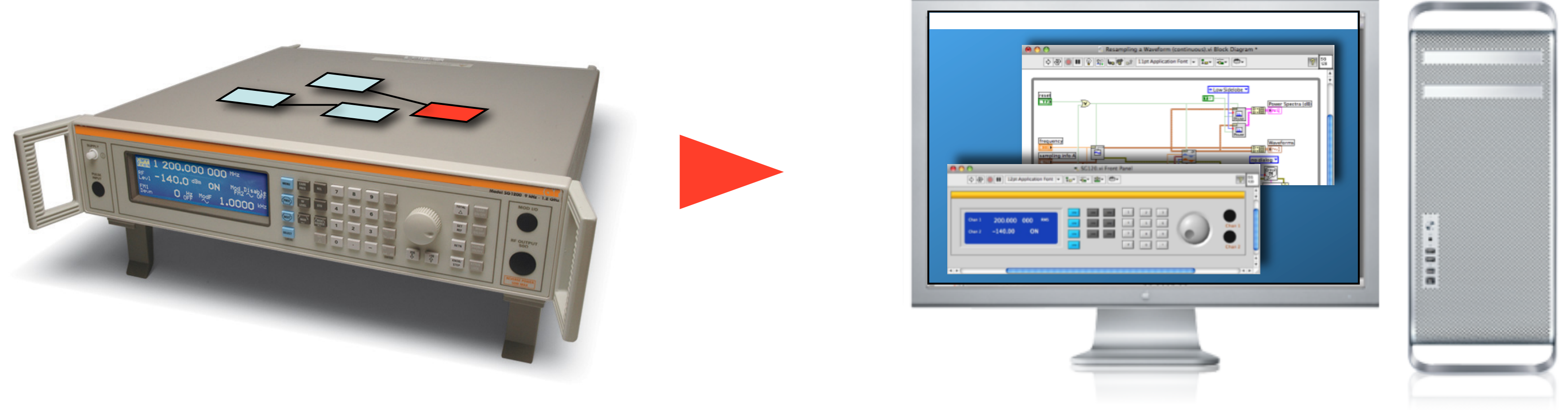
2 main concepts:

Virtual instrument

Data flow programming

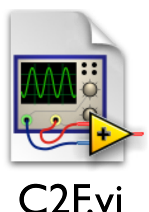
Virtual instrument

- Mimic real-life instrument



Front panel - diagram - connector pane

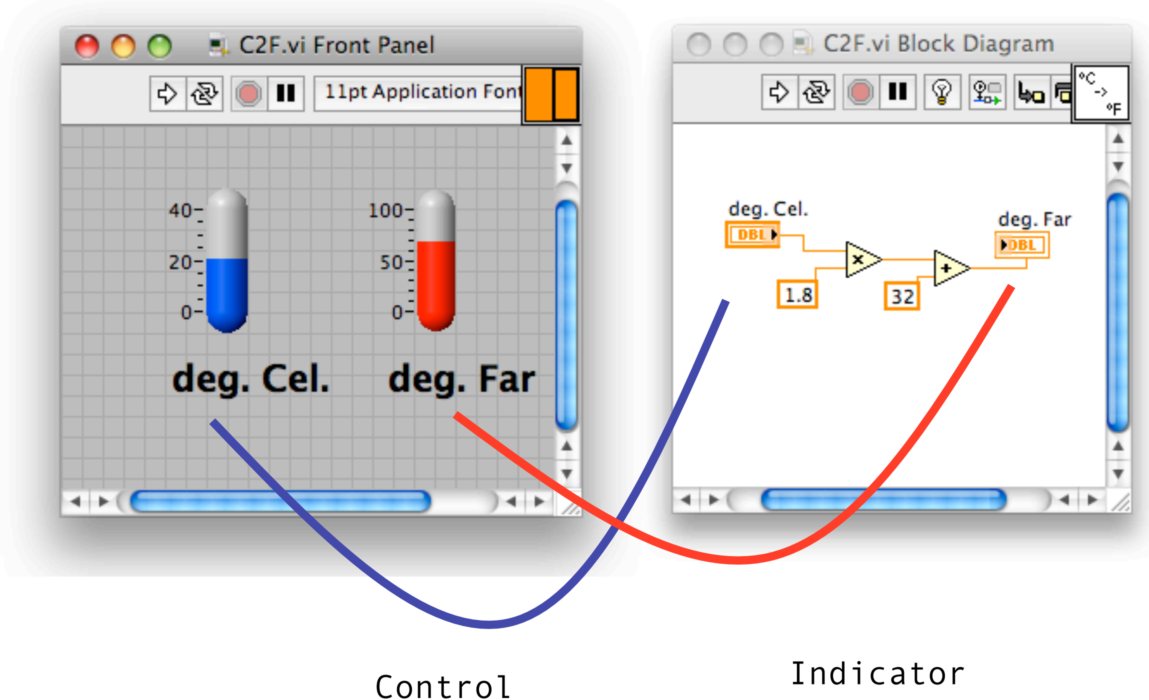
Virtual instrument (VI)



Front panel

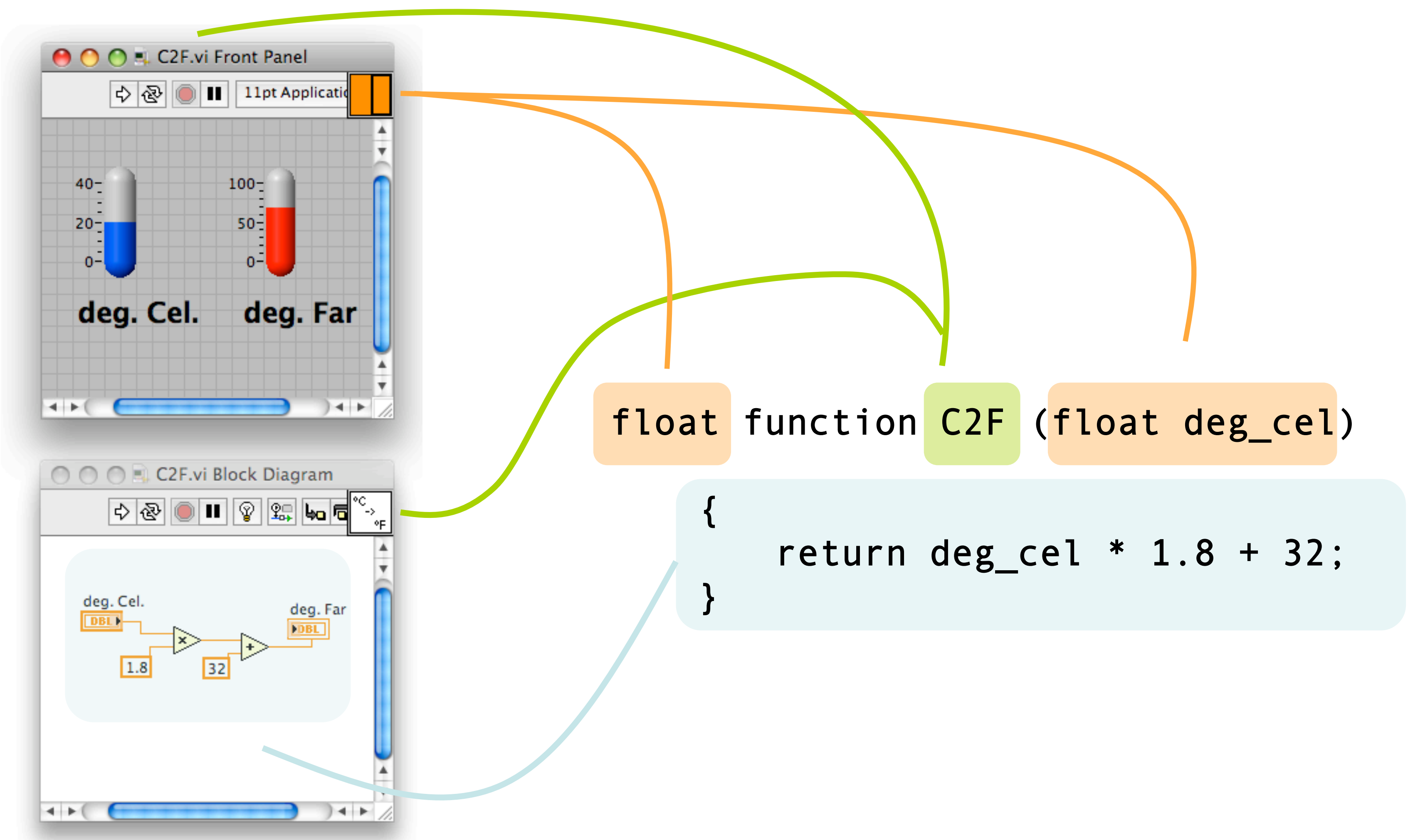
Connection Pane

Diagram

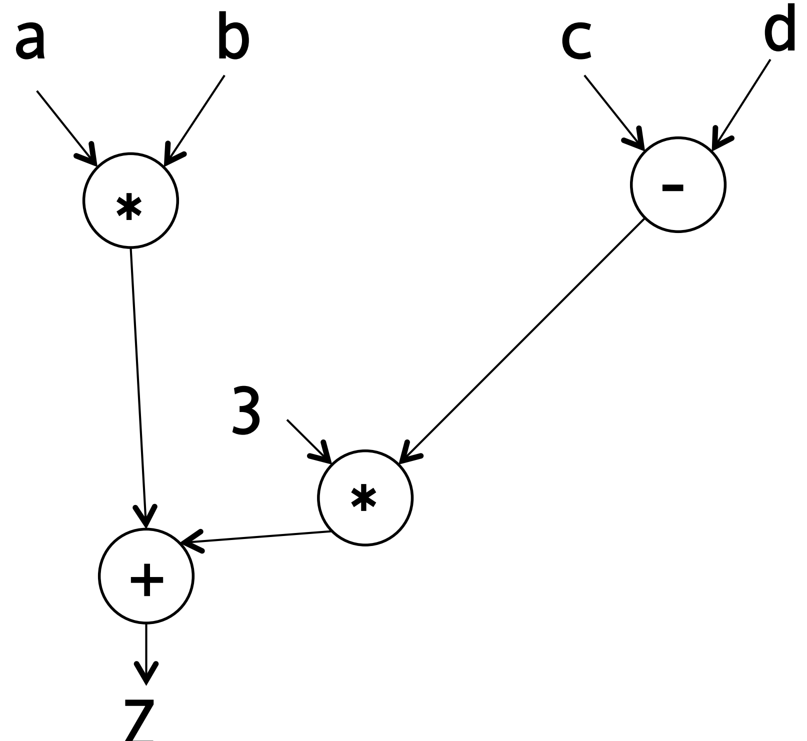


LabVIEW is programmed by dropping nodes and linking them with wires

C equivalent



Data flow programming



The flow of data controls the program execution.

It's like small rivers coming together to form bigger rivers, then split to form other rivers.

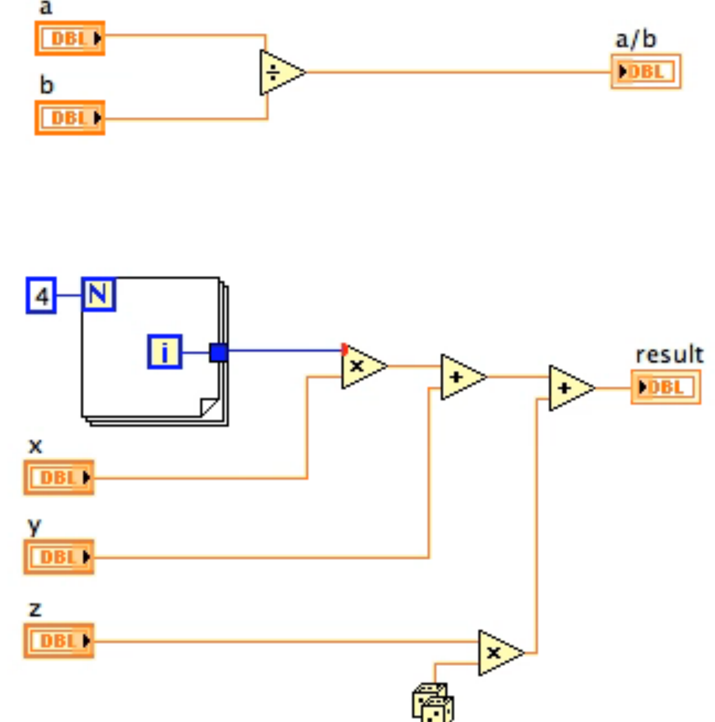
The data that flows is like the water, or the electricity in a circuit.

A node is executed only when all its inputs are know.

```
Z = (a * b) + 3 * (c - d)
```

Data flow programming

The flow of data controls the program execution
Parts of the diagram may run in parallel

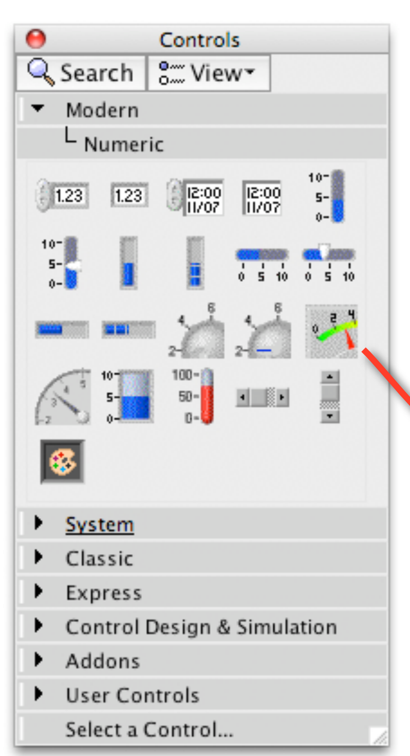


LabVIEW environment

- LabVIEW IDE provides all the needed tools
  - **GUI builder – front panel**
  - **G code editor - diagram**
  - *Debugger*
  - *Project manager*
  - *Wizards*
  - Code structure generator (state diagram, OOP)
  - Compiler, cross-compiler
  - Code analysis/metrics
  - Code coverage
  - Source versioning, diff tools
  - etc.

… and a lot of examples

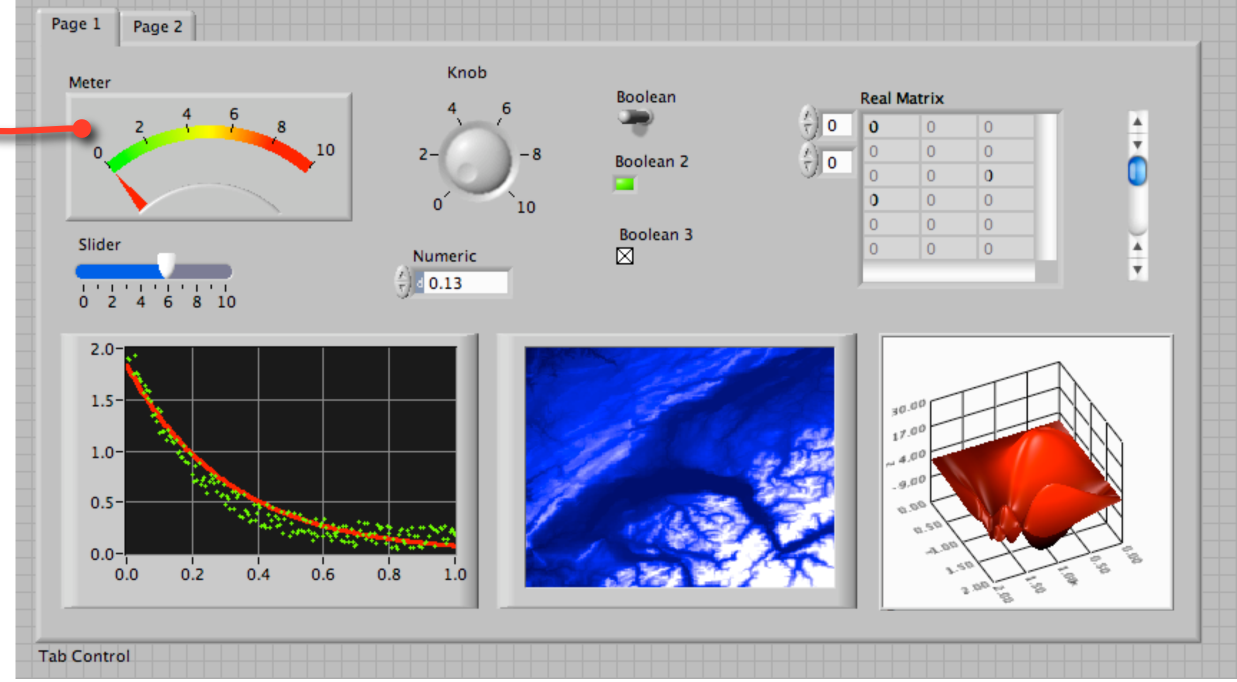
GUI Builder - Front panel



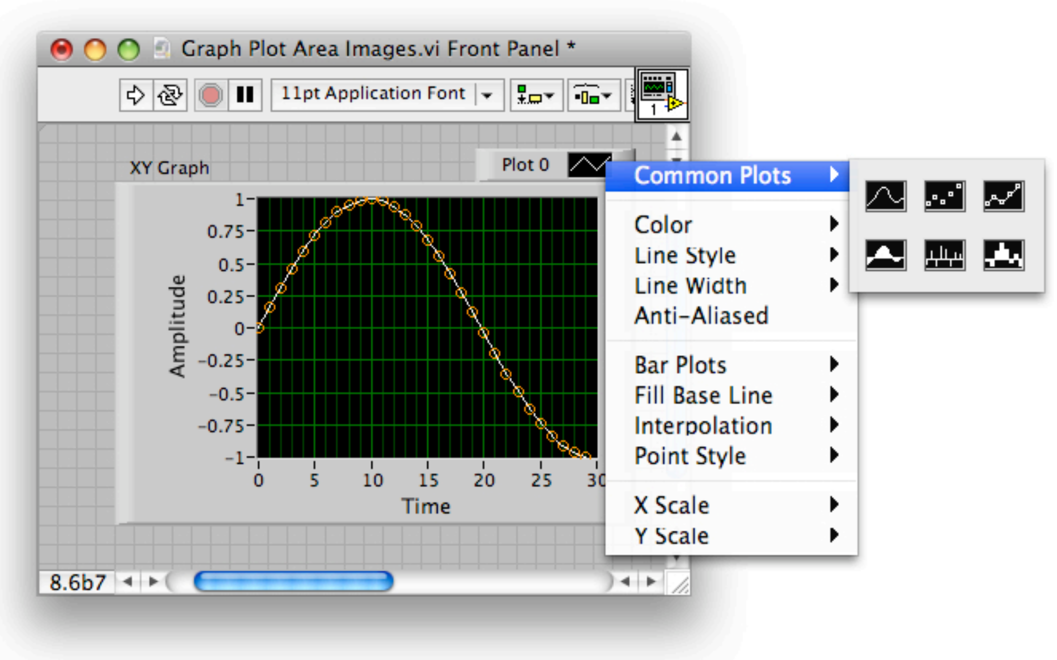
Draw the GUI as one would with another drawing program (illustrator, powerpoint, etc.)

Controls and indicators are accessible via the **Controls** palette

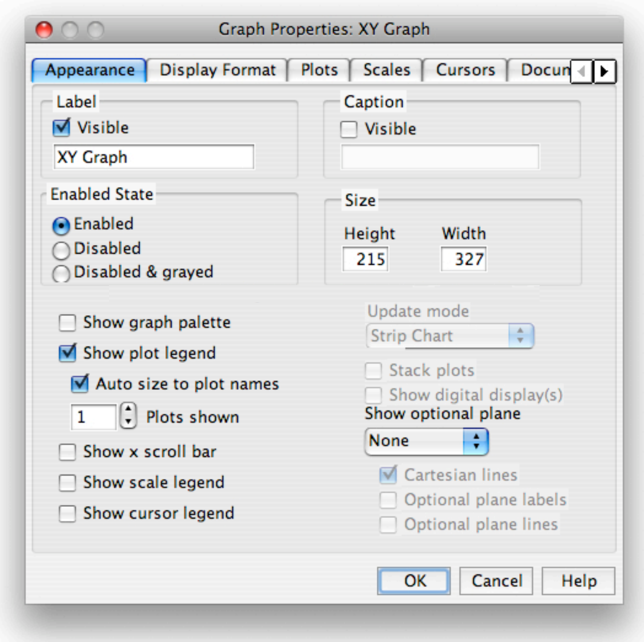
Use right click or drag&drop to place control/indicator



GUI builder - edit object



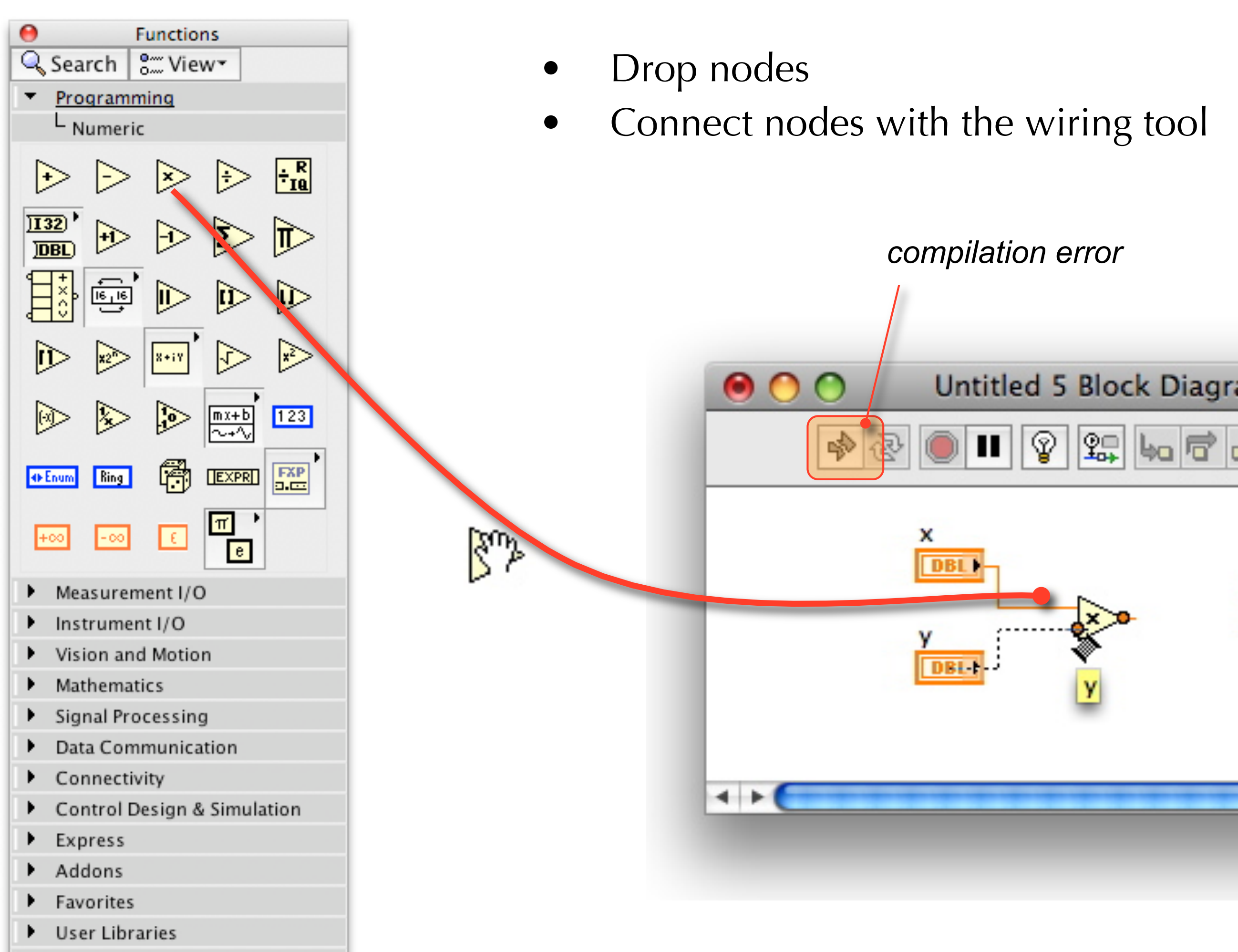
Right-click on the given part and select the desired option



Edit all properties at once via a dialog

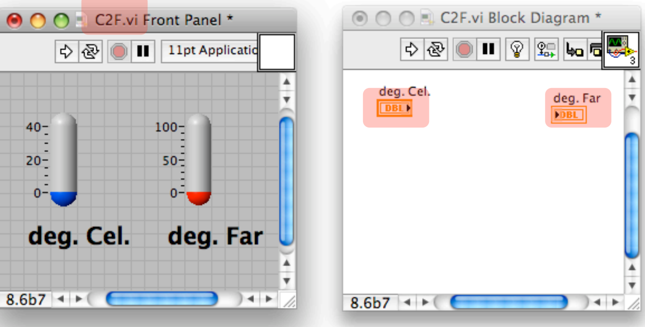
G - editor

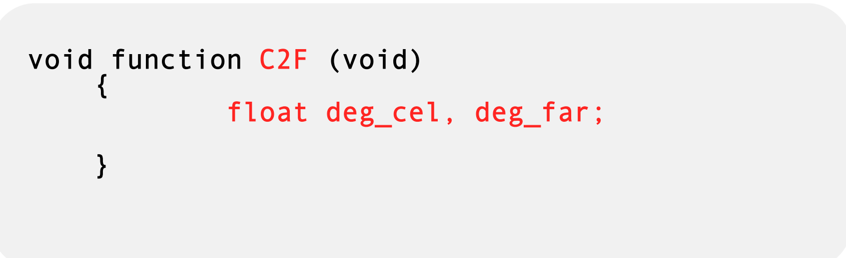
- Drop nodes
- Connect nodes with the wiring tool



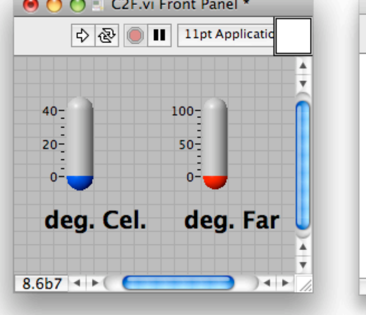
G editor - C equivalent

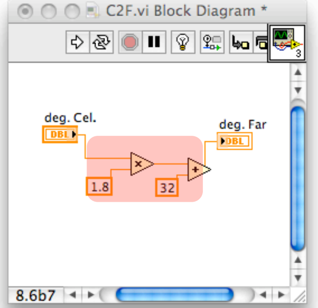
Name

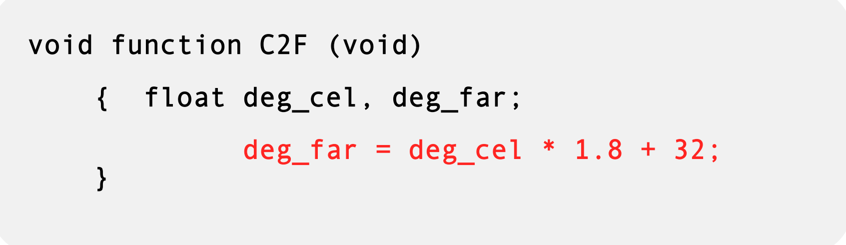




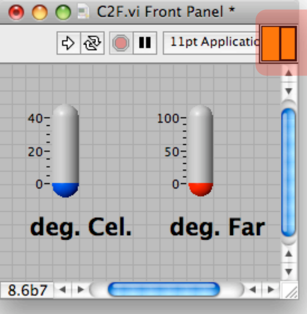
Code







Params.

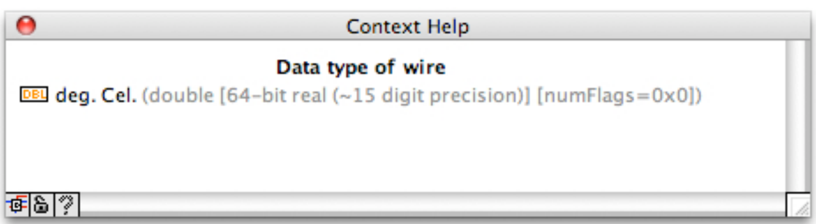


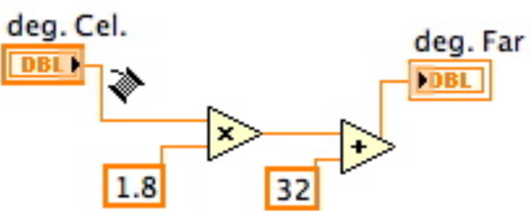

```
float function C2F (float deg_cel)
    {
            return deg_cel * 1.8 + 32;
    }
```

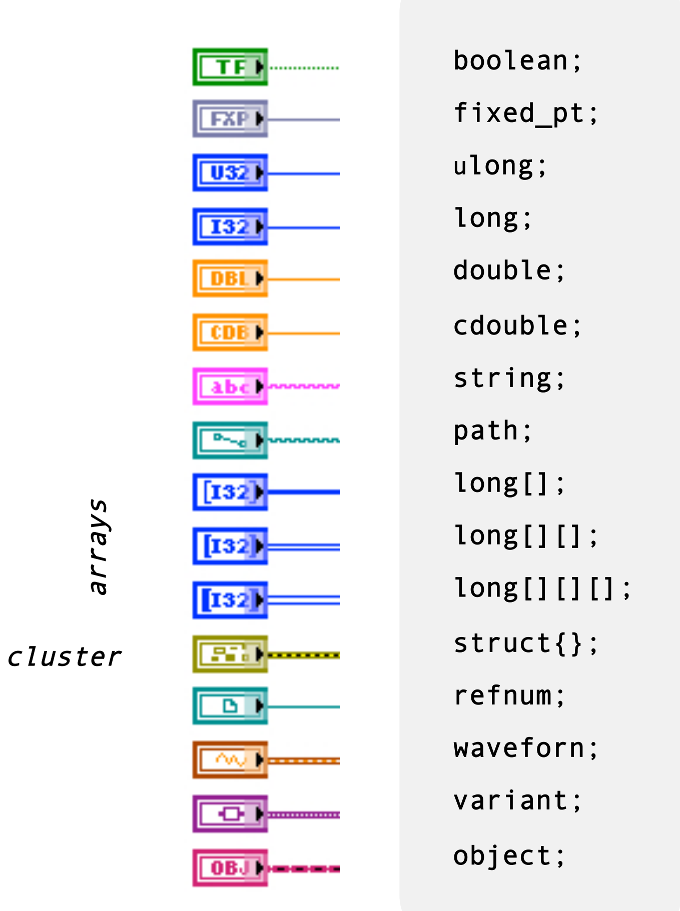
G data types

G data types

- G is strongly typed
- Colors and sizes define data types
- the wiring tool inform about types & unit

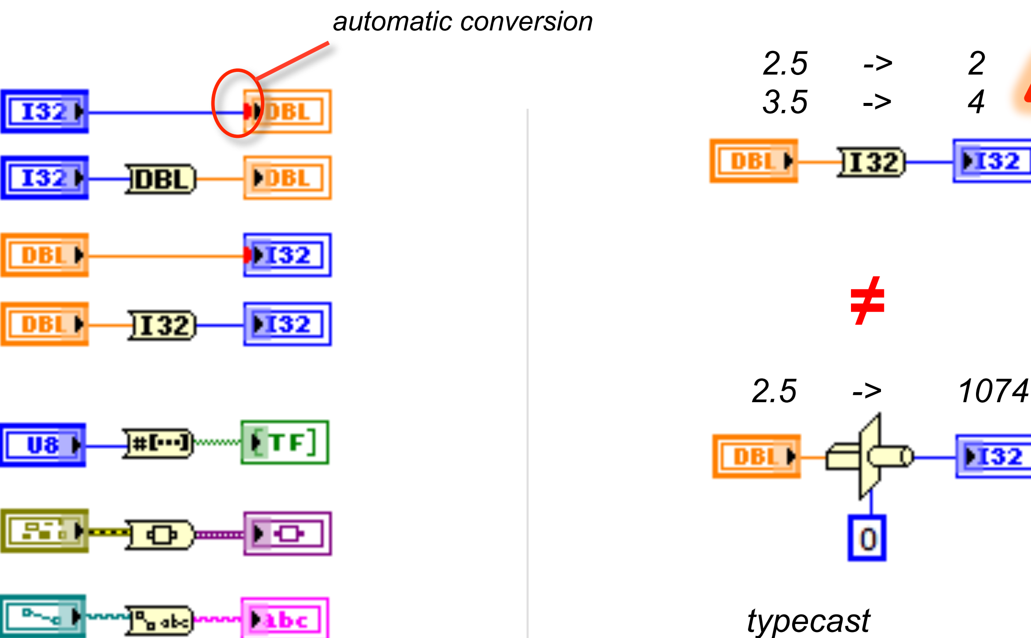






G data types

- Automatic type conversion may occur, red dot
- Type conversion can be explicit, follows IEEE 754
- ⚠ type conversion ≠ typecast



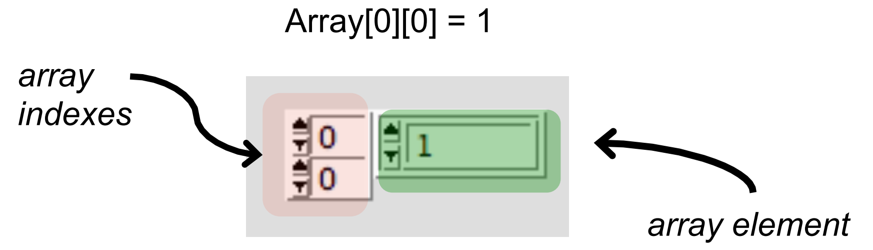

```
out = RoundtoInt (in);
```

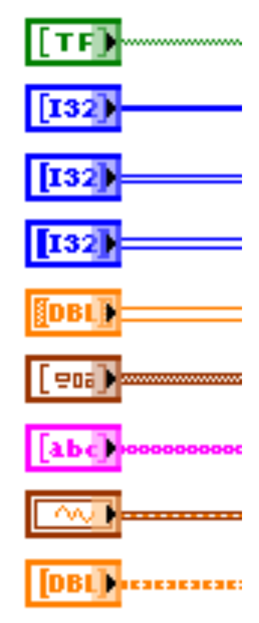
```
double in;
out = *(int32*)&in;
```

G data types - array

Array -> structure with all elements having the same type

- Arrays can be of any dimension (up to 64)
- Arrays can be of any types
- Array are dynamically allocated & expended
- Width of ‘[]’ visually informs about dimension
- Many array primitives exists, even more linear algebra functions
- Waveforms are specific 1D array type
- Matrix (real or complex) are specific 2D array types



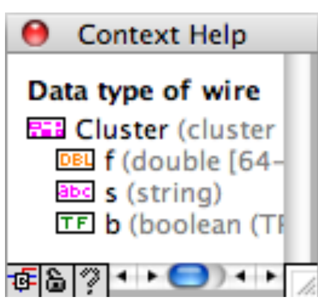


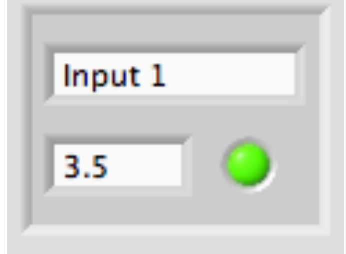

```
Bool[];
long[];
long[][];
long[][][];
float[][][][][]..
struct[];
Str[];
waveform;
matrix;
```

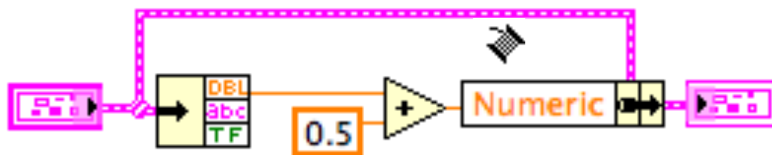
G data types - cluster

Cluster-> structure with element of mixed types

- Similar to de-multiplexer
- Cluster size (# elements) is fixed
- Cluster color indicates if its elements are of fixed size (brown) or not (pink)
- Cluster can be nested
- Cluster element can be accessed by position or name (recommended)







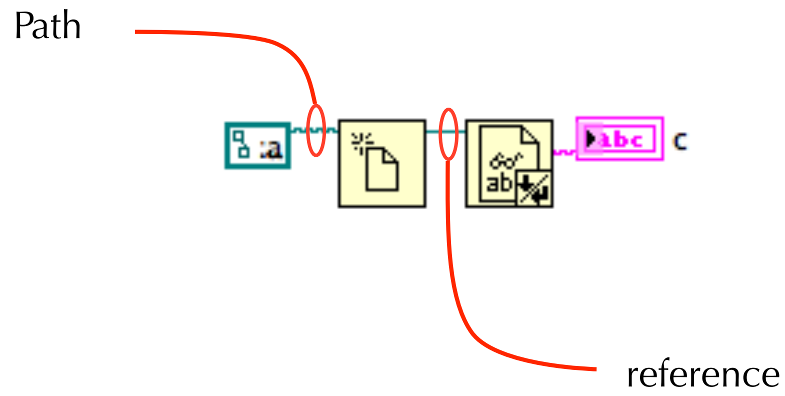

```
struct {
    double f;
    char[] s;
    bool   b;
} c;
```

```
c.f = c.f + 0.5;
```

G data types – reference

Are like pointers/references in c++

- Store a reference to LabVIEW data and objects
- Green wire, cannot be "viewed"
- Used for file, network, etc. access
- Front panel and diagram elements can be referenced

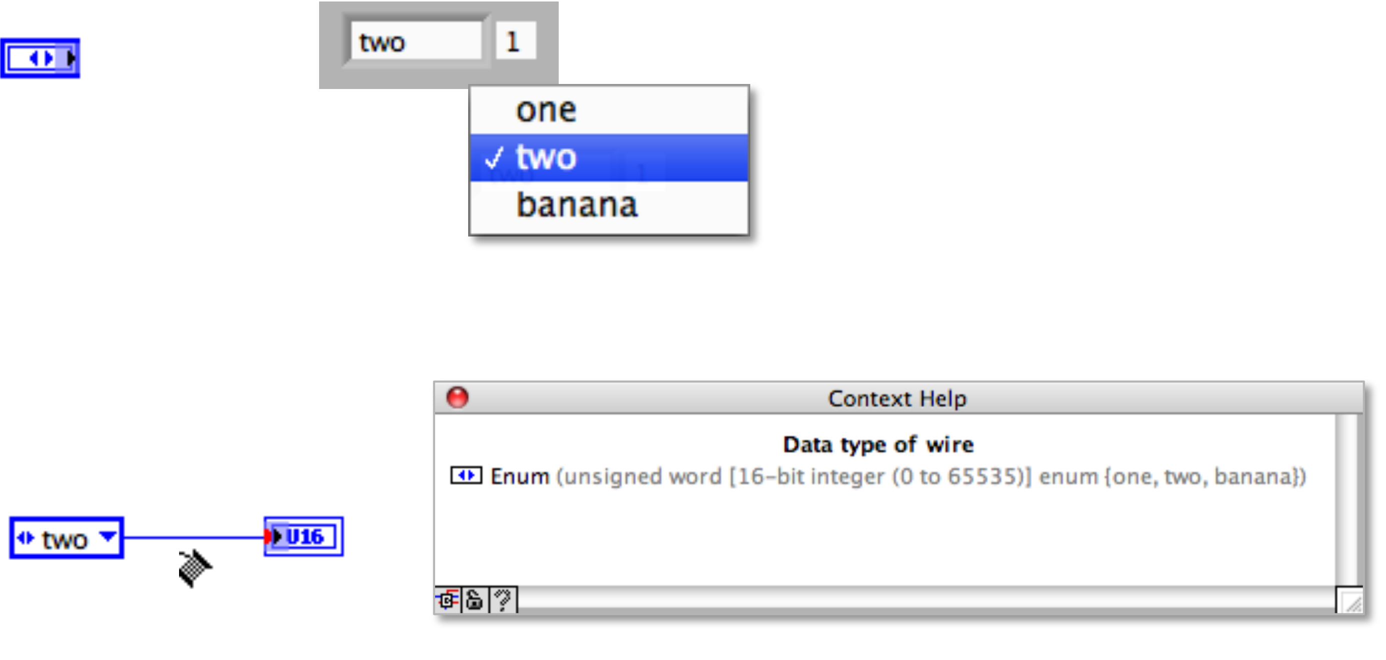



```
FILE       *f;
char[255]  c;

f= fopen("a");
e= fread(f,c);
```

G data types - enumeration

Enumeration

- set of names represented by an integer value



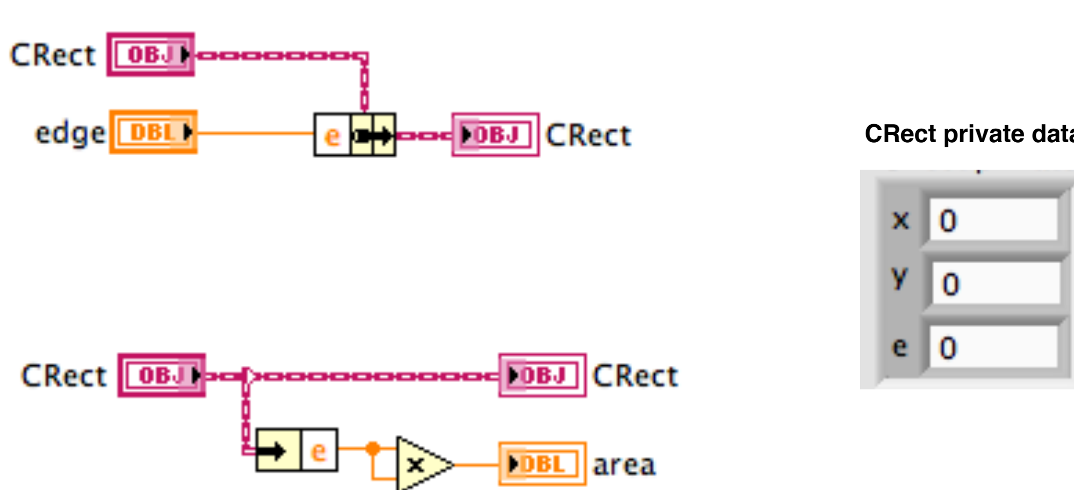
U16 = 1

```
enum {
  one,
  two,
  banana
}

where

  one is 0
  two is 1
  banana is 2
```

G data types – object

As in Object Oriented Programming

- Different implementations (NI, others)

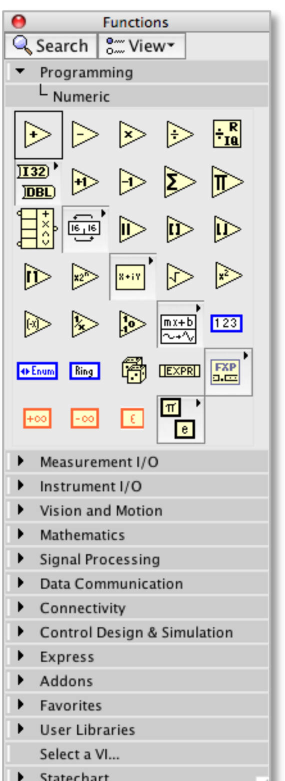


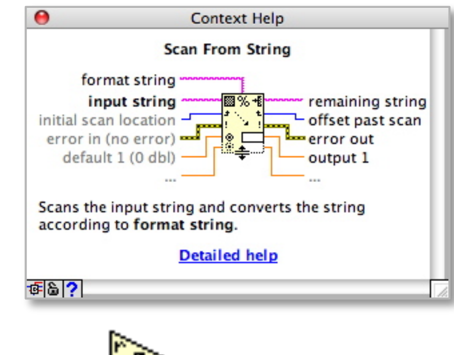

```
class CRectangle {
 int x, y, e;

public:
void set_center
     (int,int,int);
void set_edge(int);
int area (void);
} rect;

rect.set_edge(2);
int a = rect.area();
```

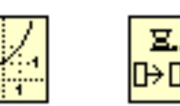
G-functions

G-functions

- Primitive functions have a yellow background
- They cannot be edited
- Hundreds of them grouped by data type in palettes
- Many functions are polymorphic







```
+
and
>=
r*e^(i*theta)
exp
enqueue
find in array
find str
millisecond
show dialog
read text file
parse string for float
```

G-polymorphism

- Polymorphism: same code for different data types, functions supporting more than one data types

  int32, double, bool, 2D array, ..
- Many primitives are polymorphic
- LabVIEW may automatically convert data type (red dot)

T   TF   F
F   TF

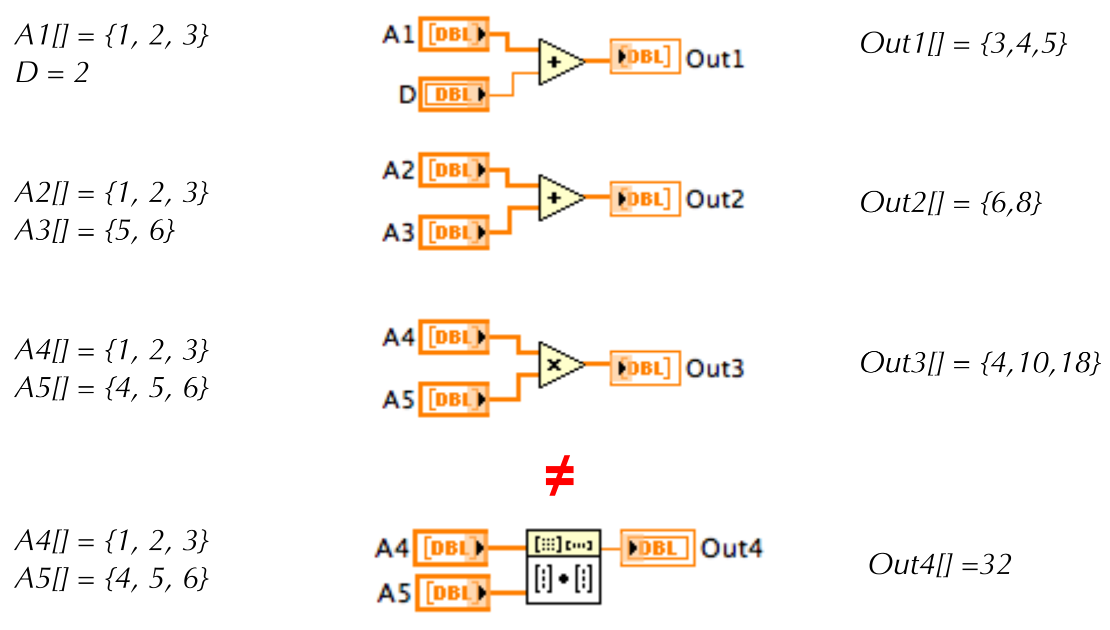
0100_b 4   I32   0
0011_b 3   I32

RoundtoInt(3.5) -> 4 0100_b 3.5   DBL   4
0111_b 7   I32

```
Bout = B1 AND B2

Lout = L1 AND L2
    // bit by bit op.

Lout = RoundtoInt(F1)
AND L2
```

G-polymorphism

- Array polymorphism
- ⚠ operations are performed element by element, not vector/matrix




```
for (i=0;i<Size(A1);i++)
    Out1 = A1[i] + D;

n=min(size(A2),size(A3));
for (i=0;i<n;i++)
    Out2 = A2[i] + A3[i];

n=min(size(A2),size(A3));
for (i=0;i<n;i++)
    Out2 = A2[i] * A3[i];

Out4 = Dot(A4,A5);
```

G - structures

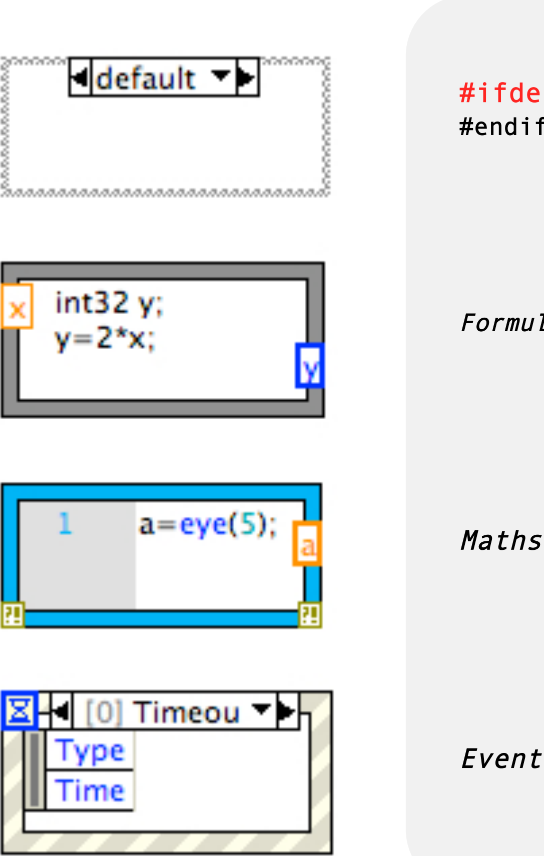
G - structures

```
for (i=0;i<N;i++) {
}
```

```
i:=0;
do {
} while (cond; i++)
```

```
switch(cond) {
   case:
     break;
   default
}
```

```
Sequence1;
```




```
#ifdef cond
#endif
```

Formula node

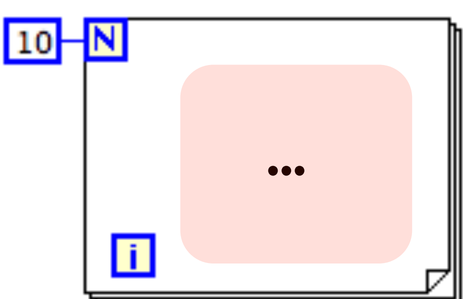
Mathscript node

Event node

Loops - for

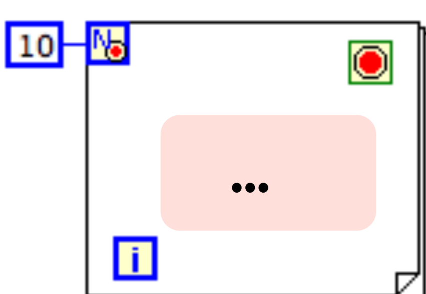
Execute its content a fixed number of time

- [i] goes from 0 to 9 ([N] - 1)



Execution can be stopped (LV 8.6)

- Stops if [stop] is true

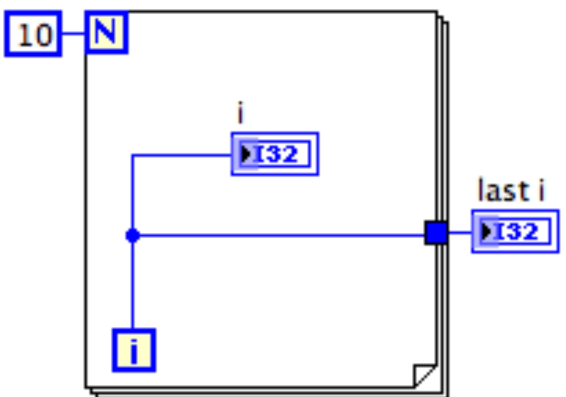



```
for (i=0;i<N;i++)
   {
     …
   }
```

```
for (i=0;i<N;i++)
   {
     …
     if () break;
   }
```

Loops - for

Execute its content a fixed number of time



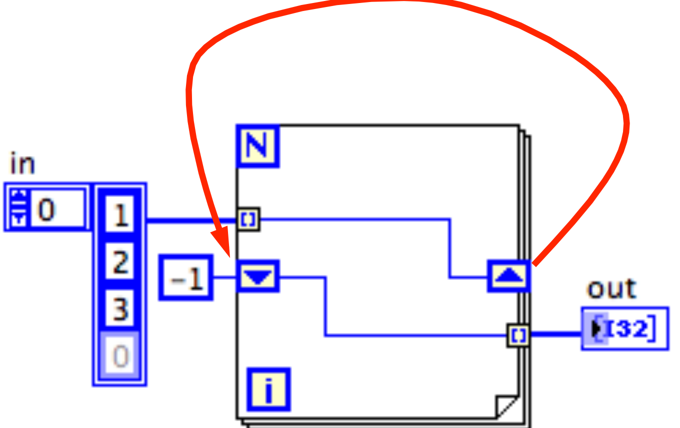
i goes from 0 to 9
last_i = 9

```
N=10;
for (i=0;i<N;i++)
   {
     i;
   }
last_i = i;
```

Loops - for

Shift registers

- sets the value for iteration *i*
- retrieves the value of the iteration *i*-1



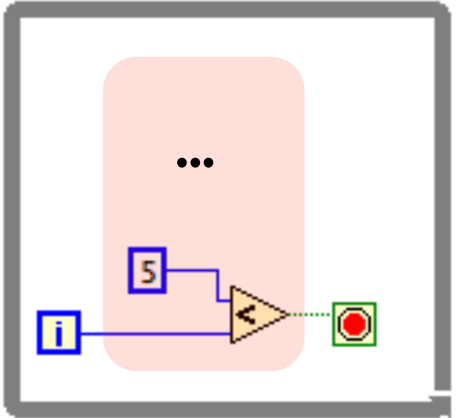
- out[] = {-1, 1, 2}

```
for(i=0;
     i<sizeof(in[]);
       i++)
 {
   if (i==0)
     out[i] = -1;
   else
     out[i] = in[i-1];
   }
```

Loops - while

Execute its content until a condition is reached

- The while loop is executed 7 time
- i goes from 0 to 6

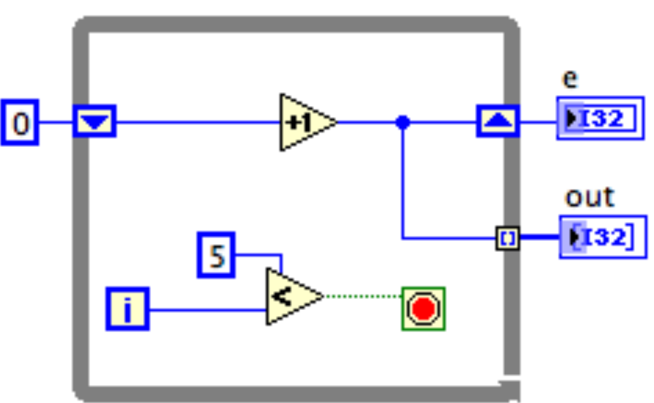



```
i = 0;
do {
    …
  }
  while(5 < i; i++);
```

Loops - while

Execute its content until a condition is reached

- The while loop is executed 7 time
- 5 < i is true when i = 6
- i goes from 0 to 6



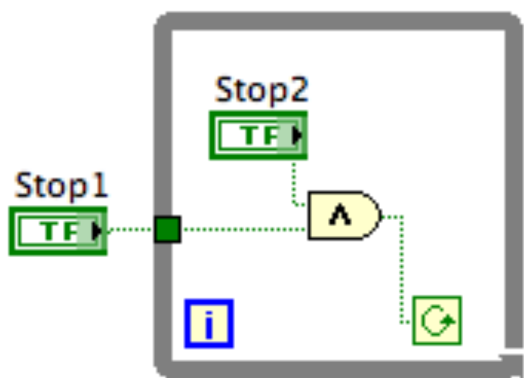
- e=7
- out[] = {1,2,3,4,5,6,7}

```
i  = 0;
do {
    if (i==0)
      sr[i] = 0;
    else
      sr[i]=sr[i-1]+1;
    out[i]=sr[i];
    }
  while(5 < i; i++);
e = sr;
```

Loops - while

Wires are evaluated at the loop borders

- “Stop1” is evaluated once, before entering the loop
- “Stop2” is evaluated at each loop iteration

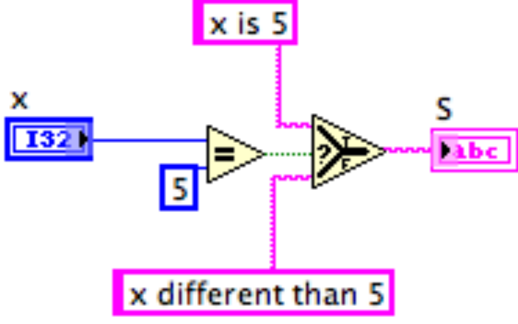


- *The loop will exit after one iteration is Stop1 is False*
- *If Stop1 is True, the loop will exit when Stop2 is False*

```
tmp = Stop1;
do {
      ...
   }
  while( tmp & Stop2 );
```

Conditional - if

- If the condition is True
    pass the value connected to T
  otherwise
    pass the F value

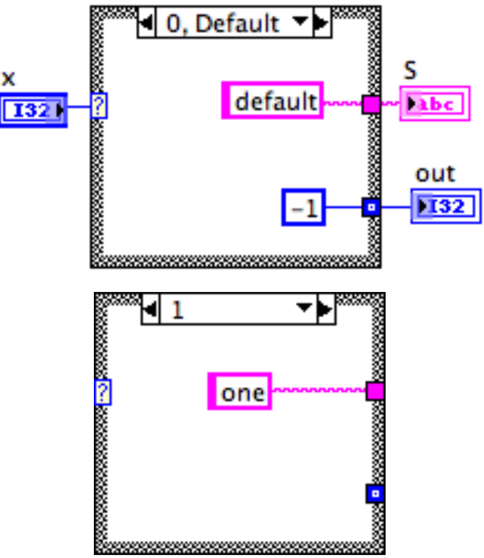


- if x = 5 -> S = “x is 5”; S =“x different than 5” otherwise

```
if (x==5)
     S = “x is 5”;
   else
     S = “x different
                 than 5”;
```

Conditional - case

- Case structure is similar to *switch* statement
- ■ indicates that all cases are defined
- ■ indicates that if a case is not wired use the default value

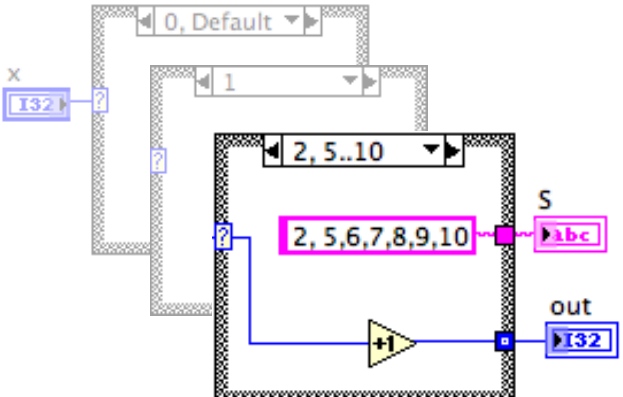


- *S= "one"* when *x==1*, and *S="default"* for all other values of x
- *out* = *-1* or *0*

```
out = -1; //default val.
switch (x)
    case 1:
      S="one";
      break;
    case 0:
    default:
      S = "default";
      out = -1;
```

Conditional - case

- Case type [?] will adapt to the source format (typecast may occur)
- Cases can contains range of values with ".." or separated with ","
- Specific code can be executed in given case
- Case structures can be nested

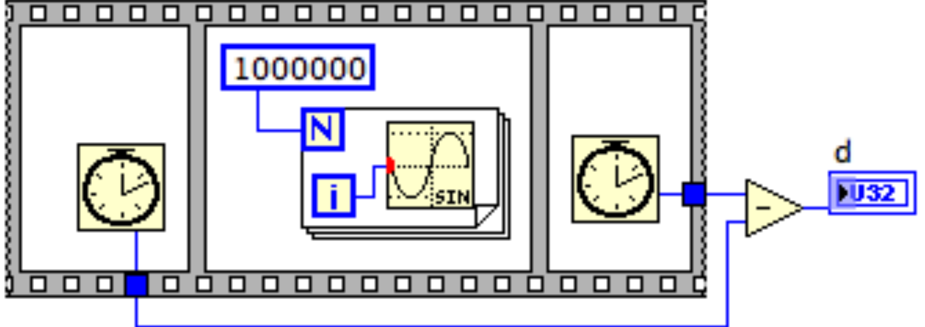


- when *x is in [2,5,6,7,8,9,10]*
  S= "2,5,6,7,8,9,10"
  out = x+1

```
out = -1; //default val.
switch (x)
      ..<skipped>..
      case 2:
      case 5..10:
        S = "2,5,6,7,8, \
                9,10" ";
        out = x+1;
        break
```

Sequence

- Force LabVIEW to execute code in a given order
- Should be avoided
- Main use: measure execution time




```
t1=millisec();
for (i=0;i<1000000;i++)
     sin(i)
t2=millisec();

d= t2 - t1;
```

- for the above case on a MacPro, *d = 4 [ms]*

My first VI

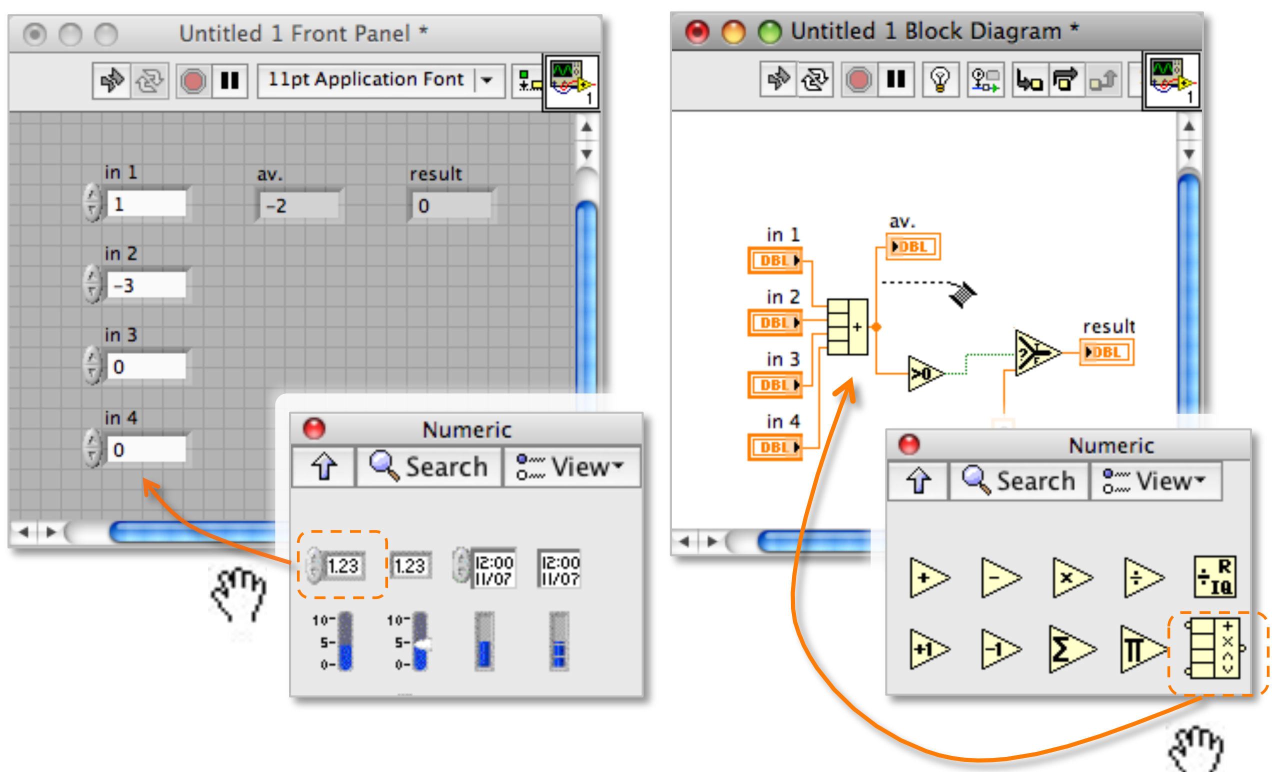
Once the VI's functionality is defined

- Design the GUI (controls & indicators)
- Wire the program (diagram)
- Test and Debug
- Add the documentation to the VI
- Define the interface (connector pane)
- Draw the Icon

My first VI – average

Specifications:

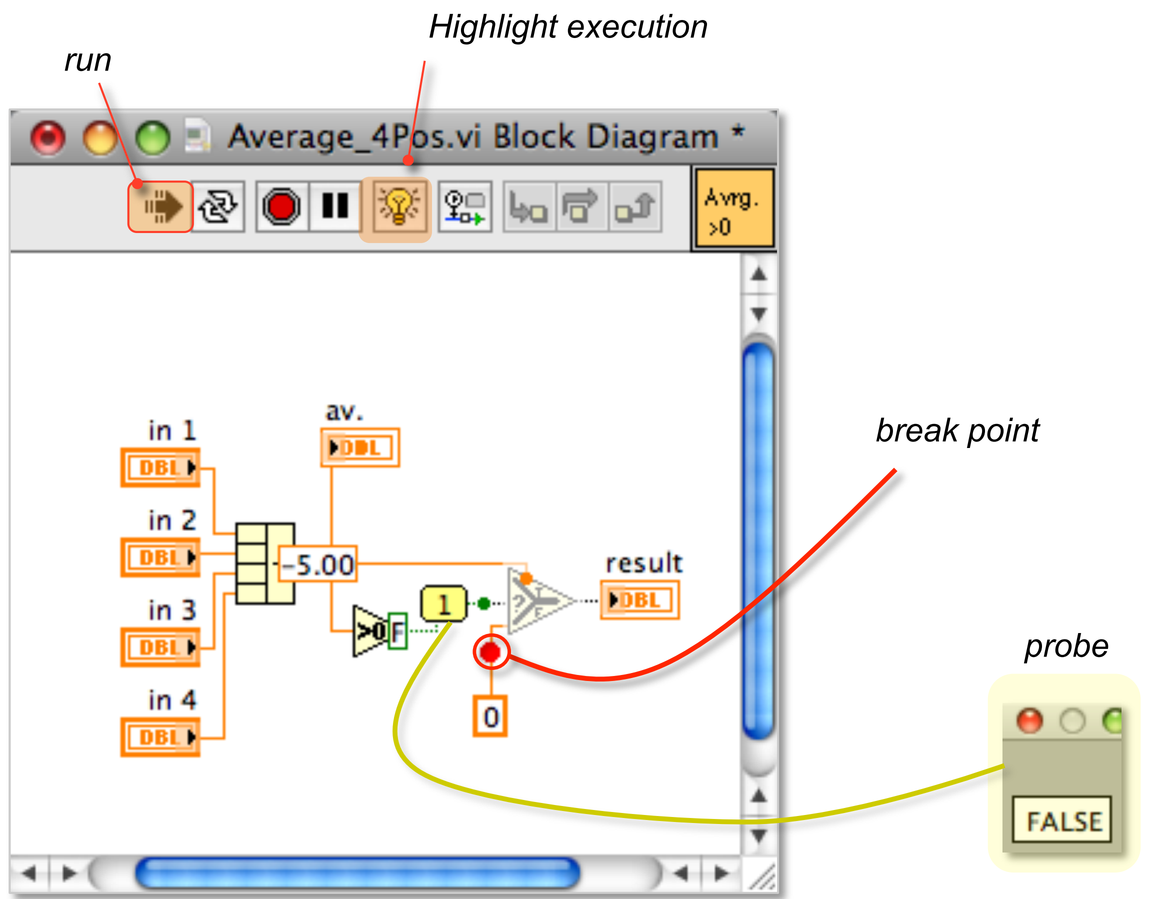
- Compute the average of the 4 input values
- If the result is less than 0 set it to 0




```
av = (in1+in2+in3+in4);
if (av>0)
    result = av;
  else
    result = 0;
```

My first VI - debug

- Test and debug



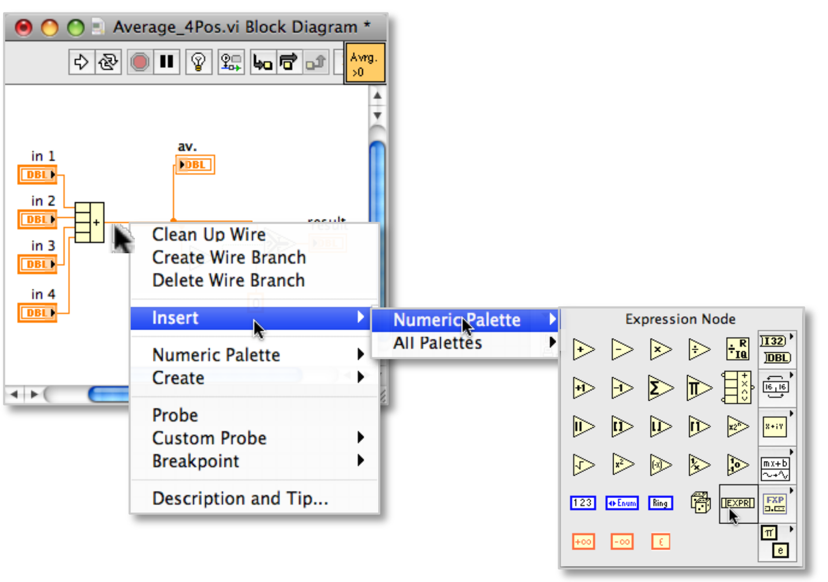

```
function Average_4Pos {
  av = (in1+in2+in3+in4);
  if (av>0)
    result = av;
  else
    result = 0;
}
```

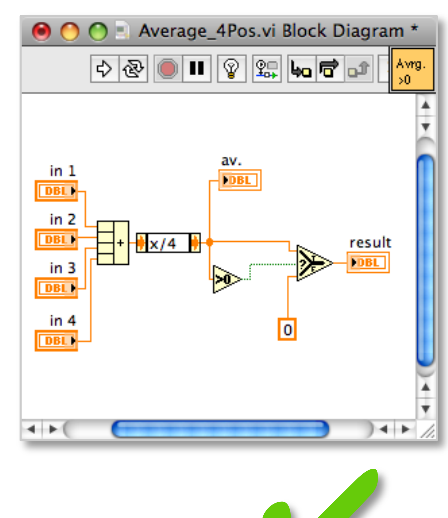
Variables

av>0	FALSE

My first VI - modify

- fix and test again

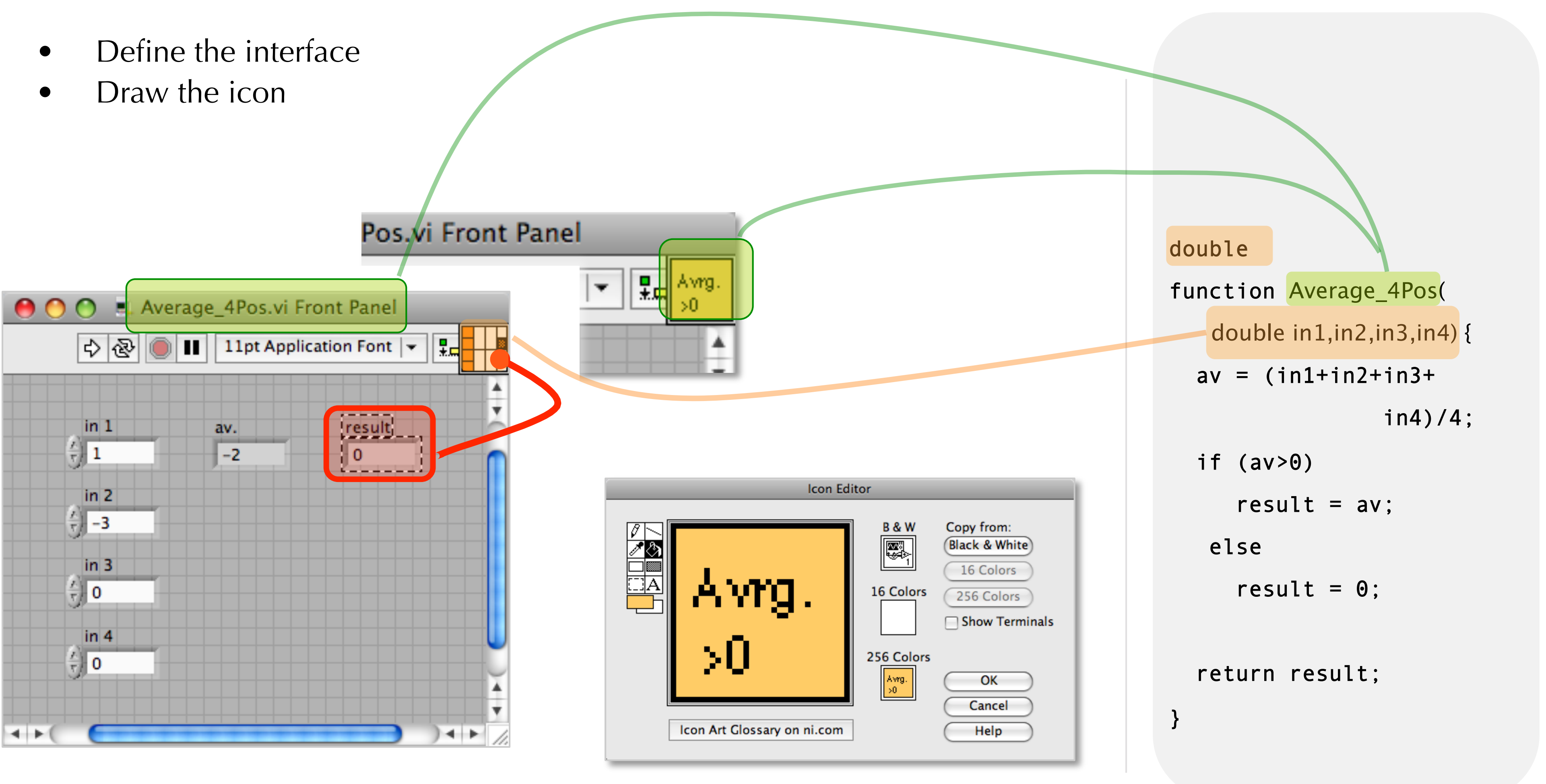





```
function Average_4Pos {
  av = (in1+in2+in3+
                  in4)/4;
  if (av>0)
     result = av;
   else
     result = 0;
 }
```

My first VI – As a function

- Define the interface
- Draw the icon

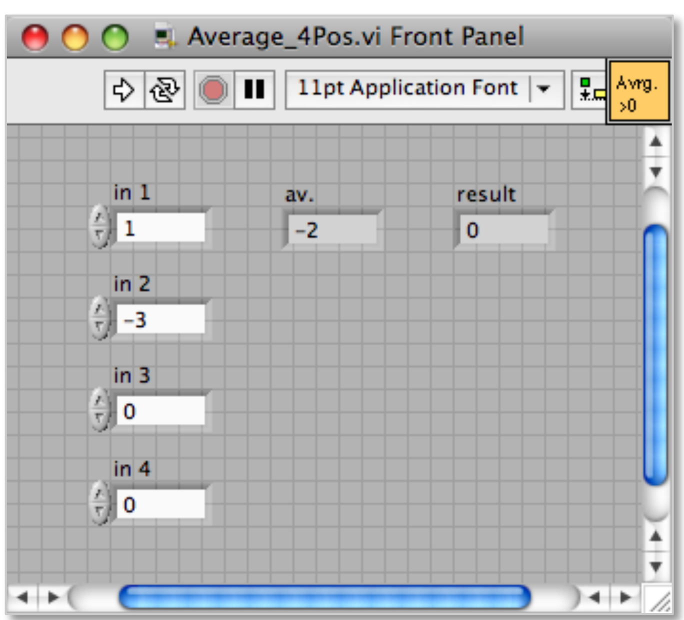


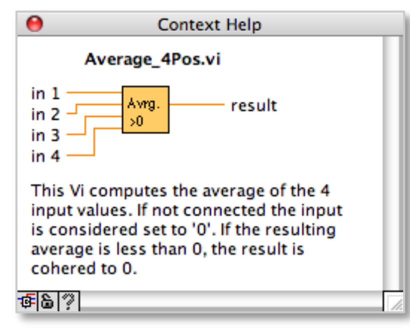

```
double
function Average_4Pos(
  double in1,in2,in3,in4) {
  av = (in1+in2+in3+
                   in4)/4;
  if (av>0)
     result = av;
   else
     result = 0;

  return result;
}
```

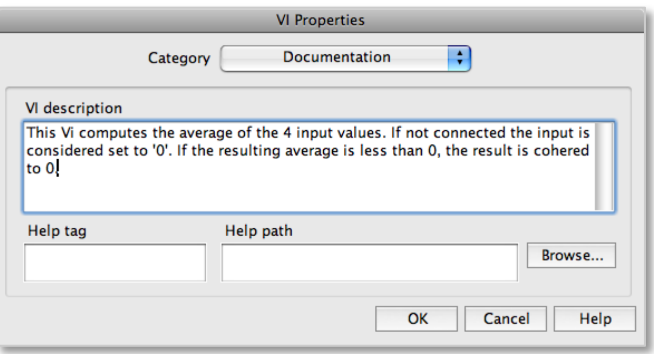
http://www.ni.com/devzone/idnet/library/guidelines_for_creating_icons.htm

My first VI - Documentation

- Write the documentation
- Will appear in the help window



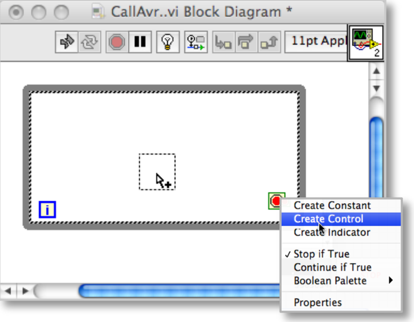


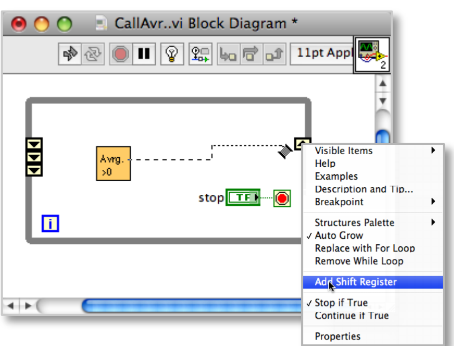


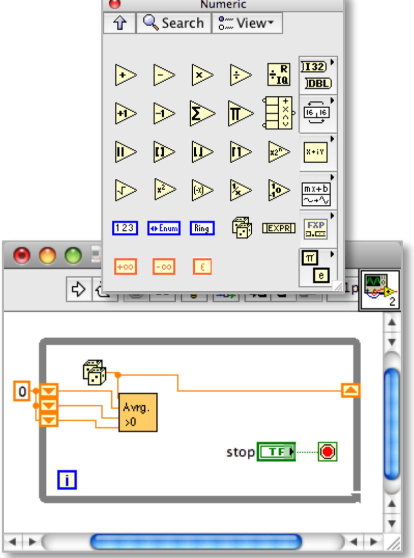

```
/*
 This Vi computes the average of the 4
input values. If not connected the
input is considered set to '0'. If the
resulting average is less than 0, the
result is cohered to 0.
*/
double
function Average_4Pos(
    double in1,in2,in3,in4) {
  av = (in1+in2+in3+
                 in4)/4;
  if (av>0)
     result = av;
   else
     result = 0;

  return result;
}
```

My first VI – Call it

- Create a new vi
- drag and drop the Avrage_4Pos.vi from either the finder, the connector pane, or the palette
- create a control for the Stop button
- Add a shift register, initialize it to 0
- Connect the random number generator node



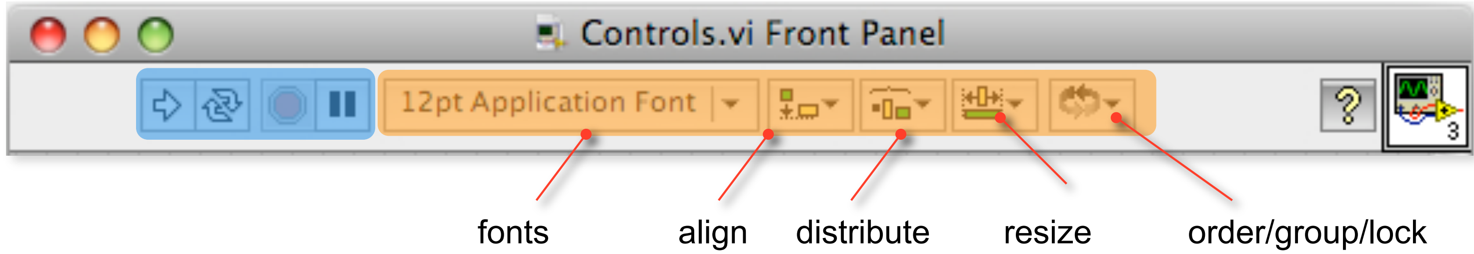


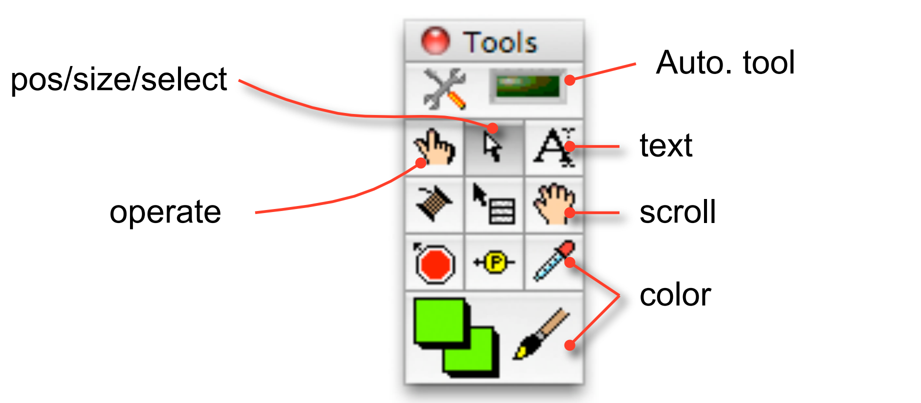


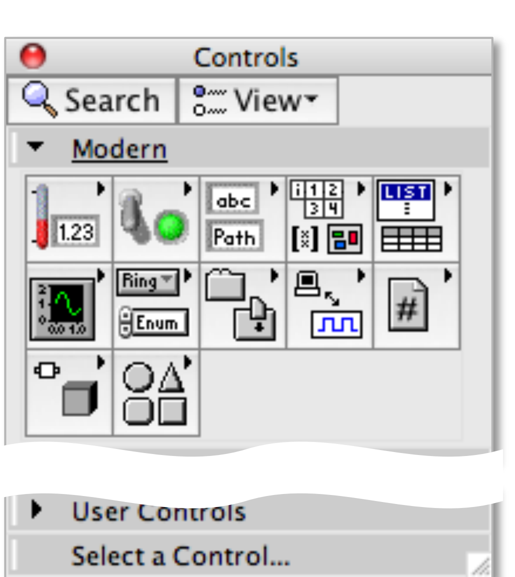

```
void function CallAvr(void){
  sr1=0;
  sr2=0;
  sr3=0;
  While (!Stop) {
     sr=rand();
     sr1=sr;
     sr2=sr1;
     sr3=sr2;
     Avrage_4Pos(sr,sr1,
            sr2,sr3);
     };
}
```

Tools bars - front panel

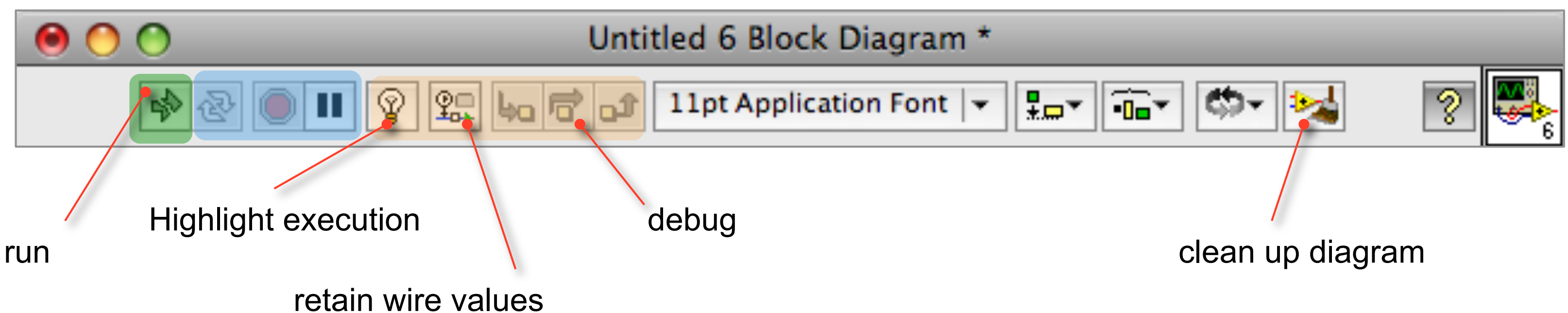
Execution

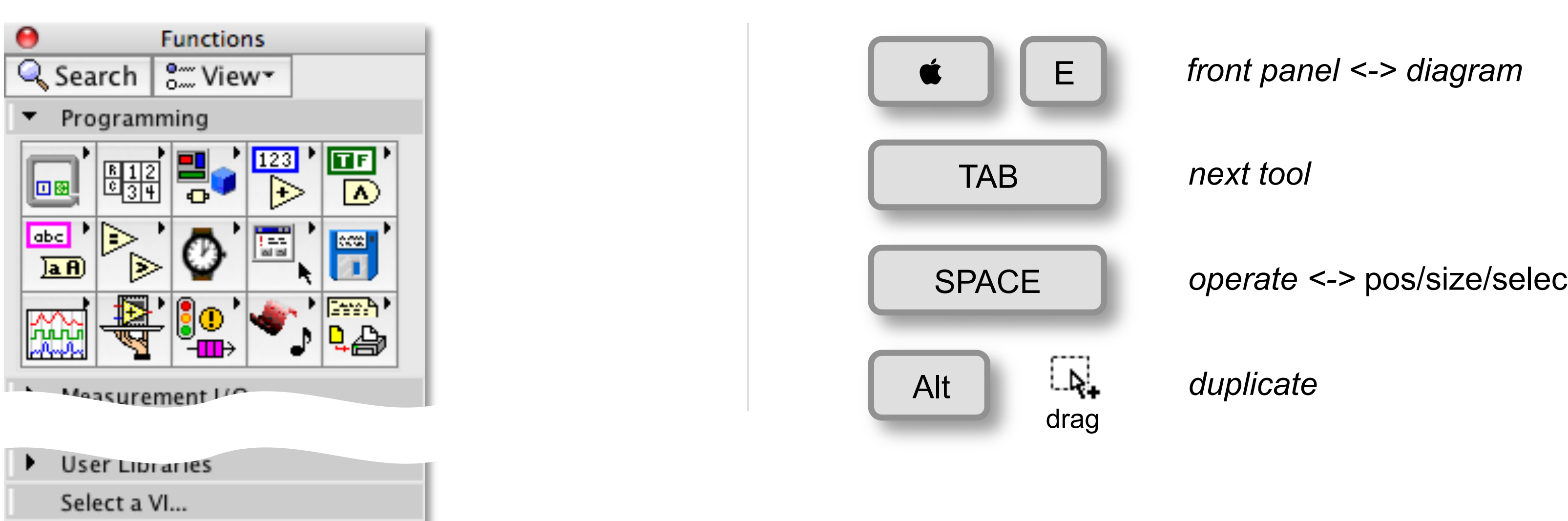




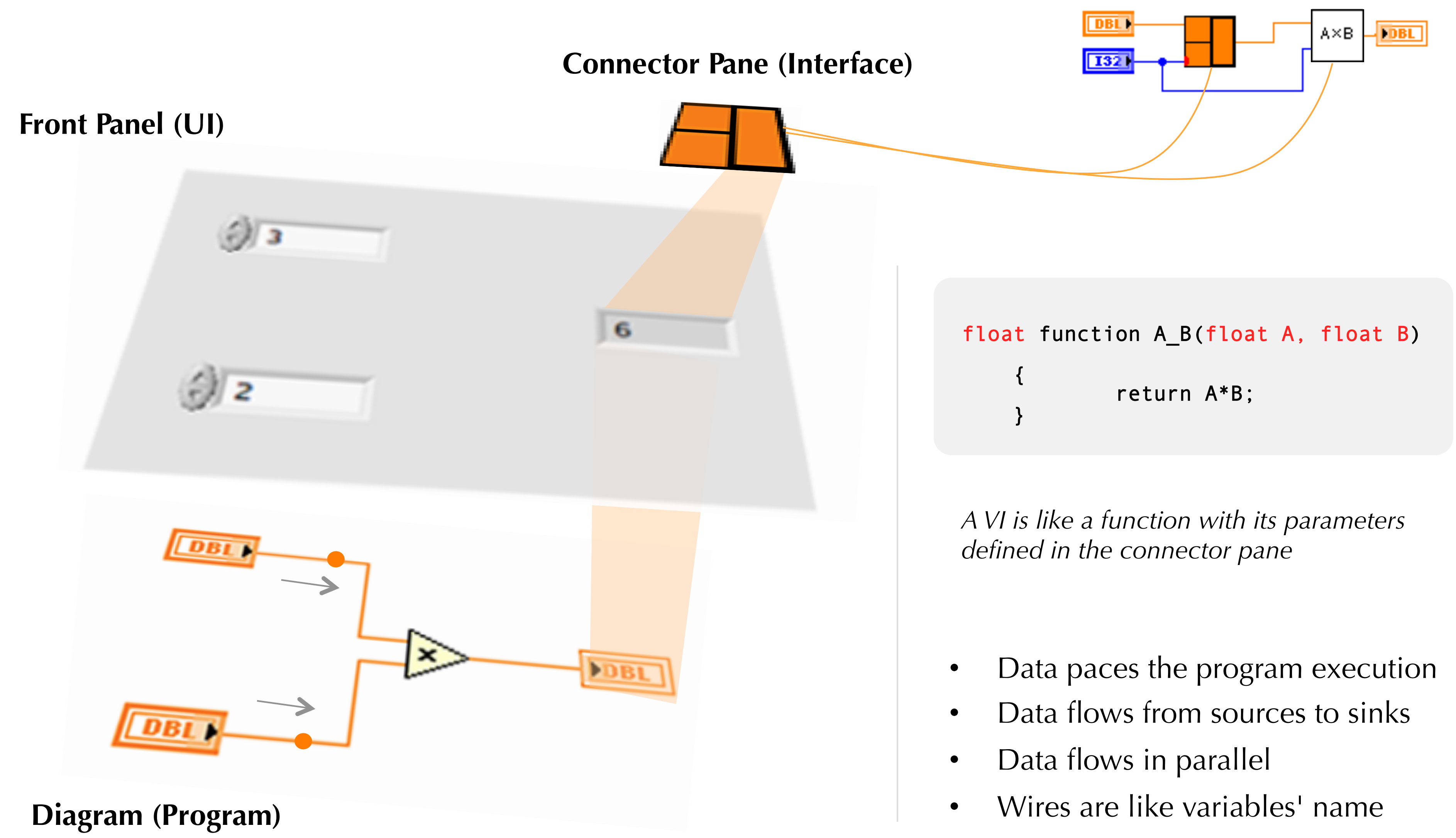


Tools bars - diagram





Recap: Virtual Instrument (VI)




```
float function A_B(float A, float B)
    {
            return A*B;
    }
```

A VI is like a function with its parameters defined in the connector pane

- Data paces the program execution
- Data flows from sources to sinks
- Data flows in parallel
- Wires are like variables' name

Recap

- Virtual Instrument & data flow programming
  - Vi is made of a front panel, a diagram, a connector pane and some documentation
  - The execution of a node is only possible when all the needed data are ready
- *G* is strongly typed, wire color indicate its type, wire thickness indicates its dimension
- All classical structures are available in *G*
- In loops (For/While) wires are evaluated once at the loop border
- Shift registers hold their values until the VI is removed from memory (= no reference to it, as in sub-vi)

Resources

- http://www.mech.uwa.edu.au/jpt/tutorial/index.html
- http://www.mines.edu/academic/courses/eng/EGGN383/ref/r29/
- http://www.eelab.usyd.edu.au/labview/main.html
- http://online.physics.uiuc.edu/courses/phys405/fall05/P405_Labs/Lab4_LabVIEW_Primer/Lab4_LabVIEW_primer.pdf
- http://www.iit.edu/~labview/Dummies.html
- http://www.ee.upenn.edu/rca/software/labview.html
- http://egweb.mines.edu/eggn350/labview/
- http://oldwww.rasip.fer.hr/research/labview/gintro.html
- http://www.tufts.edu/as/tampl/program/workshops/workshop2.html
- http://www-ee.eng.buffalo.edu/faculty/paololiu/edtech/roaldi/tutorials/labview.htm
- http://c.webring.com/hub?ring=labview&id=97&prev5
- https://sine.ni.com/apps/we/nigb.main?code=GB_TUTLV