EPFL

Programmation pour Ingénieur

LabVIEW 1

ME 3e semestre

rev. 2025.2

Christophe Salzmann

Laboratoire d'Automatique

Introduction à LabVIEW

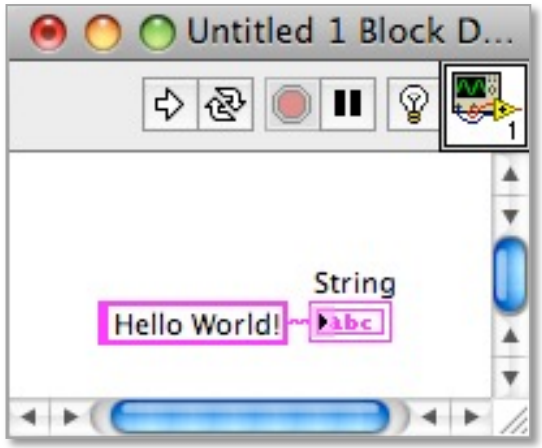
- Pourquoi LabVIEW ?
- Histoire
- Concepts de base
- Structures de contrôle et types de base
- Mon premier VI (Virtual Instrument: application)

But: **développer ma première application**

Pourquoi LabVIEW?

Pourquoi utiliser LabVIEW?

Vous serez bien plus productif!



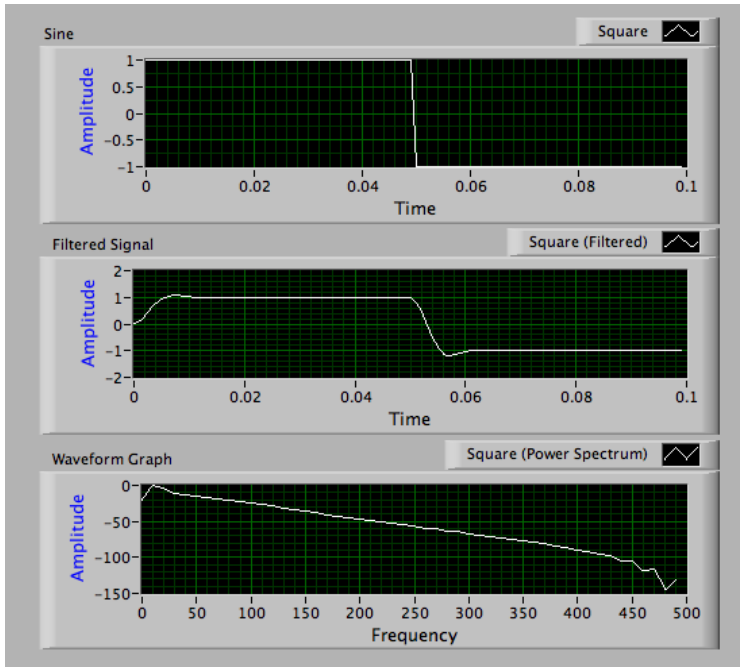
Demo 1

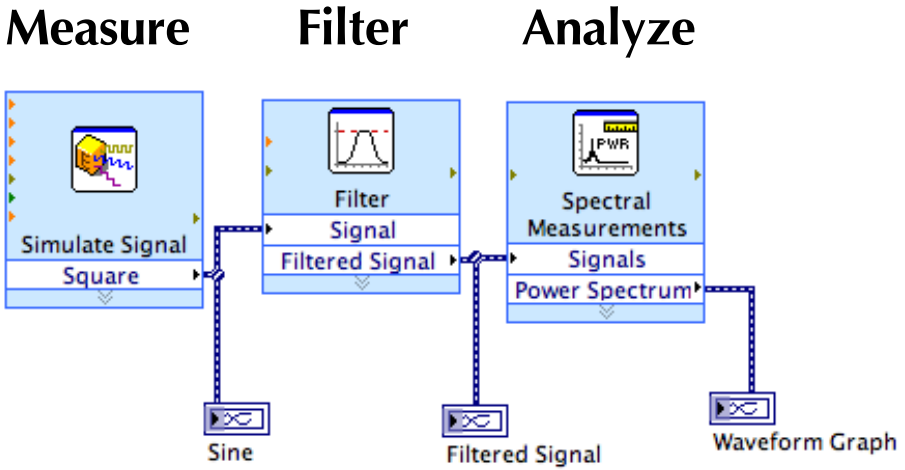
Ma première mesure <5 min

avec un arduino

Pourquoi utiliser LabVIEW ?

Vous serez bien plus productif!

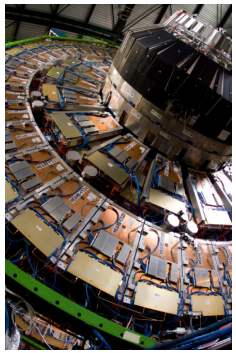


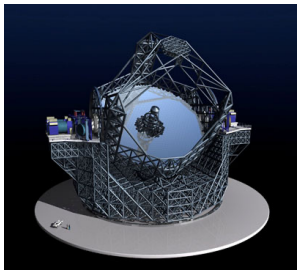


LabVIEW - examples

- CERN LHC
- SALT telescope
- ESO telescope
- Honeywell
- Control Lab MOOC

...





Pourquoi LabVIEW ?

- LabVIEW provides access to all NI hardware + other

- Access **Zillions** of PCI, PXI, FireWire, USB, Ethernet, GPIB, FPGA, etc boards/modules on multiple hardware and OS.

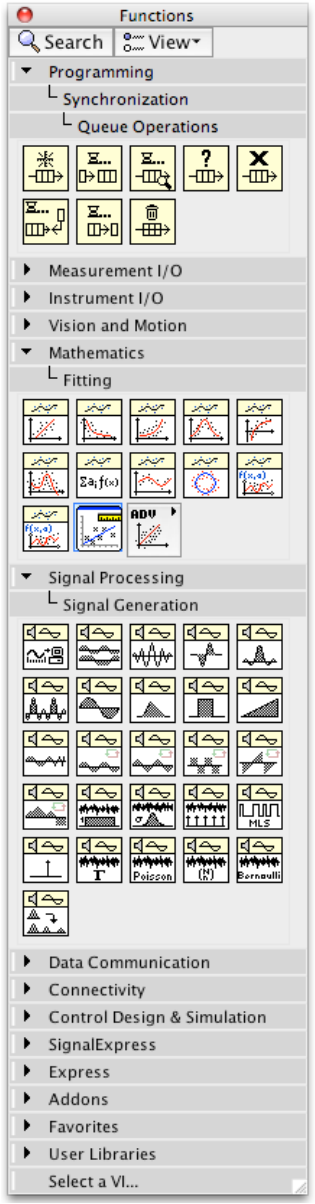
With the same environment :-)

Pourquoi 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

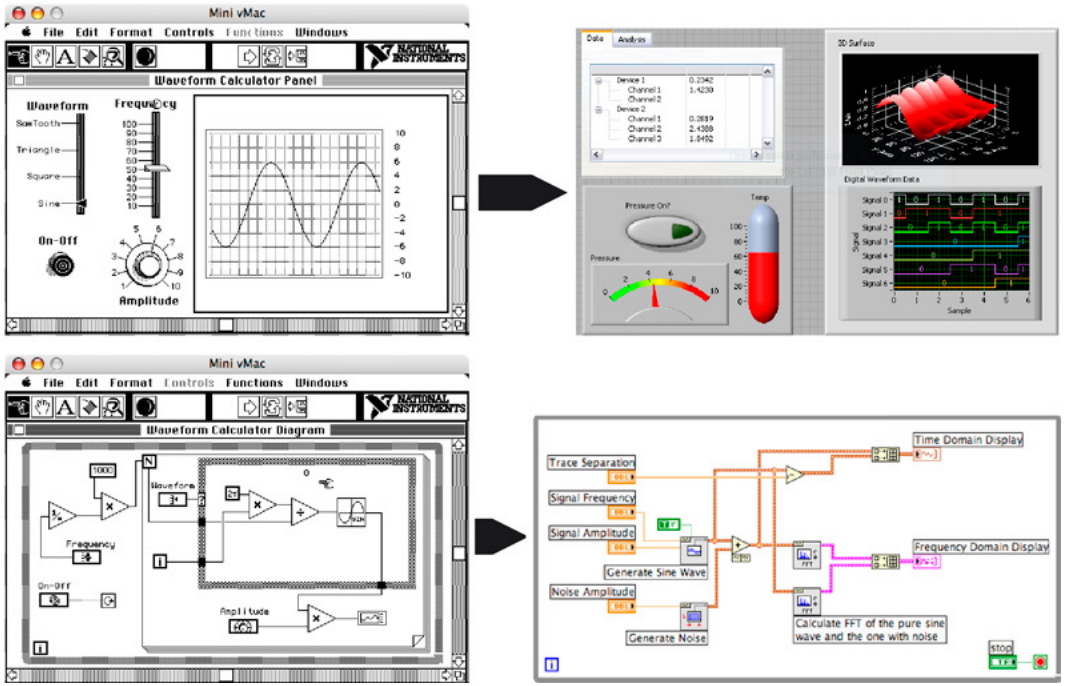


LabVIEW history (short)

1986 - LabVIEW 1

..

2025 - LabVIEW 2025



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

LabVIEW

2 concepts principaux:

Intrument virtuel

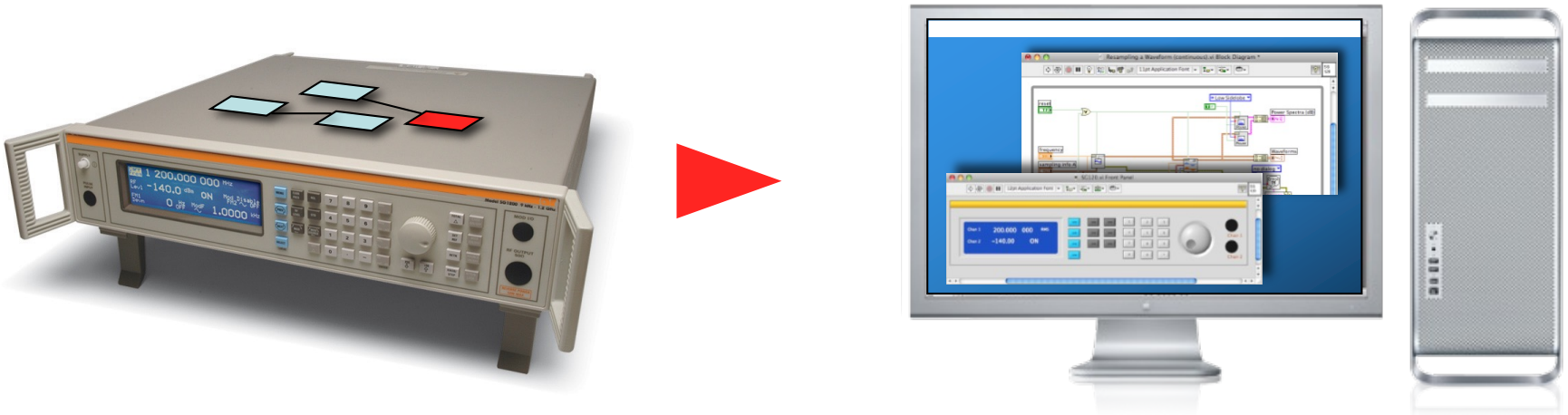
(VI:Virtual instrument)

Programmation par flot de données

(Data flow programming)

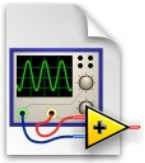
Instrument virtuel

- Mimic un instrument réel



Front panel - diagram - connector pane

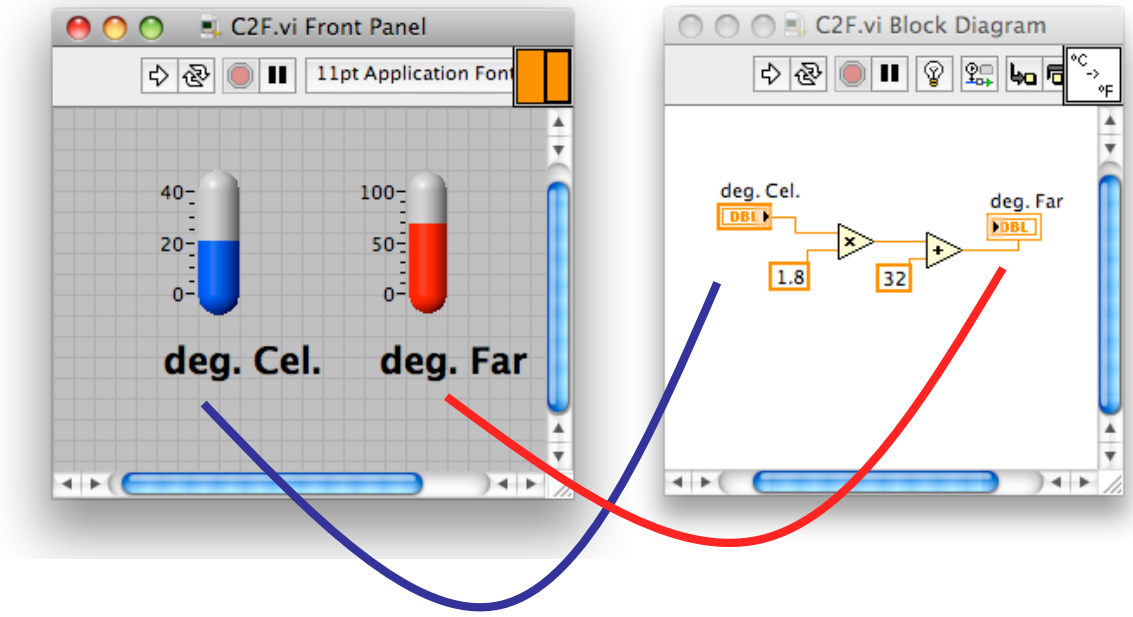
Instrument virtuel (VI)



Front panel

Connection Pane

Diagram

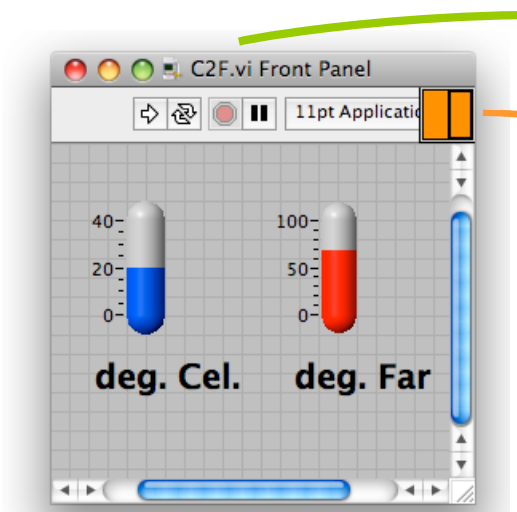


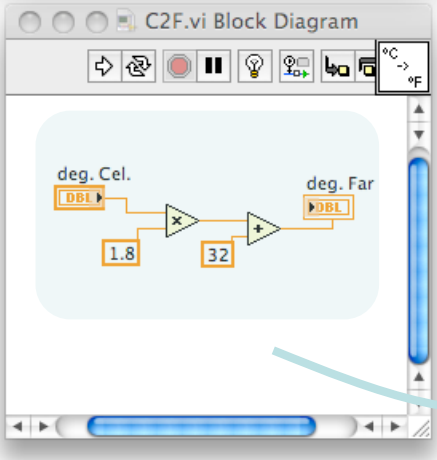
Control

Indicator

LabVIEW se programme en glissant des icones depuis une palette

Equivalent en *C*





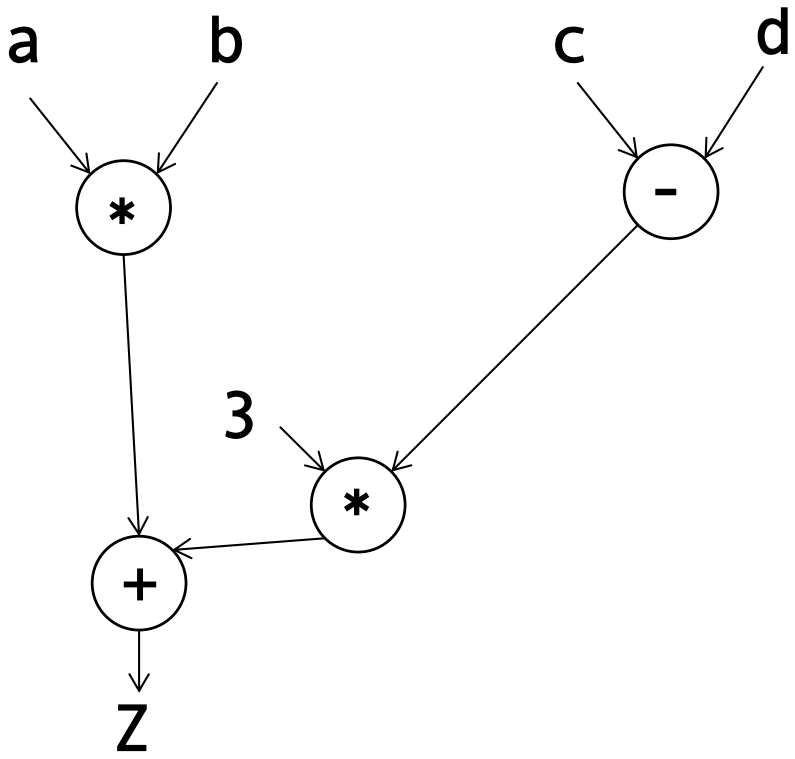

```
double function C2F (double deg_cel)
{
    return deg_cel * 1.8 + 32;
}
```

Demo – C2F

deg x 1.8 + 32

Ctrl + indicateur + couleur file + c. pane

Programmation par flot de données



L'écoulement des données rythme l'exécution du programme

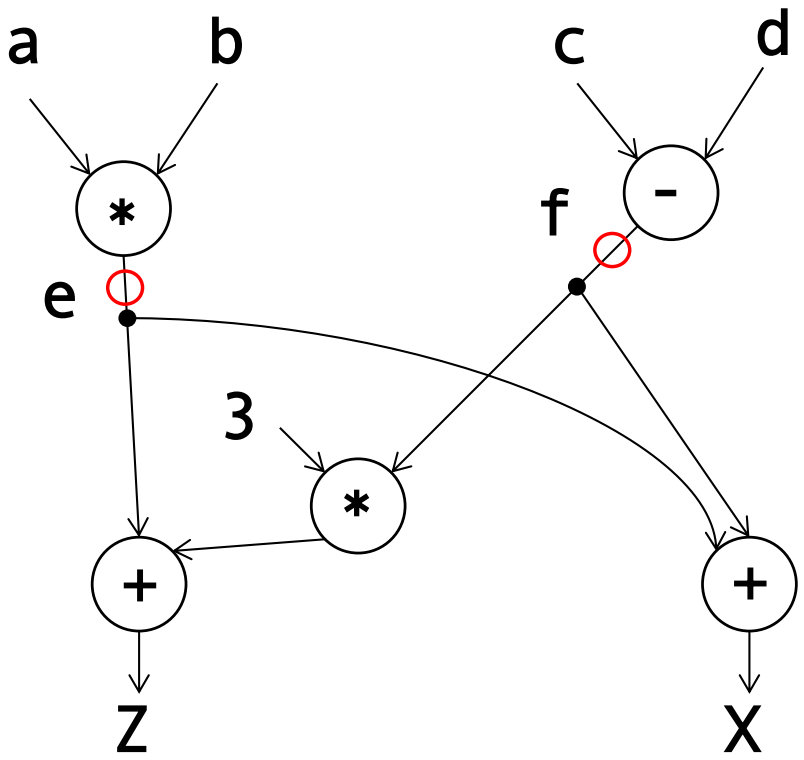
C'est comme pour les petits ruisseaux qui se réunissent pour former de grandes rivières et peuvent ensuite se séparer en plusieurs rivières

L'écoulement des données est comparable à l'écoulement de l'eau ou des électrons dans un circuit électrique

Un noeud est exécuté seulement quand toutes les entrées sont connues

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

Programmation par flot de données



L'écoulement des données rythme l'exécution du programme

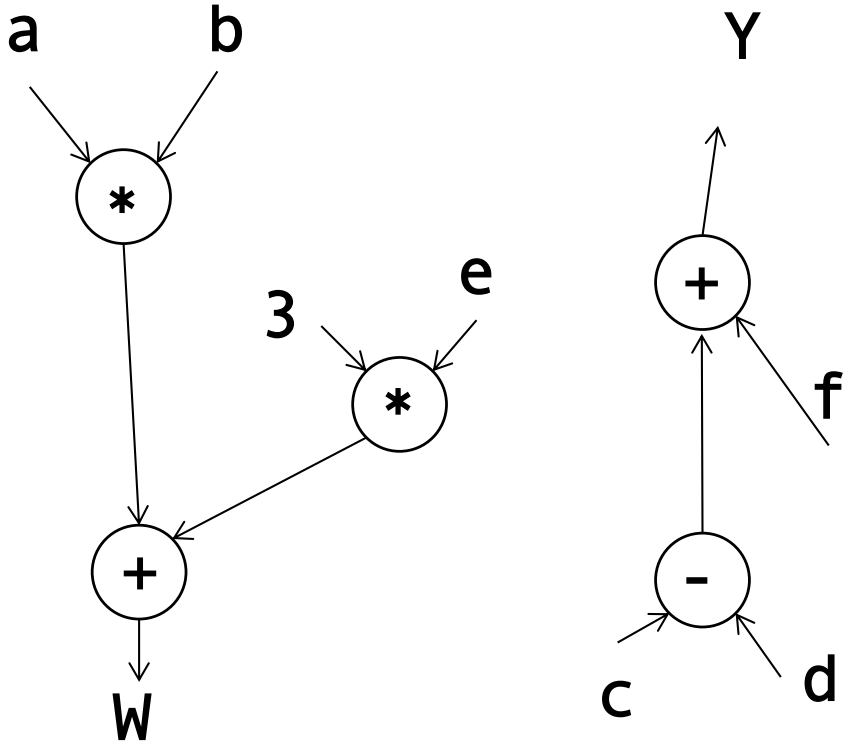
C'est comme pour les petits ruisseaux qui se réunissent pour former de grandes rivières et peuvent ensuite se séparer en plusieurs rivières

L'écoulement des données est comparable à l'écoulement de l'eau ou des électrons dans un circuit électrique

Un noeud est exécuté seulement quand toutes les entrées sont connues

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

Programmation par flot de données



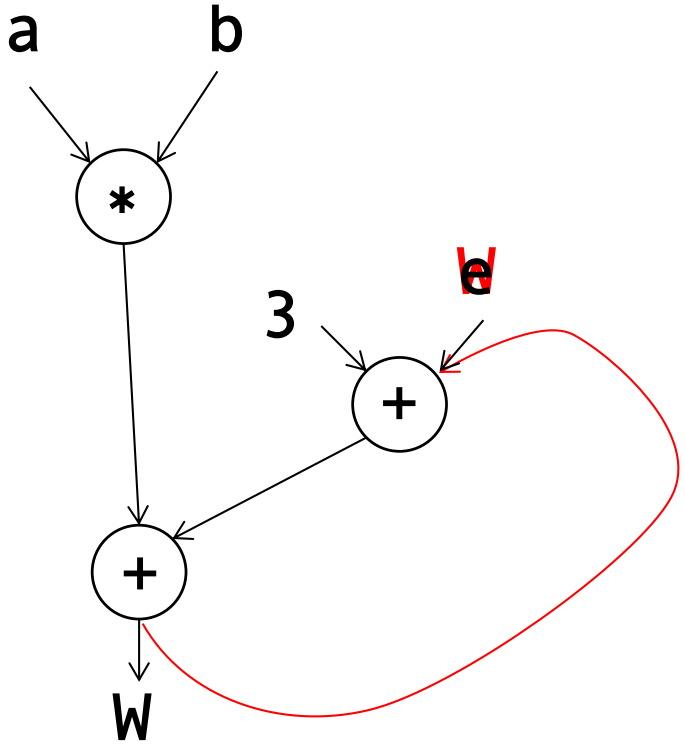
S'il n'y a pas de dépendance entre les nœuds, ils s'exécutent de manière concurrente (en parallèle) à leurs propre rythme

Il n'y a pas d'ordre de préférence pour l'exécution à par le flot des données

La notion de parallélisme est comprise dans le langage lui-même, par défaut tout ce qui peut être exécuter en parallèle l'est.

```
W = (a * b) + 3 * e
Y = (c-d) + f
```

Programmation par flot de données



Comment faire pour calculer W alors qu'il dépend de W ?

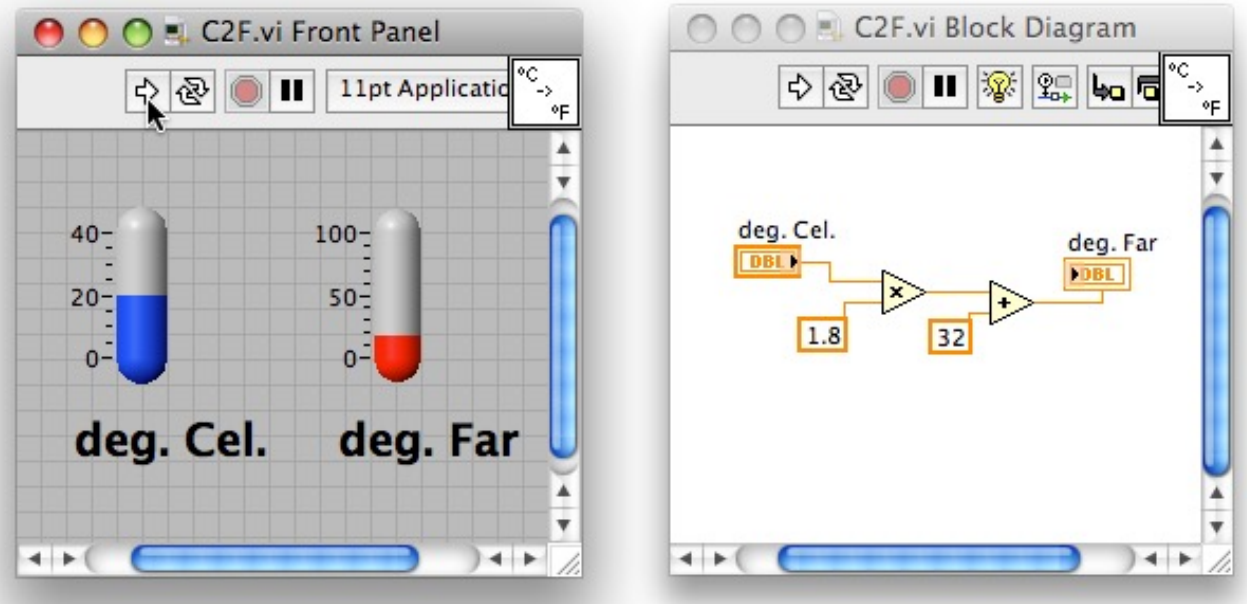
Est-ce que W et W sont les mêmes ?

*Il y a un problème de **causalité**!*

Dans LabVIEW, il est résolu en définissant W comme étant W au temps/itération précédent(e), le fil rouge ayant un effet retardateur.

```
W = (a * b) + 3 + W
```

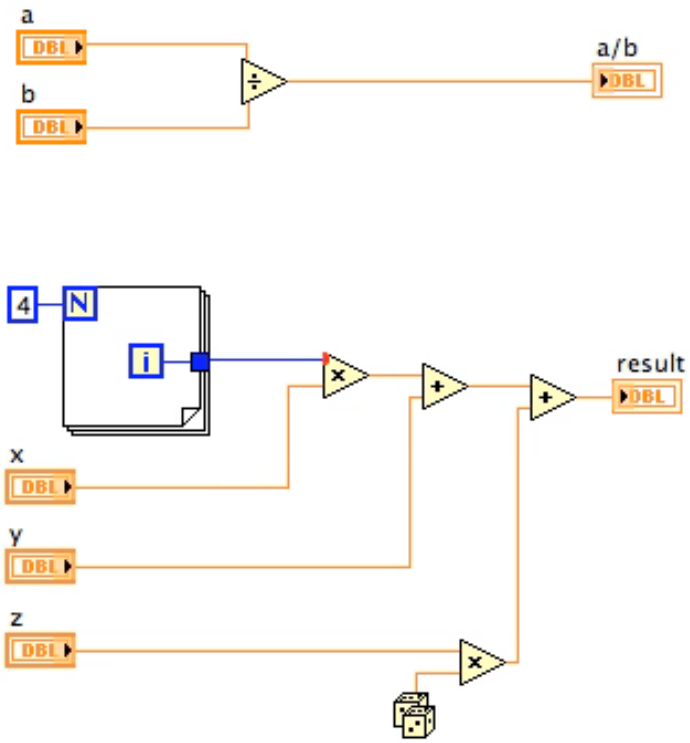
Programmation par flot de données



L'écoulement des données rythme l'exécution du programme!

Programmation par flot de données

L'écoulement des données rythme l'exécution du programme!
Les parties du diagram peuvent s'exécuter en parallèle

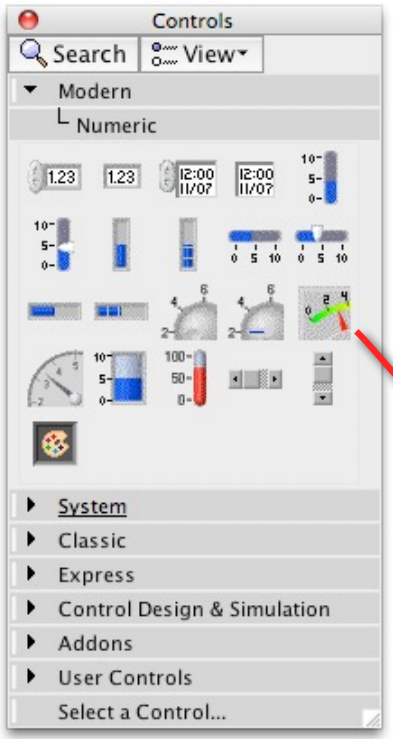


LabVIEW - environment

- LabVIEW IDE est un environment complet:
  - **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

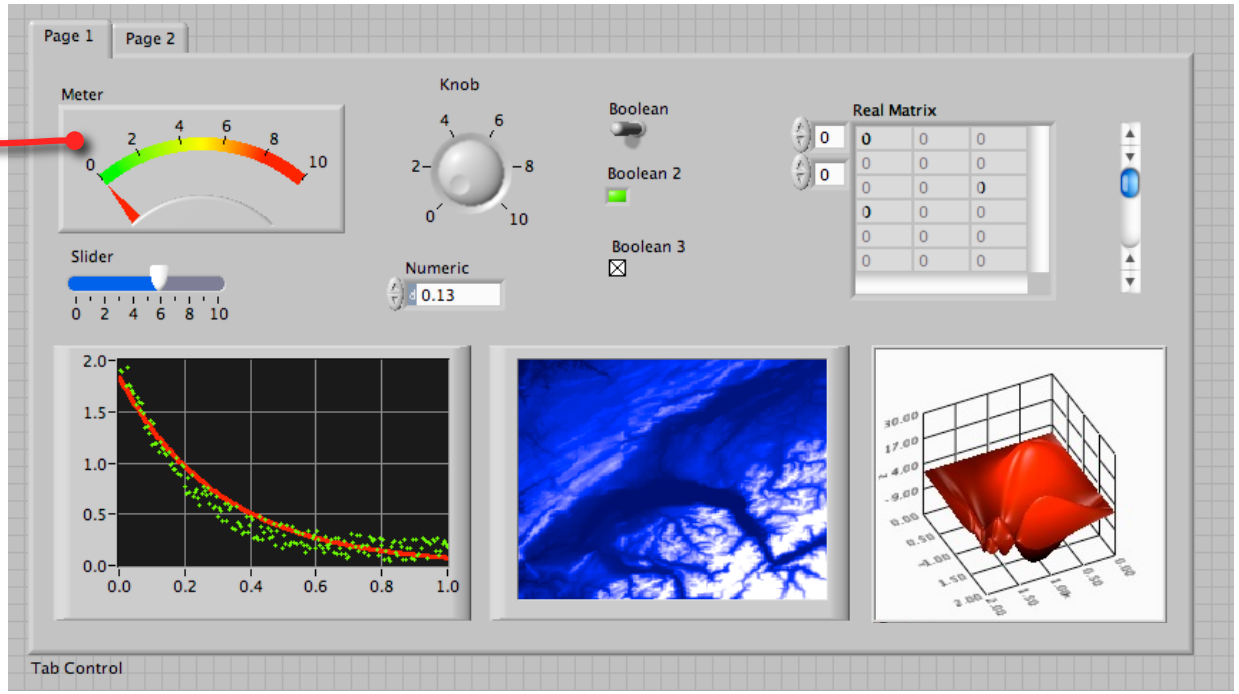
GUI Builder - Front panel



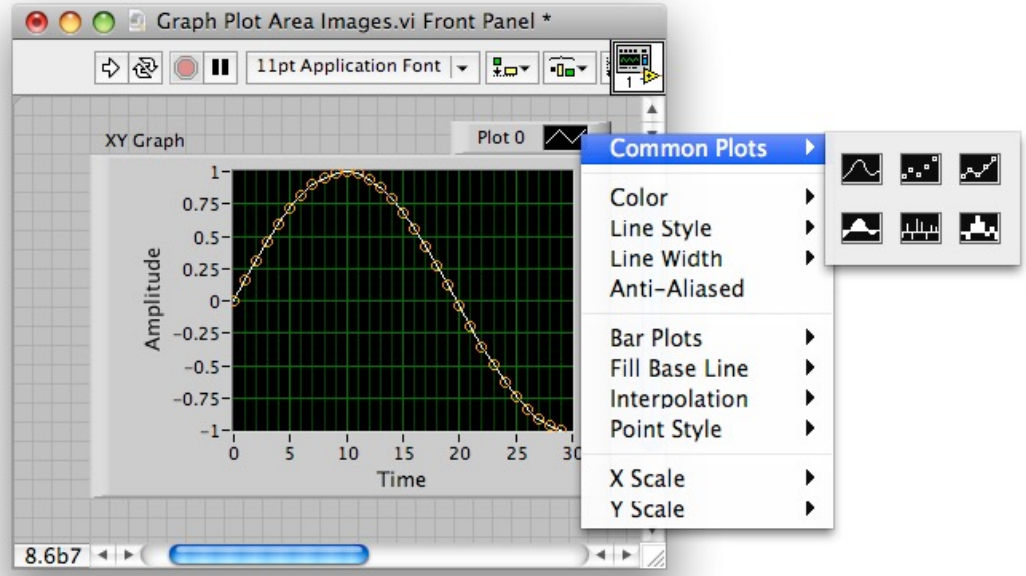
Dessiner votre interface utilisateur (GUI) comme vous le feriez avec un programme de dessin (illustrator, powerpoint, etc.)

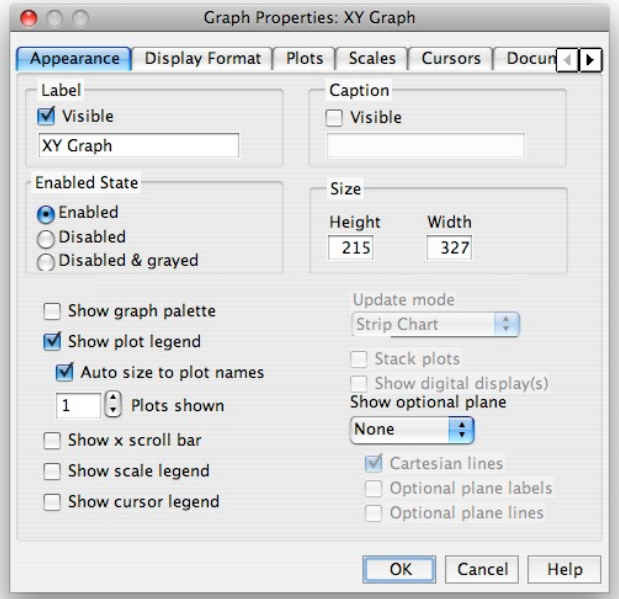
Les *Controls* et *indicators* sont accessibles via la palette **Controls**

Utilisez le right-click et/ou drag&drop pour placer vos *controls/indicators*



GUI builder - edit object

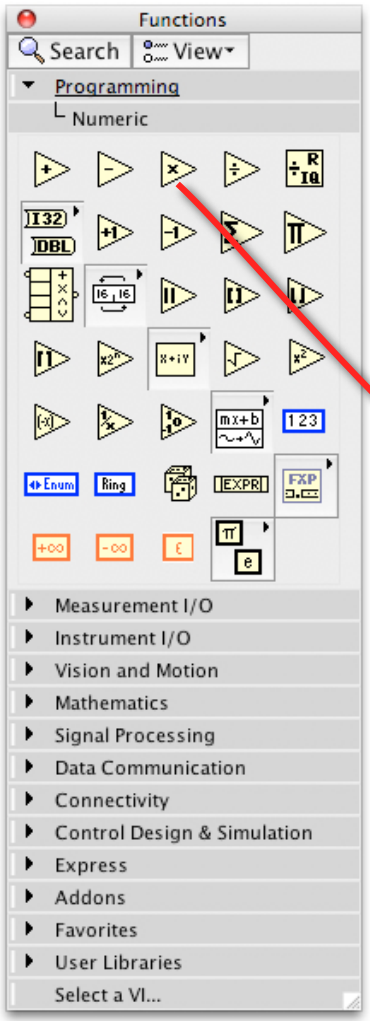




Utilisez le right-click sur un objet vous permet de spécifier les options d'affichage de vos *controls/indicators*

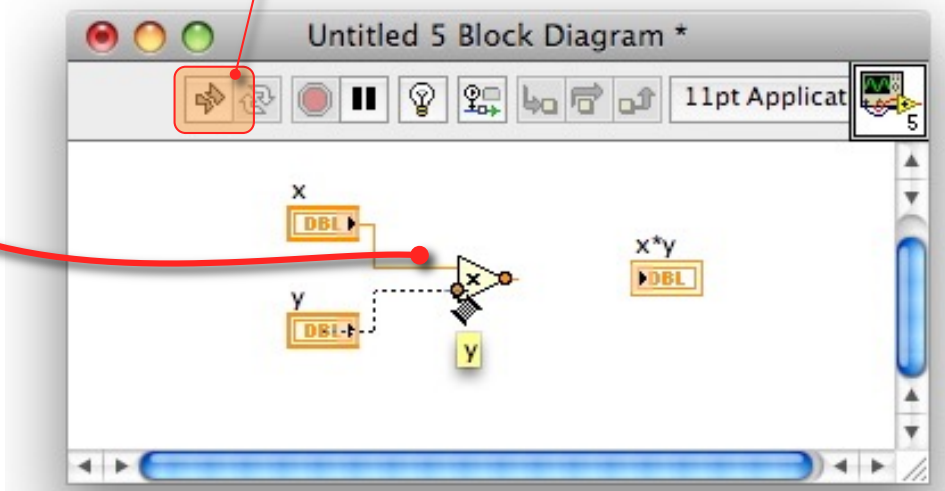
Les propriétes sont toutes accessibles au travers d'un dialogue

G - editor



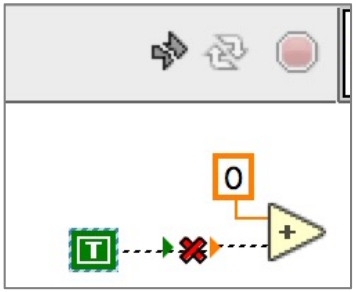
- Glisser/déposez les noeuds/fonctions
- Connectez-les avec la *wiring tool*

compilation error



G editor (diagram)

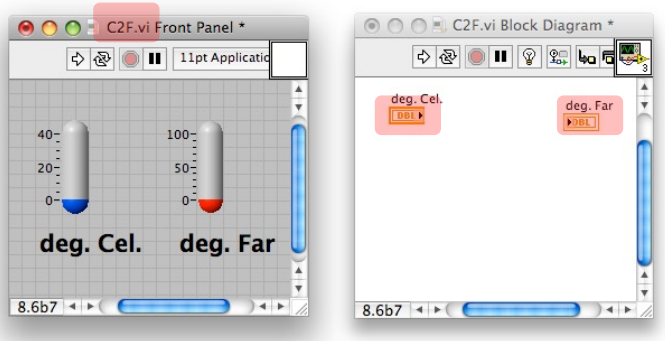
- G est le langage graphique pour programmer LabVIEW
- La programmation de LabVIEW se fait en déposant et connectant des noeuds représentants des fonctions dans le *digram* du VI
- Toutes les structures classiques existent en G: boucle, if, switch, etc.
- G est polymorphique (=auto adaptation au type des donnée)
- La vérification syntaxique se fait en continu avec un feedback direct (broken arrow & dotted wires)



- Le compilateur LabVIEW génère du code machines et cross-compile pour C, VHDL, etc.
- Le compilateur LabVIEW génère du code multi-threaded, multi-cores et temps réel

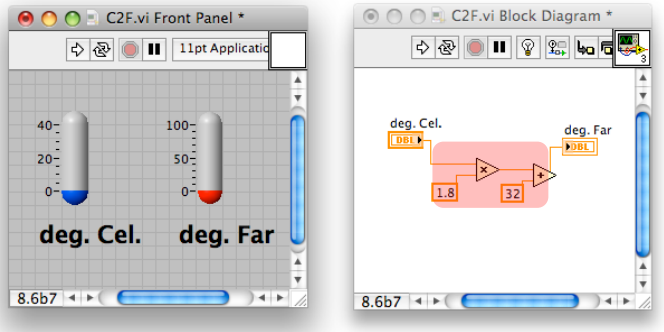
G editor - C equivalent

Name



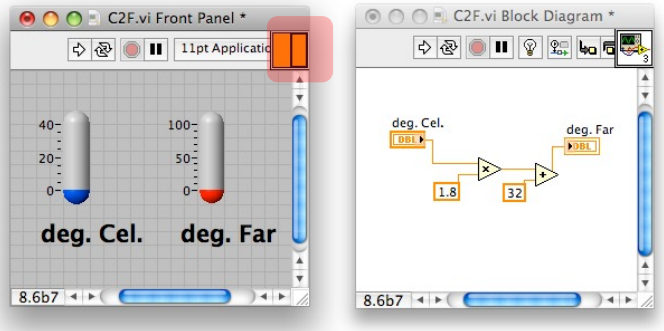

```
void function C2F (void)
    {
            double deg_cel, deg_far;

    }
```

Code




```
void function C2F (void)
    {  double deg_cel, deg_far;
            deg_far = deg_cel * 1.8 + 32;
    }
```

Params.

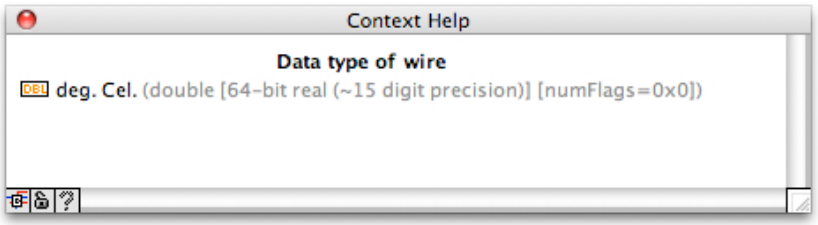


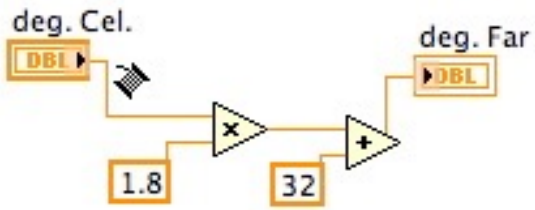

```
double function C2F (double deg_cel)
    {
            return deg_cel * 1.8 + 32;
    }
```

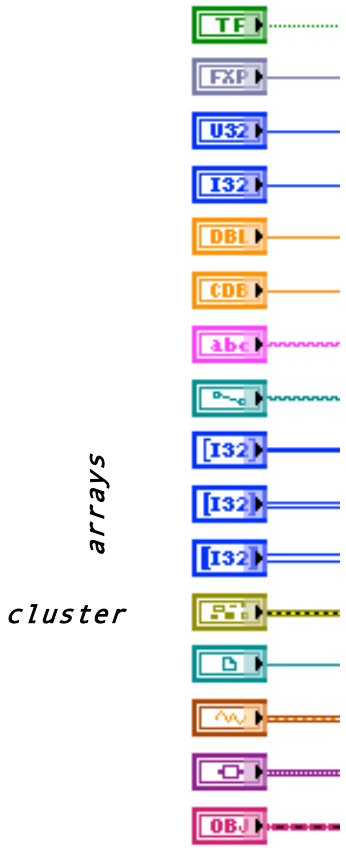
G data types

G types de donnée

- G est fortement typé, comme le C/C++
- Le type est défini par la couleur et la taille
- En positionnant la *wiring tool* sur un fil, le type et l'unité du fil sont affichés dans l'aide contextuelle





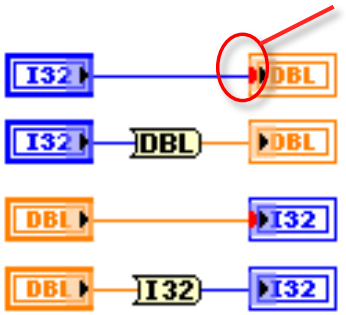



```
boolean;
fixed_pt;
ulong;
long;
double;
cdouble;
string;
path;
long[];
long[][];
long[][][];
struct{};
refnum;
waveforn;
variant;
object;
```

G data types

- La couleur défini le type
- L'épaisseur du trait défini les dimensions
- Un point rouge indique une conversion automatique
- ⚠ **Les conversions sont différentes du C/C++** , elles suivent la norme IEEE 754
- Une conversion de type peut être forcée

automatic conversion



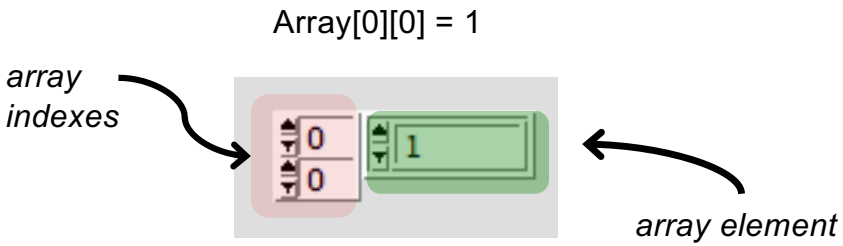
2.5 -> 2
3.5 -> 4 ⚠

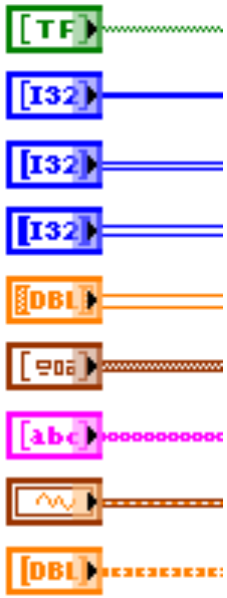
Rounding to the nearest Even integer:
Pour les nombres qui sont exactement à mi-chemin entre deux entiers comme par ex. 2.5, 3.5, etc. ils sont arrondis à l'entier **pair** le plus proche. (pour évitez les erreurs systématiques)
ex. 2.5 -> entier pair le plus proche -> 2
3.5 -> entier pair le plus proche -> 4

G types – Tableau *array*

Tableau (Array) -> structure contenant des éléments du même type

- *Arrays* à plusieurs dimension (≤ 64)
- *Arrays* de n'importe quels types composés
- Arrays sont dynamiques, leur taille peut être changée automatiquement durant l'exécution
- La largeur des '[]' indique sur les dimensions du tableau
- Ils existe un grand nombre de fonctions prédéfinies travaillant sur les tableaux, ex. algèbre linéaire



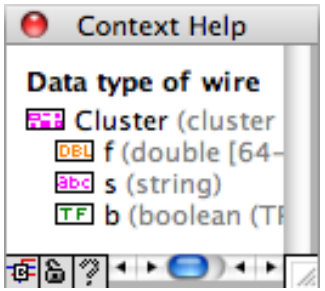


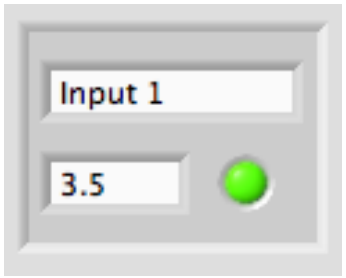

```
Bool[];

long[];

long[][];

long[][][];

float[][][][][]..

struct[];

Str[];

waveform;

matrix;
```

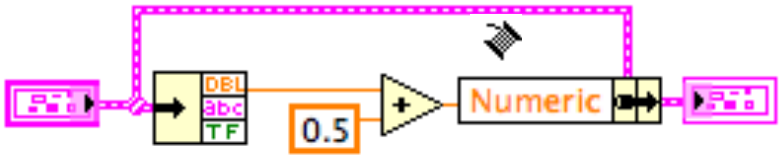
G data types - cluster

Cluster-> structure avec des éléments de types différents

- Equivalent des `struct` en C/C++
- Le nombre d'éléments des clusters est fixe
- La couleur des clusters indique si les éléments sont de taille fixe (brun) ou non (rose)
- Il est possible de mettre un Cluster dans un Cluster
- Les éléments (champs) du Cluster peuvent être accédés par leur nom (recommandé) ou leur position







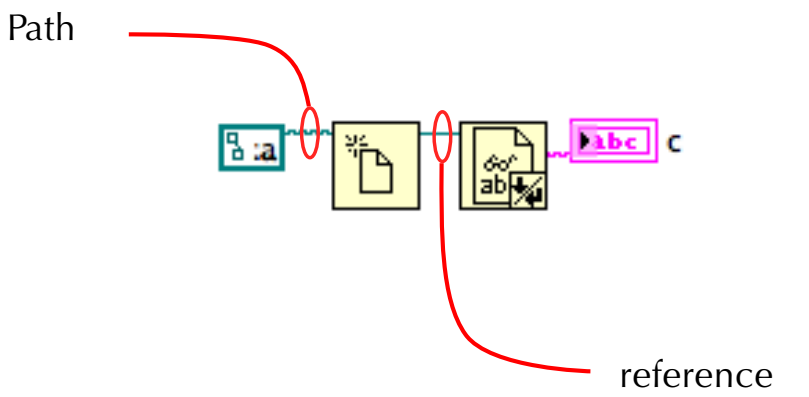

```
struct {
   double f;
   char[] s;
   bool   b;
} c;
```

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

G data types – reference

Les *references* sont similaires aux pointeurs/références en c++

- Mémorise une référence à un objet/donnée de LabVIEW
- Les *references* ne peuvent pas être observées (code hexa)
- Utilisée pour les accès fichiers, réseau, etc.

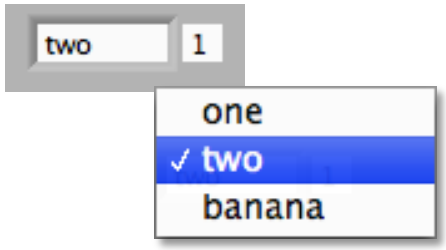


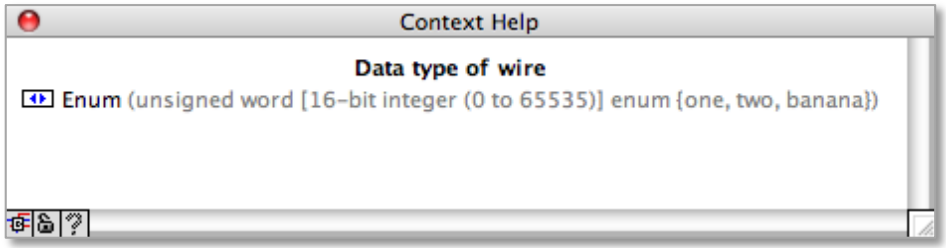

```
FILE      *f;
char[255]  c;

f= fopen(”a”);
e= fread(f,c);
```

G data types - enumeration

Enumeration

- Ensemble de noms représentés par des nombres





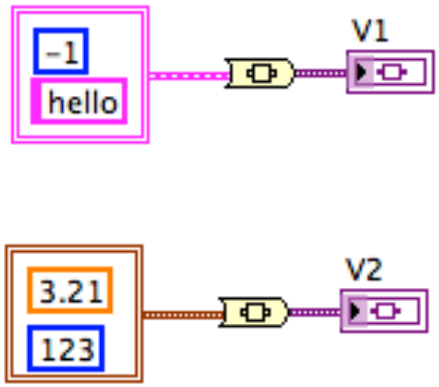
U16 = 1

```
enum {
  one,
  two,
  banana
}


where

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

G data types – variant

Conteneur générique

- Stock tout type de données LabVIEW
- Stock des données et leurs descriptions
- Peuvent être "sérialisés"
- Les descriptions des *variants* peuvent être éditées (add, remove)

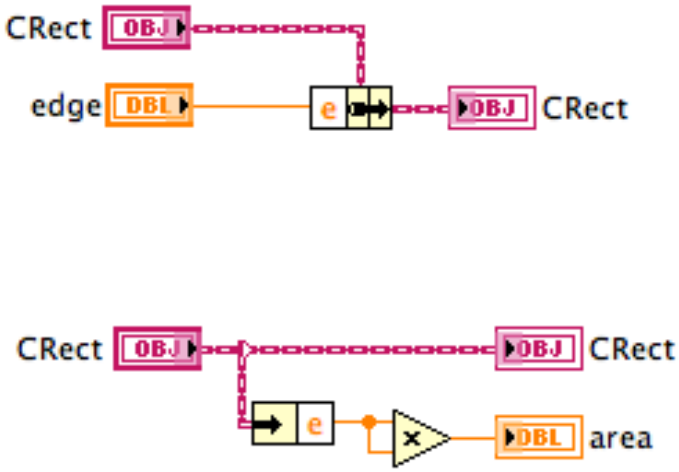


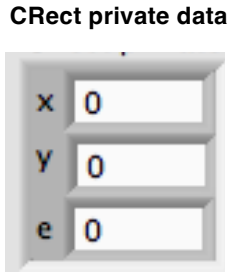

```
struct {
  int i;
  char s[255];
} S1;

struct {
  double b;
  int j;
} S2;

void* V1, V2;

V1= &S1;
V2= &S2;
```

G data types – object

Objet pour la programmation Object Oriented Programming

- Différentes implémentations (NI, others)



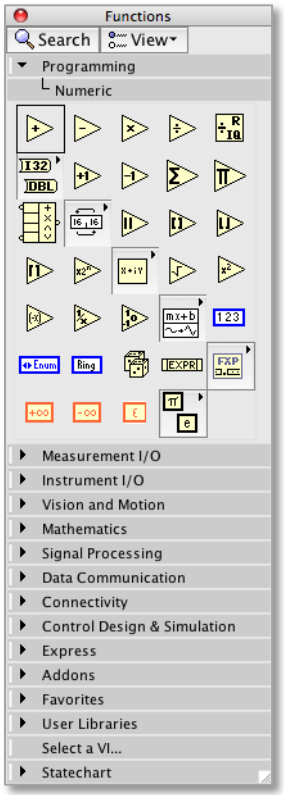


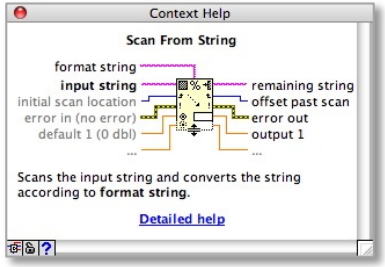

```
class CRectangle {
 int x, y, e;

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

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

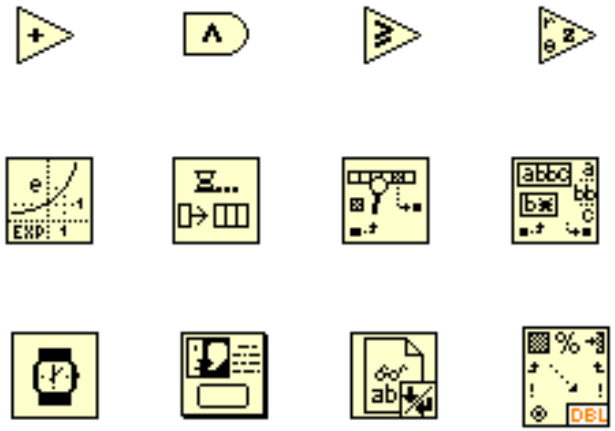
G-functions

G-functions

- Les fonctions/noeuds de base ont un fond jaune
- Elles ne peuvent pas être modifiées
- Elles sont regroupée par type dans des sous-palettes
- La pluspart des fonctions sont polymorphiques, i.e. s'adaptent au type de données connectées



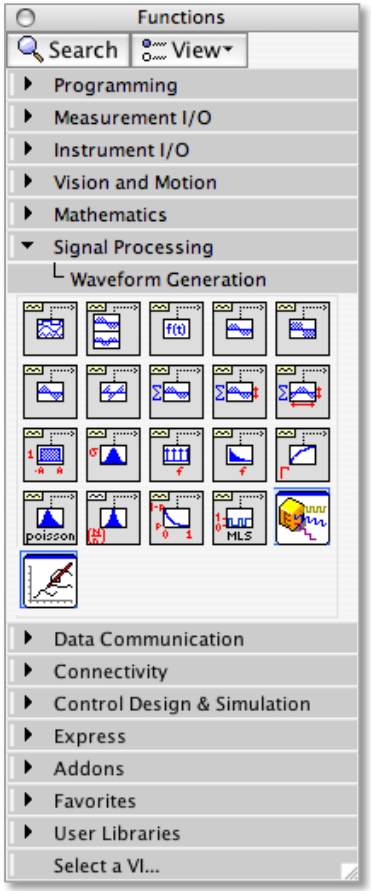


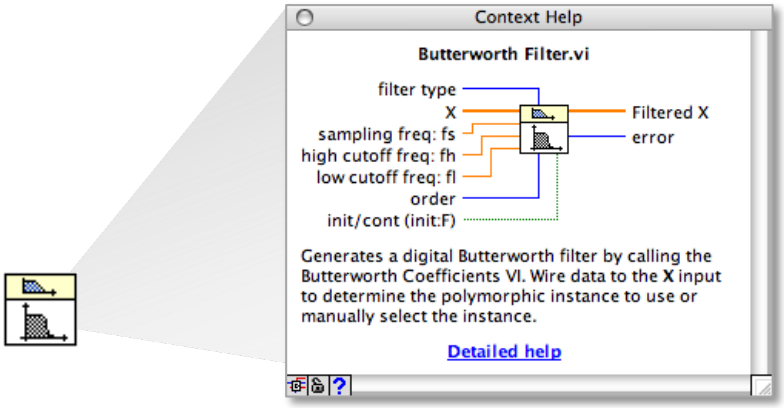


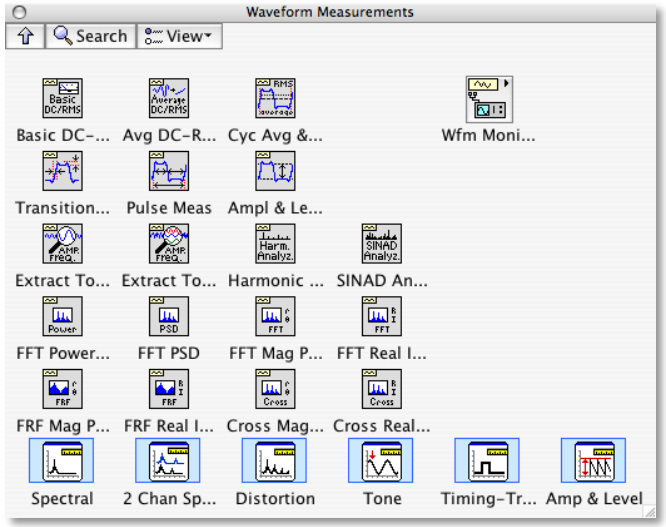
```
+
and
>=
r*e^(i*theta)
exp
enqueue
find in array
find str
millisecond
show dialog
read text file
parse string for float
```

G-functions - VIs

- Fonctions avancées sont des VIs dont le code peut être visualisé/modifié

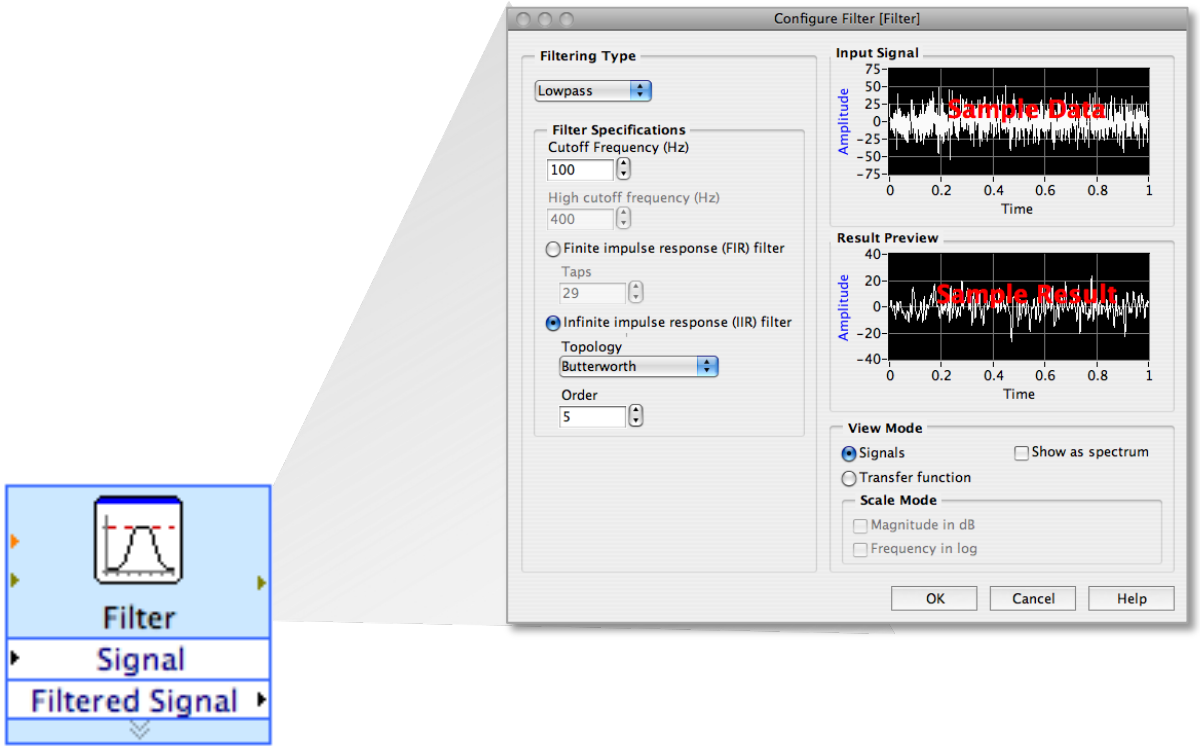






G-functions – Express VI

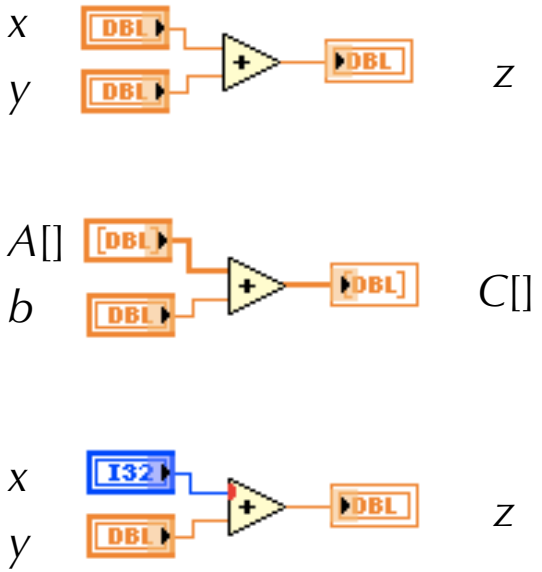
- Les VIs Express sont des fonctions avancées, ex: générer un signal, écrire dans un fichier
- Ils sont configurables au travers d'un dialogue




```
EditParam(p);

FS=Filter(p,S)
;
```

G-polymorphism

- Polymorphism: une fonction/noeud supporte plusieurs types de donnée
  int32, double, bool, 2D array, ..
- La plupart des fonctions/nœuds sont polymorphiques
- LabVIEW peut faire un conversion automatique de type (point rouge)




```
z = x + y
```

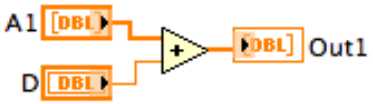
```
for (i=0;i<sz(A);i++)
   C[i] =  A[i] + b;
```

```
z = convtoDbl(x) + y
```

G-polymorphism

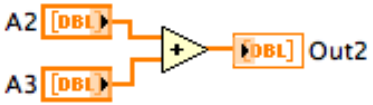
- Polymorphisme avec les tableaux
- ⚠ operations are performed element by element, not vector/matrix as in matlab!

A1[] = {1, 2, 3}
D = 2



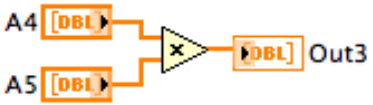
Out1[] = {3,4,5}

A2[] = {1, 2, 3}
A3[] = {5, 6}



Out2[] = {6,8}

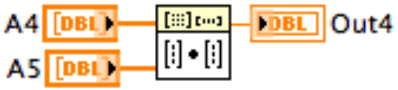
A4[] = {1, 2, 3}
A5[] = {4, 5, 6}



Out3[] = {4,10,18}

≠

A4[] = {1, 2, 3}
A5[] = {4, 5, 6}

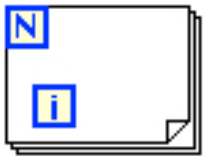


Out4[] =32

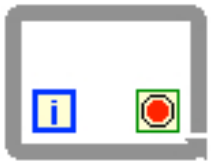
```
for (i=0;i<Size(A1);i++)
    Out1 = A1[i] + D;

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

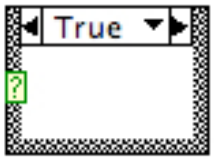
G - structures



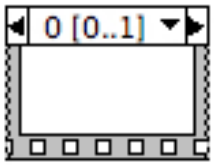

```
for (i=0;i<N;i++)
{
}
```



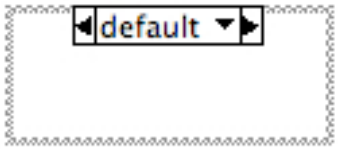

```
i:=0;
do {
} while (cond;
i++)
```



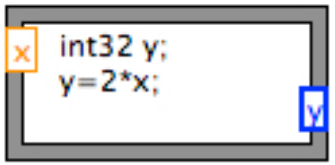

```
switch(cond) {
   case:
     break;
   default
}
```



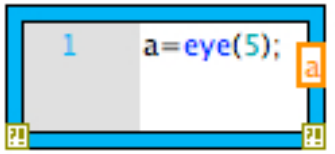

```
Sequence1;
```



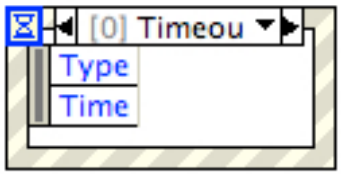

```
#ifdef cond
#endif
```



Formula node



Mathscript node

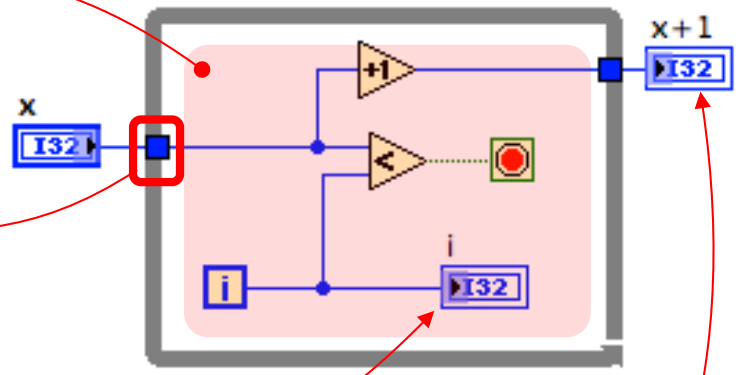


Event node

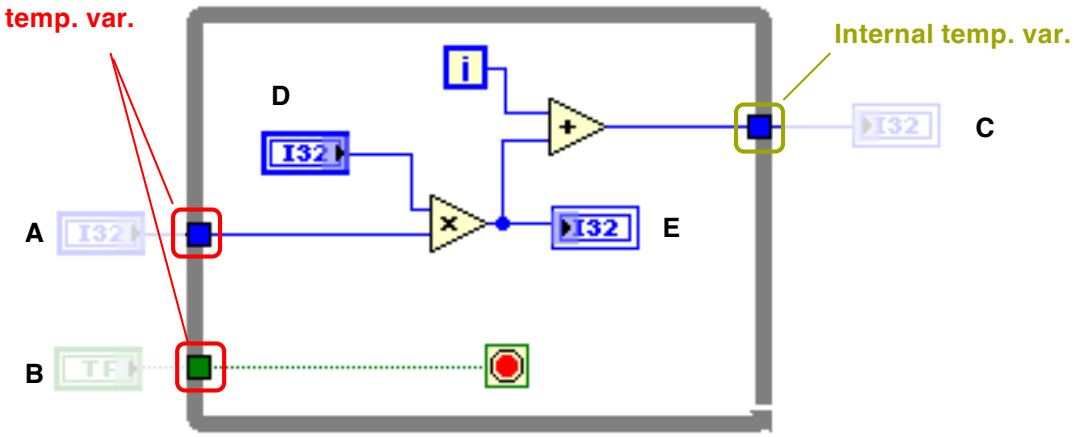
L'intérieur des structures définissent l'équivalent d'un bloc C/C++ { … }

G - structures

- Le contenu des structures définissent l'équivalent d'un bloc C/C++ { … }
- Les données (fils) qui entrent dans les structures sont évalués aux limites de la structure
- Un *indicator/control* à l'intérieur d'une structure sera évalué/mis à jour à chaque itération
- Un *indicator/control* à l'extérieur d'une structure sera évalué/mis à jour avant ou après l'exécution de la structure



G – structures + data flow




```
int tmpA = A;

int tmpB = B;

int tmpC = n/a;

do {

    E = D * tmpA;

    tmpC = E + i;

  } while (!tmpB);

C = tmpC;
```

Une structure (loop, sequence, case, etc.) est l'équivalent d'un bloc { } en c:

Elle définit la portée (scope). Les données entrantes sont comme copiées dans des variables temporaires (ou paramètres d'une fonction), et les données sortantes sont copiées depuis des variables internes temporaires.

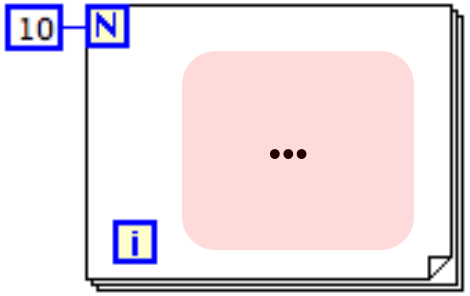
Demo for loop

- Inside/outside the loop
- I, N
- Fast 10000000 iterations
- Array indexing out
- Array indexing in, who wins ?
- 0 x loop
- Conditionnal stop
- Conditionnal add indexing

Boucle - for

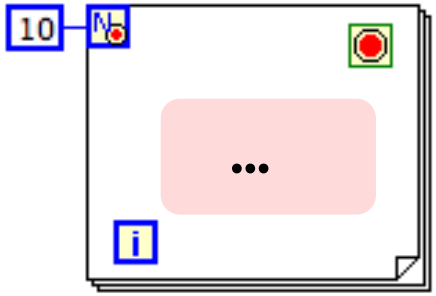
Exécute le contenu un nombre de fois donné

- i de 0 à9 (N - 1)



L'exécution peut être stopée (≥ LV 8.6)

- Arrêt si ◉ est vrai

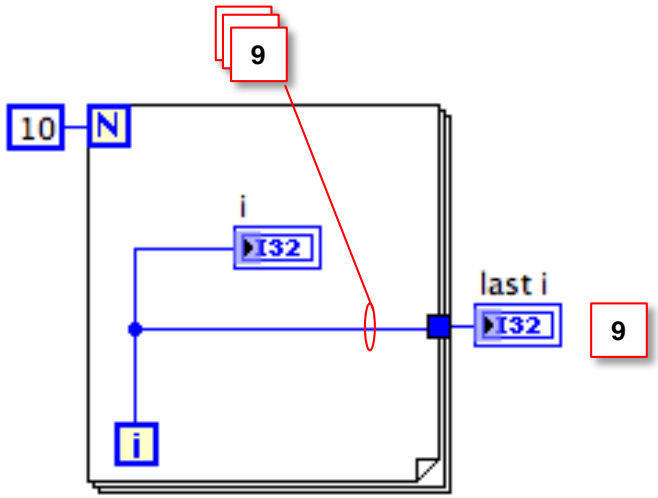



```
for (i=0;i<N;i++)
   {
    …
   }
```

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

Boucle - for

Exécute le contenu un nombre de fois donné



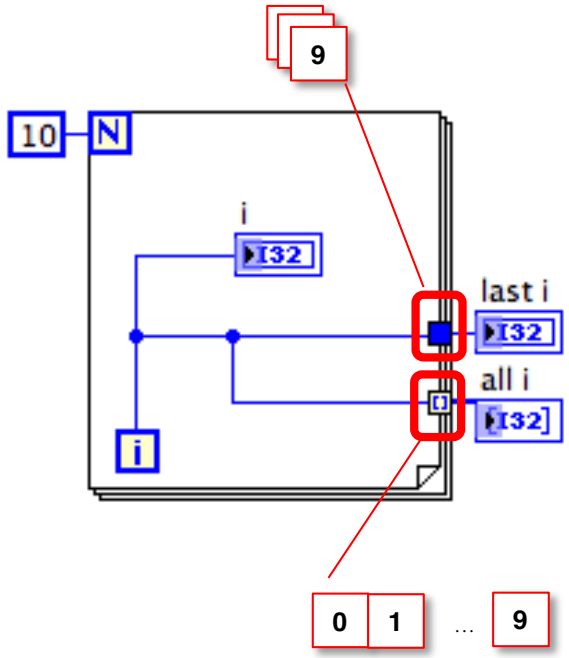

```
N=10;
for (i=0;i<N;i++)
   {
     i;
   }

last_i = i;
```

i *va de 0 à 9*
*Le dernier i (*last i*) vaut 9*

Boucle - indexing

Auto indexing

La boucle for *sauve en interne toutes les valeurs de i*



i *va de 0 à 9*
*Le dernier i (*last i*) vaut 9*
all_i = *[0,1,2,3,4,5,6,7,8,9]*

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

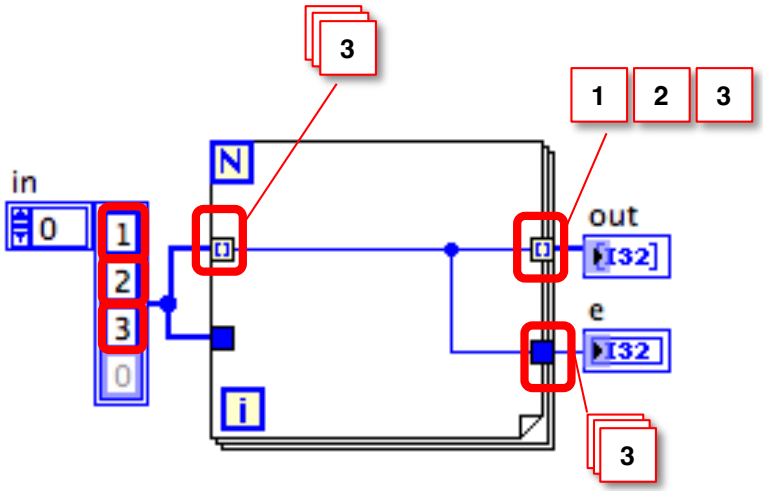
Boucle – auto-indexing

Auto indexing

- N nombre d'itérations definie par la **taille** du tableau d'entrée (in:3)

- auto indexing
- no auto-indexing

- Accède au i^{ème} élément du tableau d'entrée



- out[] = {1, 2, 3}
- e = 3
- i = 2, N = 3

```
for(i=0;

i<sizeof(in[]);
        i++)
   {
     out[i] = in[i];
     e = in[i];
   }
```

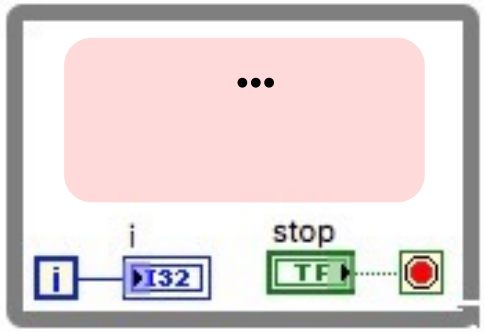
Demo While loop

- Inside-outside
- I, stop
- Double stop inside + outside
- SR, val in/out of the border
- Multiple SR
- Init SR (+1) with 0 or not connected

Loops - while

Execute le contenu jusqu'à ce que la condition soit valide (v ou F)

- La boucle while est exécutée jusqu'à ce que le bouton "STOP" soit clické
- i va de 0 à 2^{32} puis reste à cette valeur (2147483647)

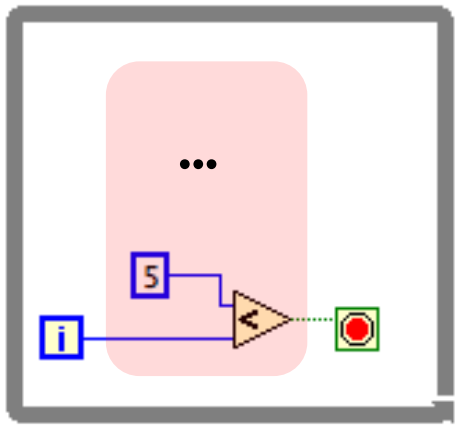



```
i = 0;
do { i++;
     …
   }
  while(!stop);
```

Loops - while

Execute le contenu jusqu'à ce que la condition soit valide (v ou F)

- La boucle while est exécutée 7 x
- i va de 0 à 6

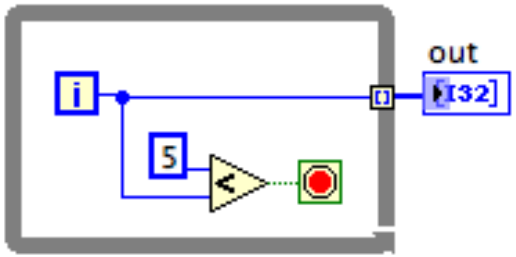



```
i = 0;
do {
    …
  }
  while(5 < i; i++);
```

Loops - while

Exécute le contenu jusqu'a ce que la condition soit atteinte

- La boucle *while* est exécutée 7 fois
- 5 < i est vrai quand i = 6
- i va de 0 à 6



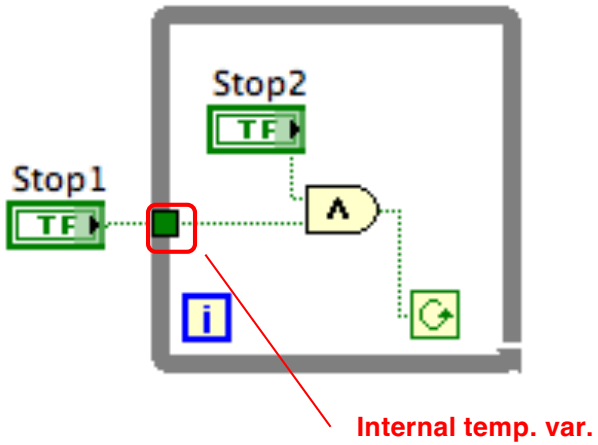
- out[] = {0,1,2,3,4,5,6}

```
i  = 0;
do {
     out[i]= i;
    }
  while(5 < i; i++);
```

Loops - while

Les fils sont évalués à l'entrée de la boucle, l'intérieur de la boucle est similaire à un bloc en c

- Stop1 est évalué une fois avant l'entrée dans la boucle
- Stop2 est évalué à chaque itérations de la boucle



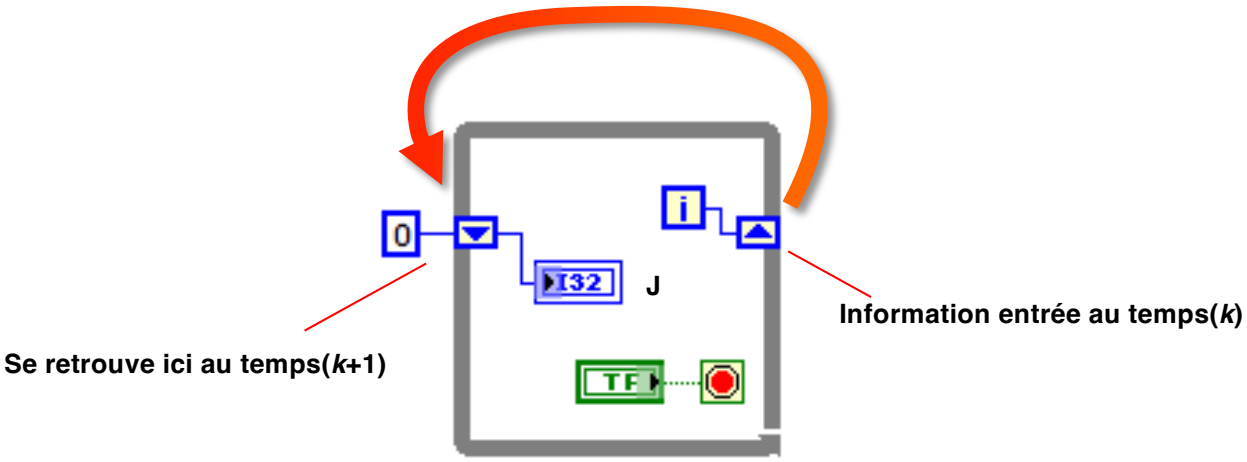
- *La boucle s'arrête après une itérations si Stop1 est False*
- *Si Stop1 est True, la boucle s'arrête quand Stop2 est False*

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

Dataflow & Shift register

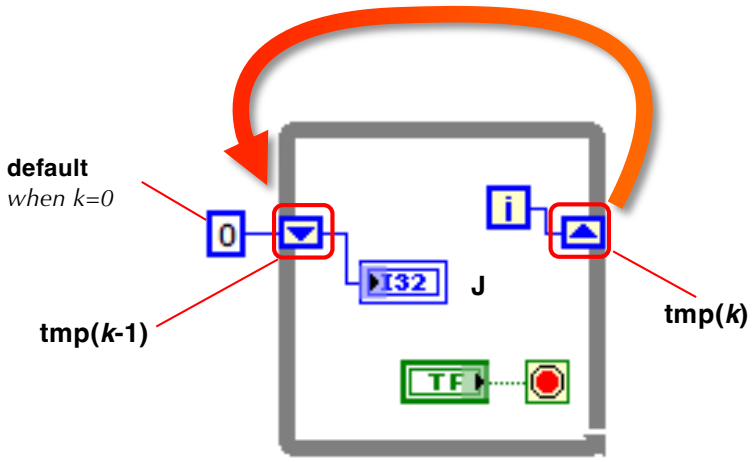
Comment faire pour mémoriser de l'information à l'intérieur d'une boucle ?

-> ***Shift register*** 



Dataflow & Shift register

Les shift registers sont des variables dépendantes du temps qui mémorisent la (les) valeur(s) précédente(s)

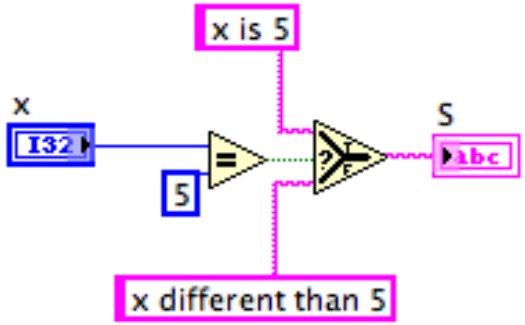


$J = \mathbf{0}, 0, 1, 2, 3, \ldots$

```
int tmpJ_k_1 = 0; // default
 // tmpJ_k_2 = 0;
int tmpJ, J, Stop;
do {
    J = tmpJ_k_1;
    tmpJ = i;
    tmpJ_k_1 = tmpJ;
  } while (!Stop);
```

Conditional - if

- Si la condition est True
  passer la valeur connectée à T
  sinon
  passer la valeur connectée à F

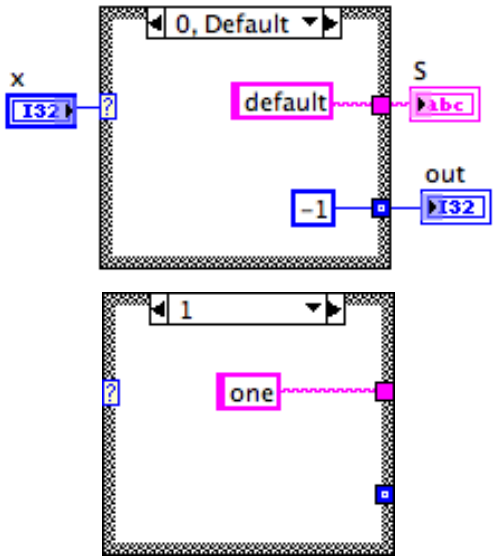



```
if (x==5)
     S = “x is 5”;
   else
     S = “x different
                than 5”;
```

- Si x vaut 5 -> S = “x is 5”, sinon S =“x different than 5”

Conditional - case

- Les *case structures* sont similaires au *switch* en C/C++
- ■ indique que tous les cas sont définis
- ▫ indique que certains cas ne sont pas définis et que la valeur par défaut sera utilisée

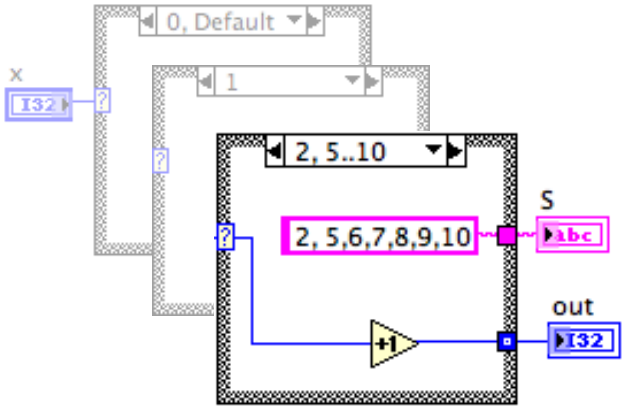



```
out = -1; //default
val.
switch (x)
     case 1:
       S="one";
       break;
     case 0:
     default:
       S = "default";
       out = -1;
```

- *S= "one"* quadn *x==1*, and *S="default"* pour toutes les autres valeurs de x
- *out = -1* or *0*

Conditional - case

- Le type du case [?] s'adapte automatiquement au format du fil connecté
- Les cas (*cases*) peuvent contenir des intervalles avec ".." ou séparé par des ","
- Chaque *case* exécute un bloc de code spécifique

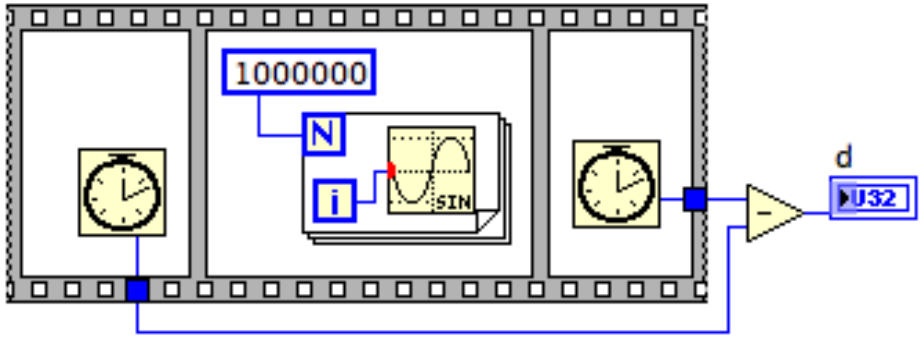


- Quand *x vaut [2,5,6,7,8,9,10]*
  S= "2,5,6,7,8,9,10"
  out = x+1

```
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

- Il est possible de forcer l'ordre d'exécution de LabVIEW à l'aide de séquences
- A éviter, car cela enlève à LabVIEW la possibilité de générer de code performant!
- Usage principal: mesure du temps d'exécution

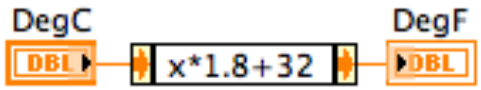


- Dans l'exemple ci-dessus *d = 4 [ms] sur un Mac*

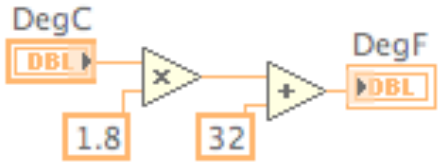
```
t1=millisec();
for (i=0;i<1000000;i++)
     sin(i)
t2=millisec();

d= t2 - t1;
```

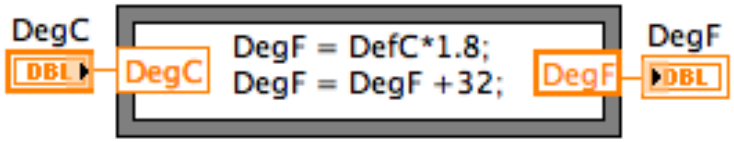
Expression/Formula node

- Permet de définir un calcul sous la forme de texte
- *Expression* node: une seule ligne de calcul
- *Formula* node: lignes multiple, code proche du C



expression node



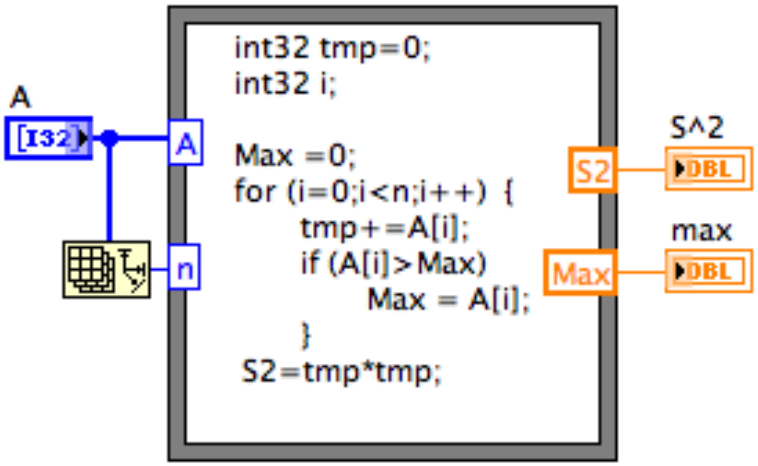


formula node

```
DegF = DegC * 1.8 + 32;
```

Formula node

- La syntaxe très proche du C
- Le code est compilé et exécuté en un bloc
- Le code ne peut pas être changé durant l'exécution, employer la *toolbox Gmath* si vous avez besoin de changer votre code



- *A=[1, 3, 2]* with *n=3*
- *S^2 = 36*
- *max=3*

```
int32 A[]={1, 3, 2};
int32 n= sizeof(A);

  int32 tmp=0;
  int32 i;

  Max =0;
  for (i=0;i<n;i++){
      tmp+=A[i];
      if (A[i]>Max)
         Max = A[i];
      }
  S2=tmp*tmp;

S^2 = S2;
max = Max;
```

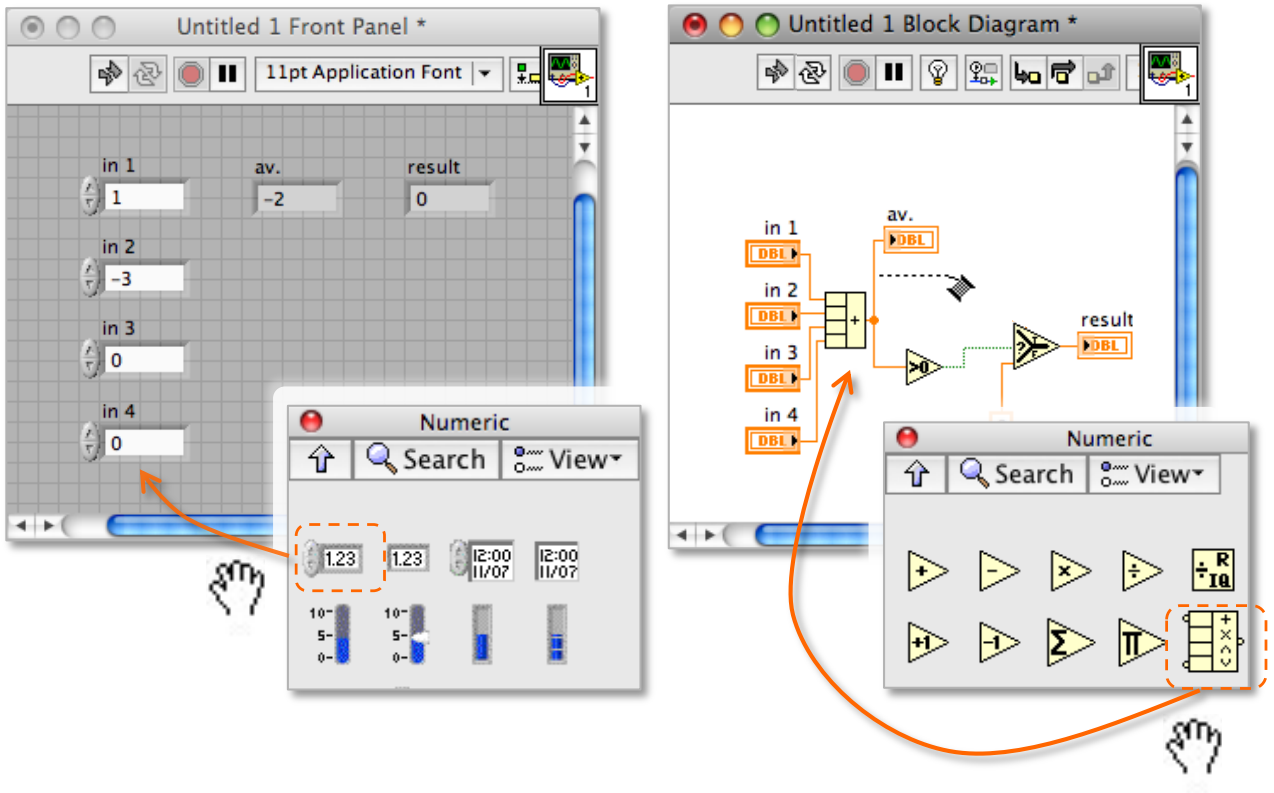
Mon premier VI

- Dessiner l'interface utilisateur (controls & indicators)
- Câbler le programme (diagram)
- Tester et debugger
- Ajouter la documentation du VI
- Définir le prototype/interface (connector pane)
- Dessiner l'icone

Mon premier VI - Moyenne

Spécifications:

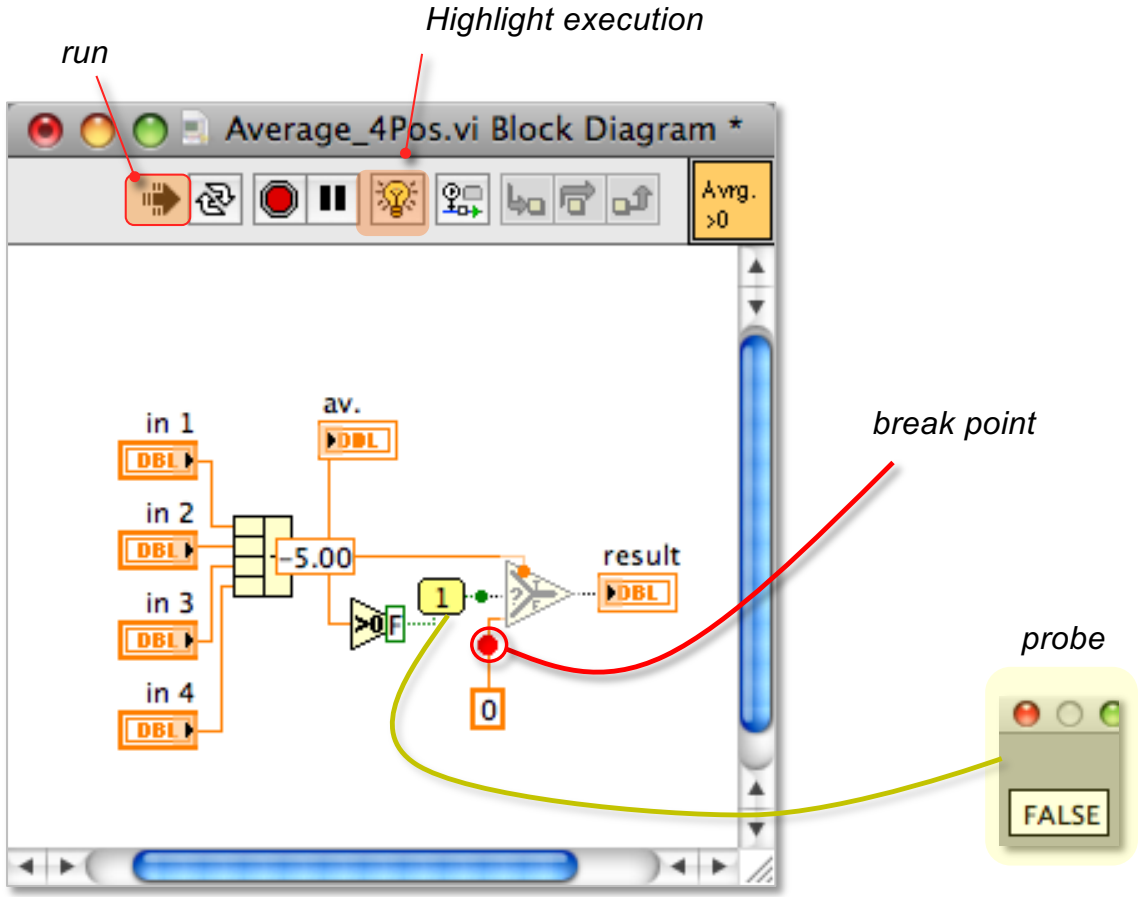
- Calculer la moyenne de 4 valeurs
- Si le résultat est plus petit que 0, le forcer à 0




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

Mon premier VI - Debug

- Test and debug



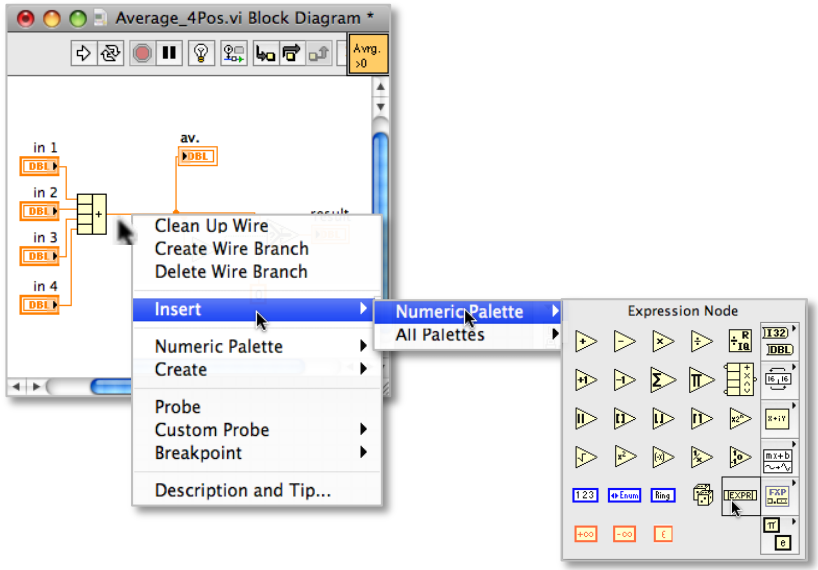

```
function Average_4Pos{
  av =
(in1+in2+in3+in4);
  if (av>0)
     result = av;
   else
     result = 0;
 }
```

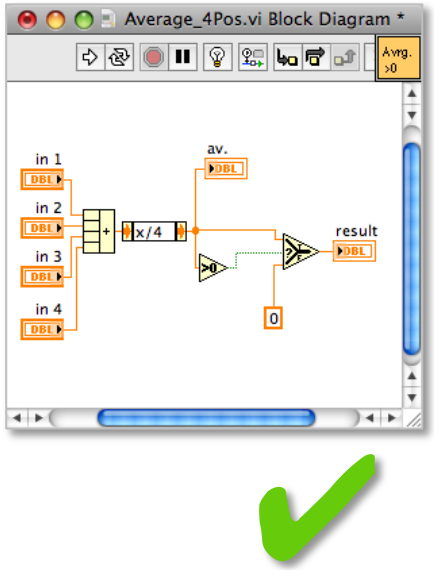
Variables

av>0	FALSE

Mon premier VI - Modification

- fix and test again

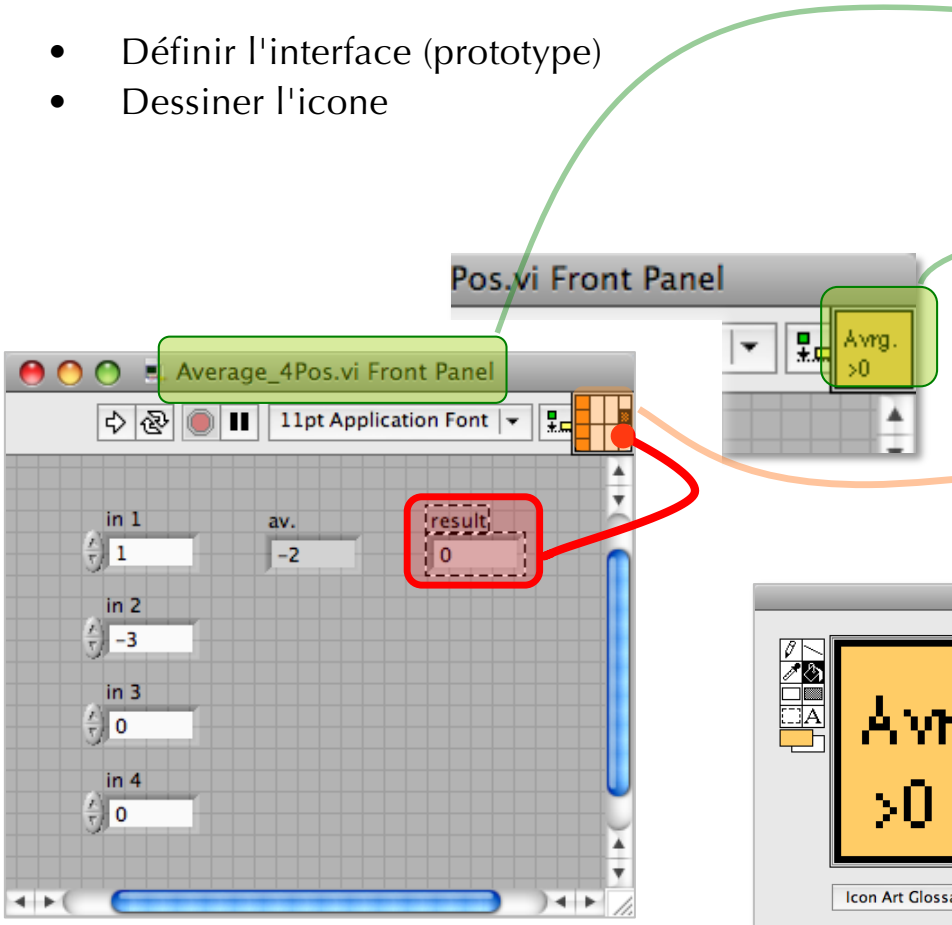


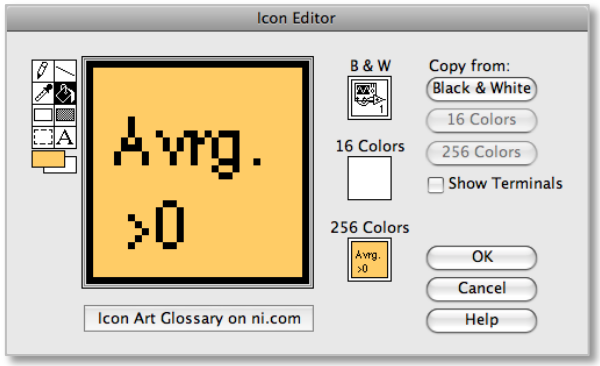



```
function Average_4Pos{
  av = (in1+in2+in3+
                  in4)/4;
  if (av>0)
     result = av;
   else
     result = 0;
 }
```

Mon premier VI -> fonction()

- Définir l'interface (prototype)
- Dessiner l'icone



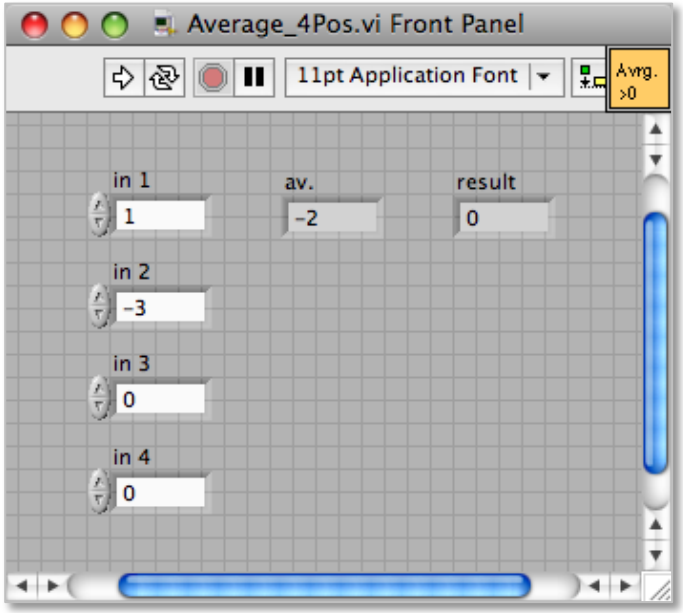


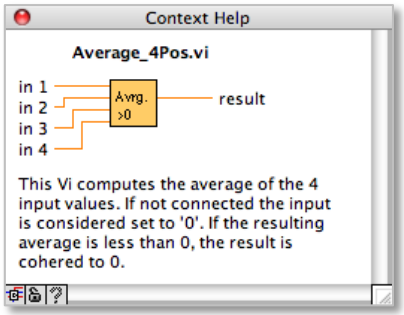

```
double
function Average_4Pos(
    double in1,in2,in3,in4){
  av = (in1+in2+in3+
                  in4)/4;
  if (av>0)
     result = av;
   else
     result = 0;

  return result;
}
```

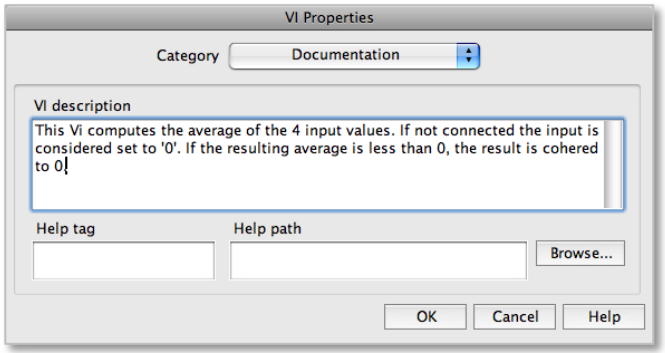
http://www.ni.com/devzone/idnet/library/guidelines_for_creating_icons.htm

Mon premier VI - Documentation

- Ecrire la documentation
- Elle apparaîtra dans la fenêtre d'aide (context help window)



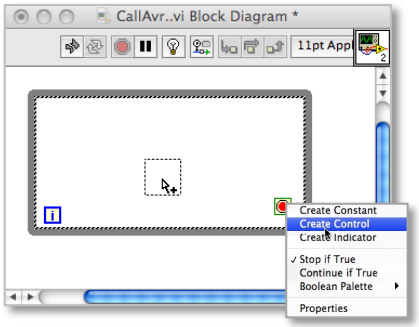


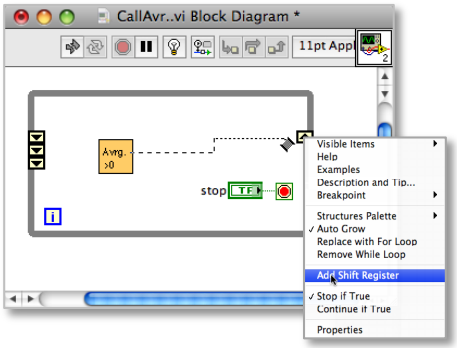


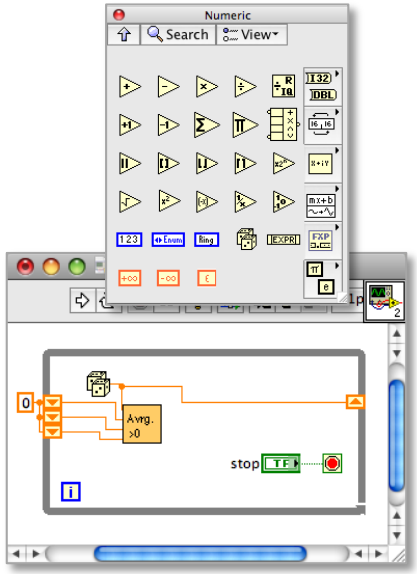

```
/*
 This Vi computes the average of the
4 input values. If not connected the
input is considered set to '0'. If
the resulting average is less than
0, the result is cohered to 0.
*/
double
function Average_4Pos(
    double in1,in2,in3,in4) {
  av = (in1+in2+in3+
                  in4)/4;
  if (av>0)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

  return result;
}
```

Mon premier VI – Appel

- Créer un nouveau vi
- Glisser/déposer the **Avrage_4Pos.vi** depuis l'OS (the finder), le *connector pane*, ou la palette (select a vi …)
- Créer un control pour le bouton **Stop**
- Ajouter un *shift register*, initialiser le à 0
- Connecter le générateur de nombre aléatoire



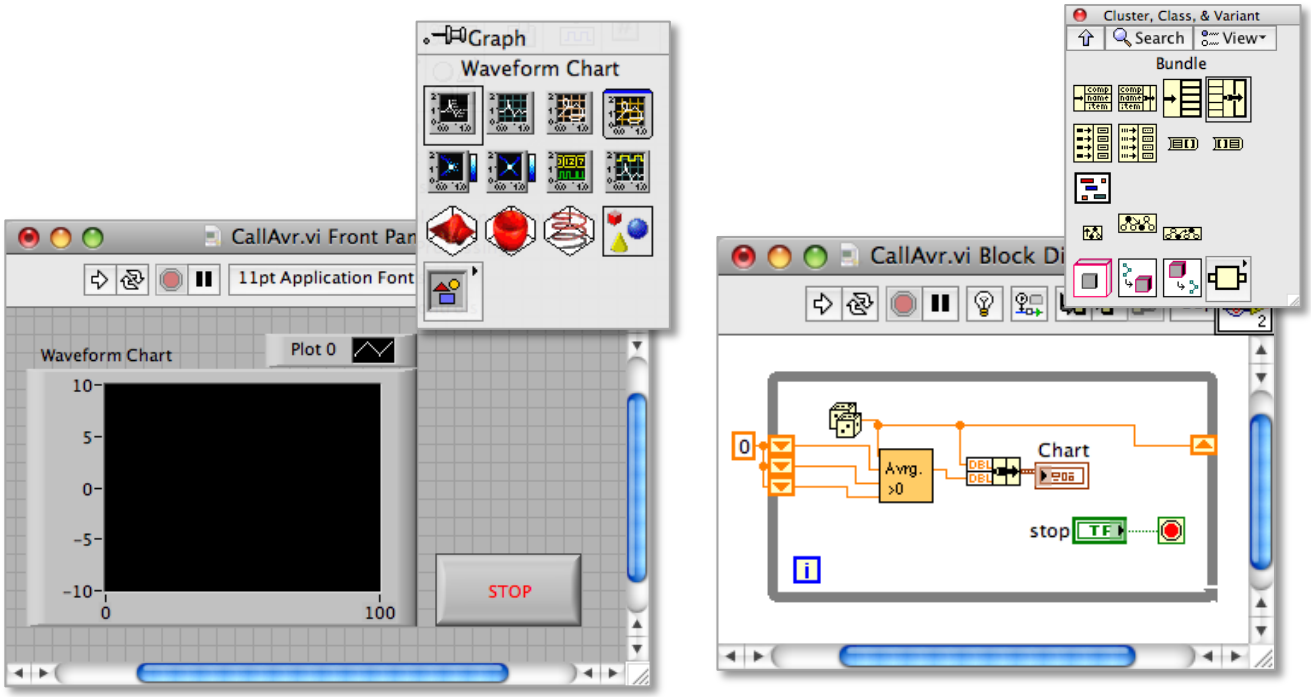





```
void function CallAvr(void) {
  sr1=0;
  sr2=0;
  sr3=0;
  While (!Stop) {
     sr=rand();
     sr1=sr;
     sr2=sr1;
     sr3=sr2;
     Avrage_4Pos(sr,sr1,
            sr2,sr3);
     };
}
```

Mon premier VI – Appel

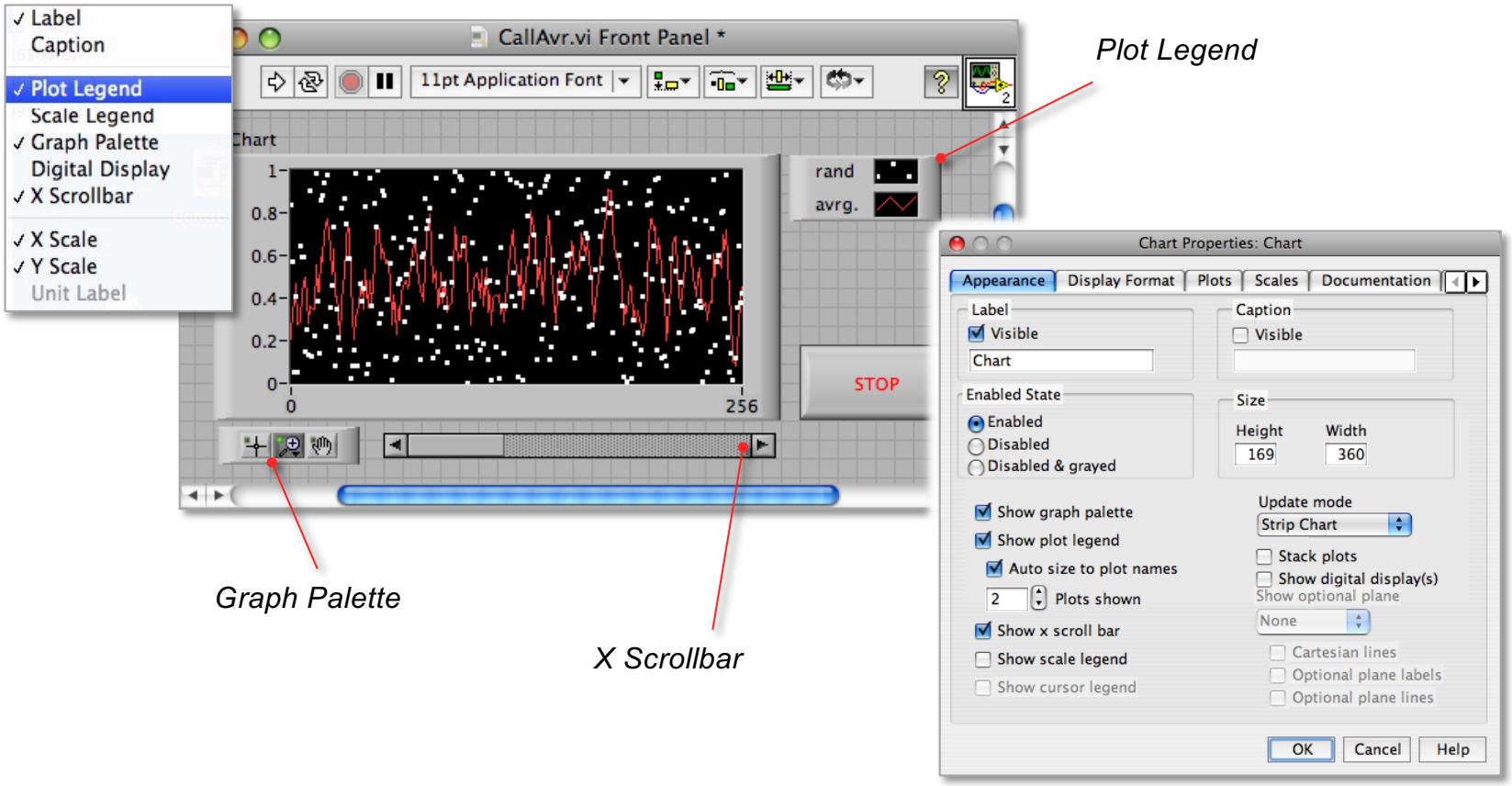
- Ajouter un *Waveform chart* pour afficher les resultats
- Utiliser un *bundle* pour combiner les 2 valeurs à afficher




```
void function CallAvr(void) {
  float r;
  sr1=0;
  sr2=0;
  sr3=0;
  While (!Stop) {
     sr=rand();
     sr1=sr;
     sr2=sr1;
     sr3=sr2;

r=Avrage_4Pos(sr,sr1,
              sr2,sr3);
     plot(Chart,sr,r);
     };
}
```

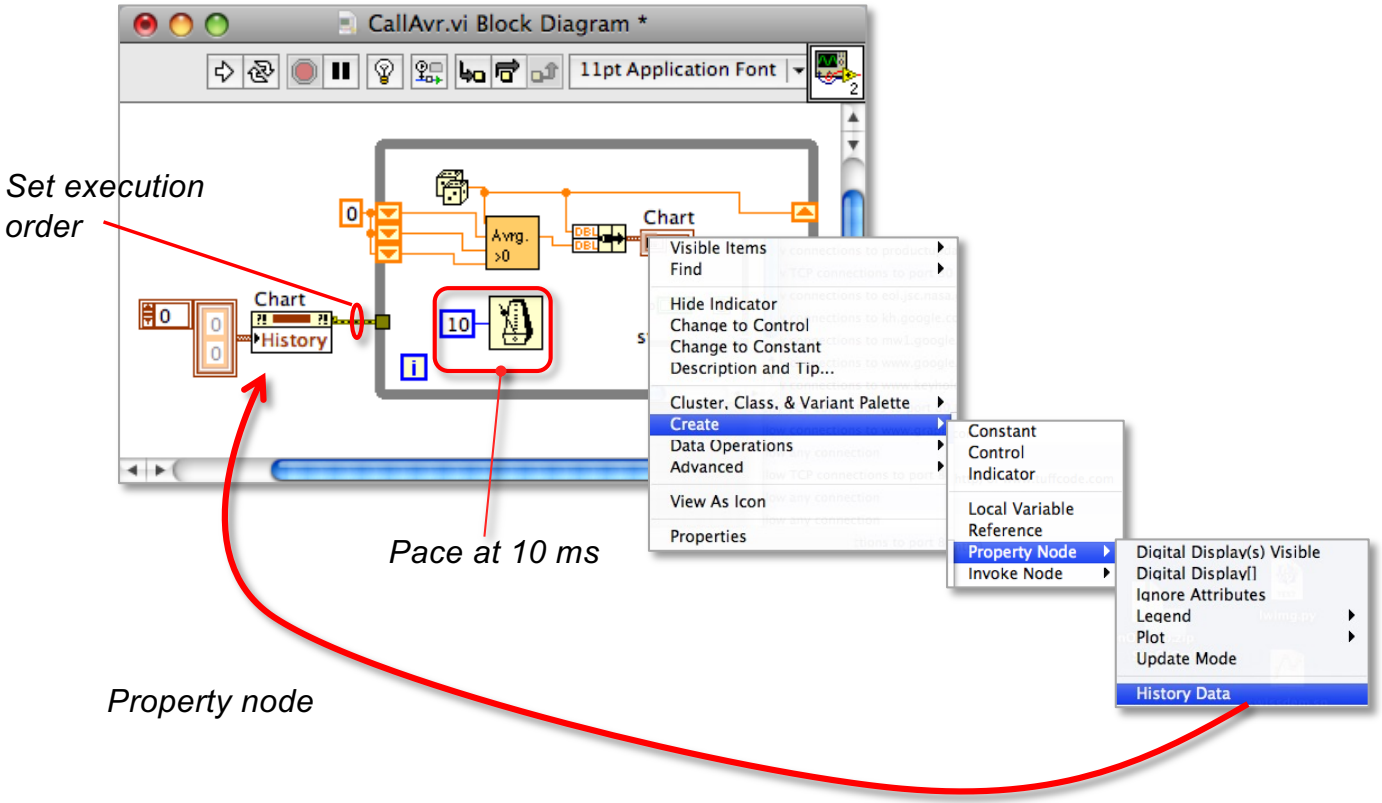
Mon premier VI – Appel

- Customisez l'affichage
- Pour cela utiliser le dialogue *properties* ou le menu contextuel (popup menu)



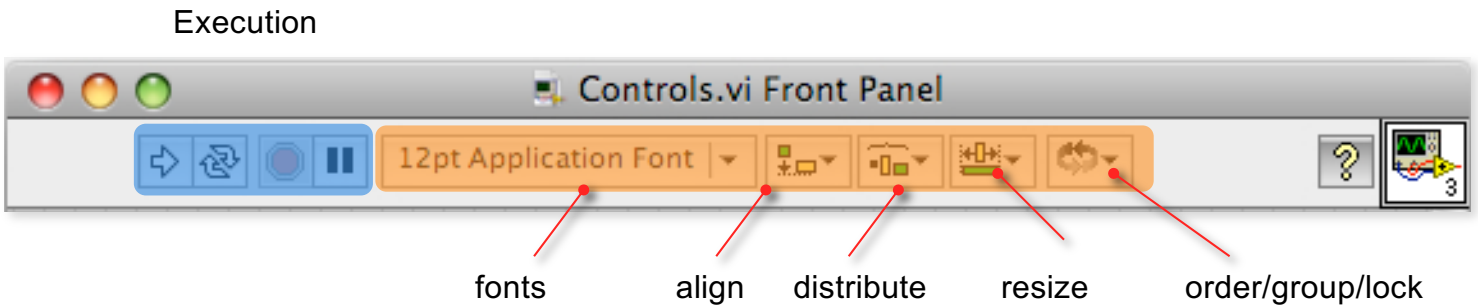
Mon premier VI – Appel

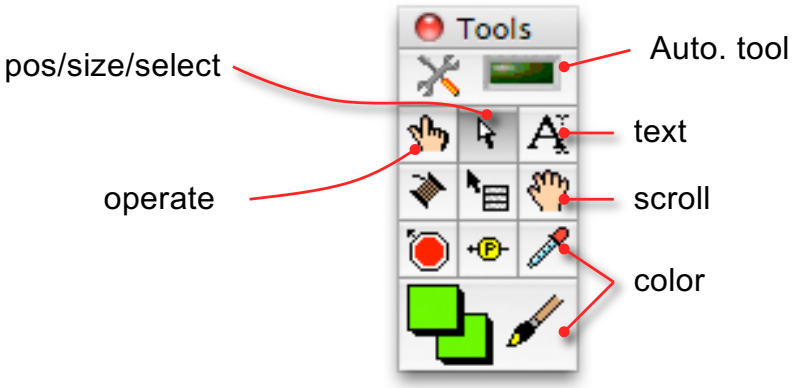
- Rythmer l'exécution (toutes les 10 ms)
- Initialiser le *Chart* via un ***property node***

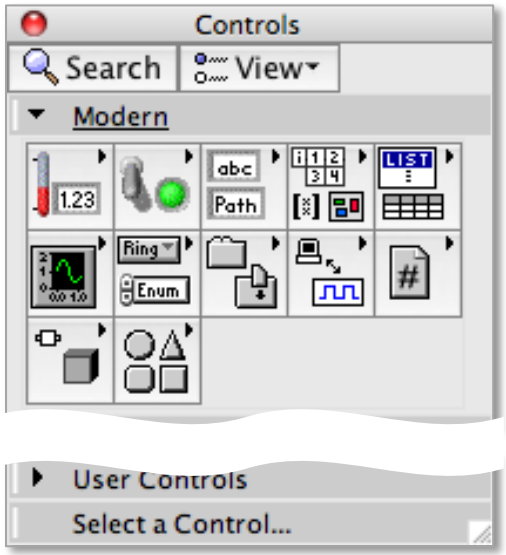



```
void function CallAvr(void) {
  float r;
  struct {double:a,b} BLD2;
  BLD2 _a[]={};
  sr1=0;
  sr2=0;
  sr3=0;
  SetHistory(Chart,_a);
  While (!Stop) {
     sr=rand();
     sr1=sr;
     sr2=sr1;
     sr3=sr2;
     r=Avrage_4Pos(sr,sr1,
                   sr2,sr3);
     plot(Chart,sr,r);
     WaitUnitlNextMS(10);
     };
}
```

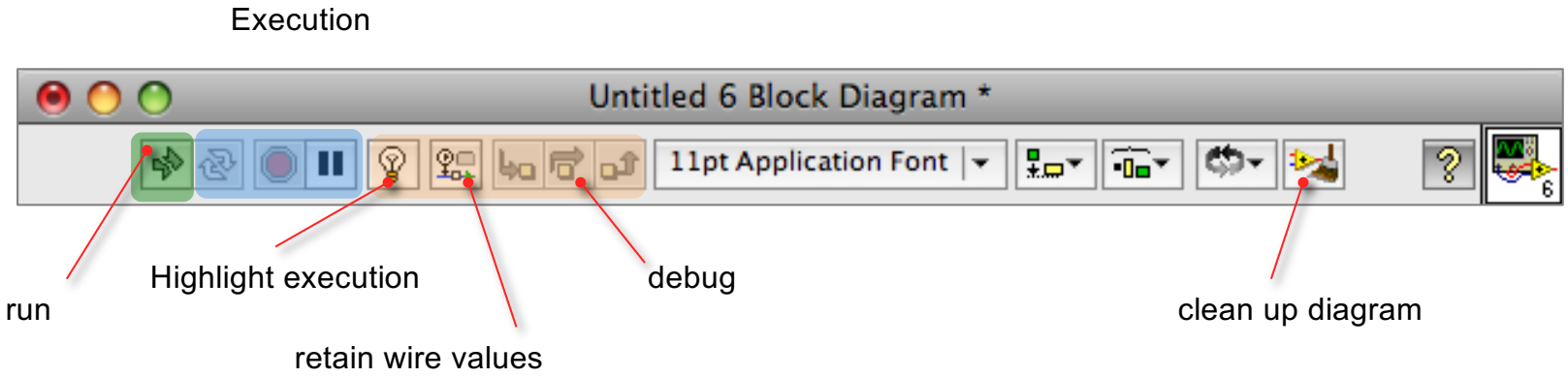
Tools bars - front panel

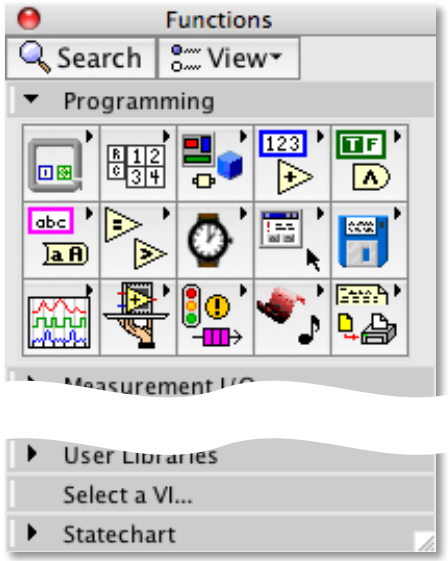






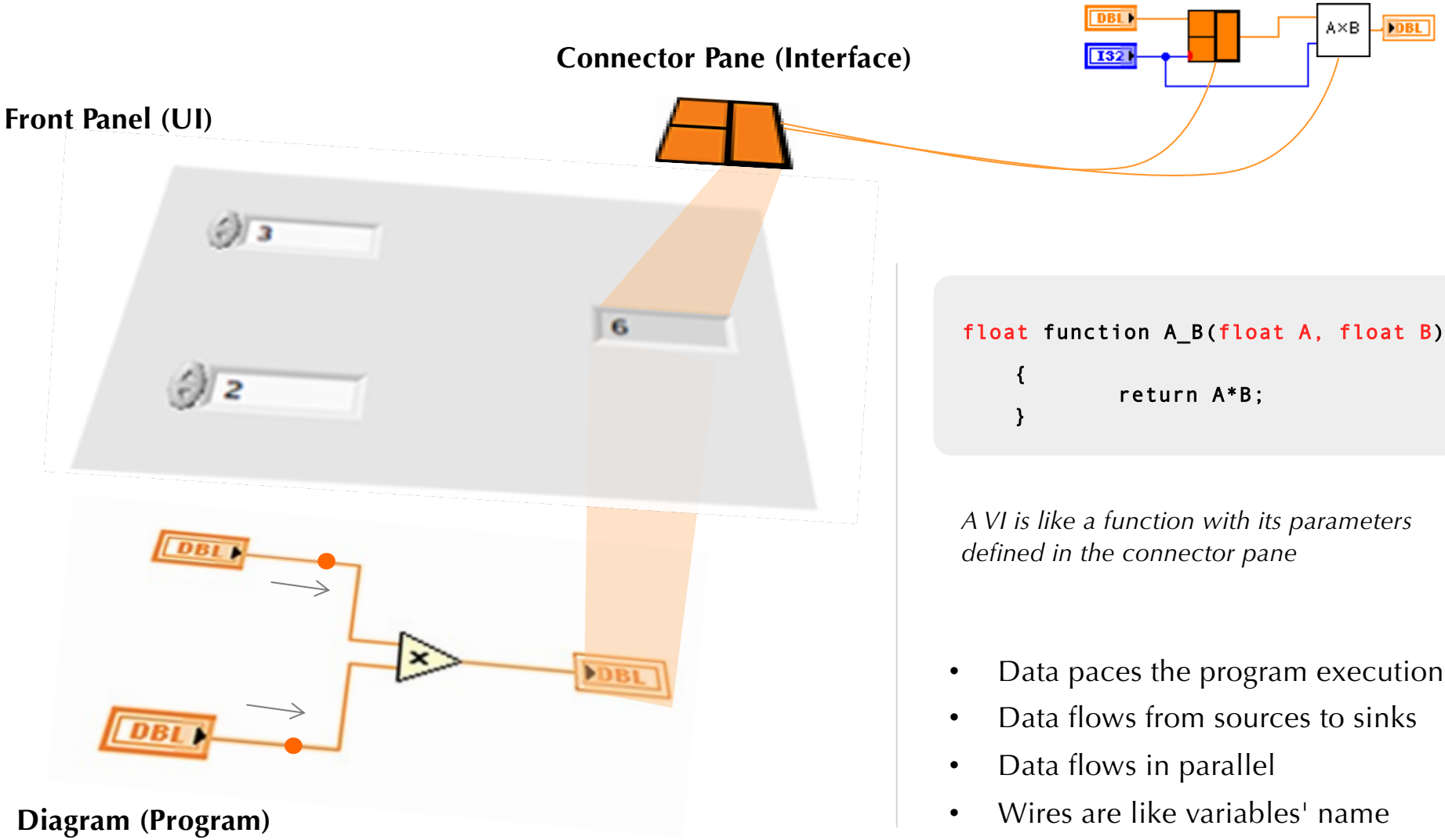
Tools bars - diagram





E	*front panel <-> diagram*
TAB	*next tool*
SPACE	*operate* <-> pos/size/select
Alt + drag	*duplicate*

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)