LabVIEW - generalità

Laboratory Virtual Instrument Engineering Workbench (1987)

Ambiente di sviluppo per applicazioni orientate a

- acquisizione dati;
- gestione strumenti di misura;
- analisi ed elaborazione dei segnali.

Principali caratteristiche:

- basato su linguaggio grafico a oggetti (linguaggio G);
- programmi realizzati in forma di diagrammi a blocchi;
- caratteristiche comuni ad altri linguaggi:
  + tipi di dati e operatori
  + strutture di controllo flusso di programma (for, while, . . .);
  + metodi di debug, funzioni di libreria, . . .
  + . . .

LabVIEW - generalità (2)

- Nei linguaggi tradizionali l'ordine di esecuzione è *normalmente* dato dall'ordine in cui le istruzioni sono scritte

⚠ Nel linguaggio G l'ordine di esecuzione è determinato dal flusso di dati

- Ciascuna istruzione viene eseguita non appena sono disponibili i dati di cui necessita.

  $\Rightarrow$ è possibile eseguire operazioni in parallelo!

- il codice grafico viene tradotto in linguaggio C e compilato
- è possibile inserire parti di codice scritto in lunguaggio C/C++.

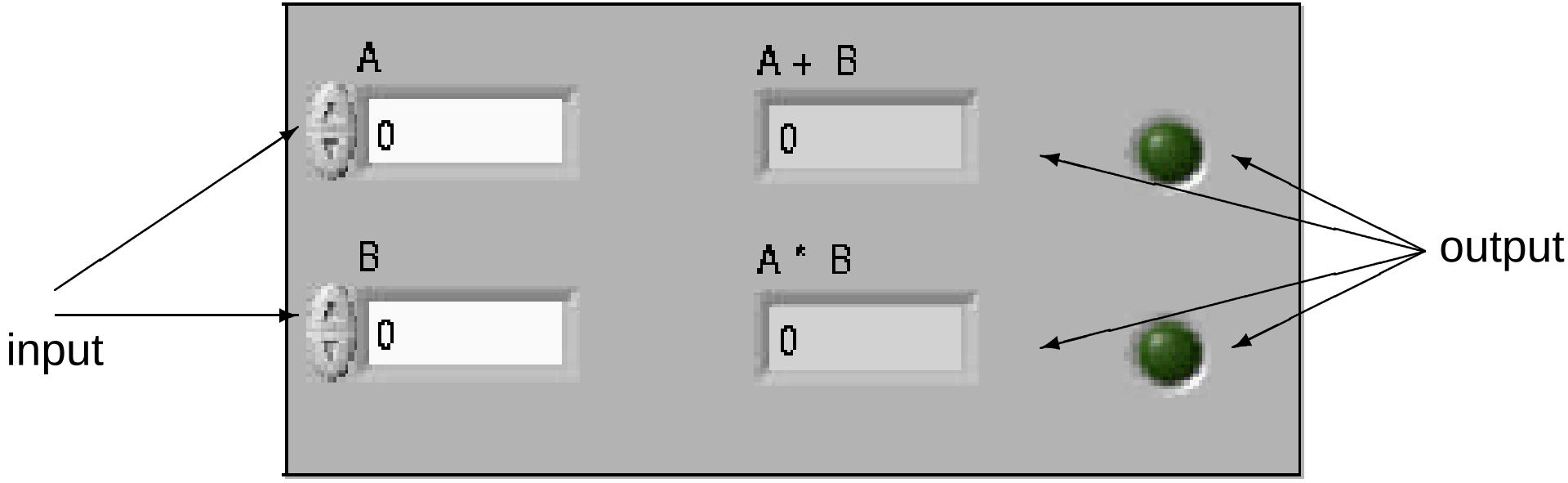
LabVIEW - generalità (3)

I programmi generati vengono detti “strumenti virtuali” (.VI):

rumento: i programmi appaiono all’utente come strumenti di misura, con un proprio pannello frontale (definito in fase di programmazione)

virtuale: l’interazione avviene con un programma (e non con uno strumento)

Pannello frontale:



Composizione di un programma LabVIEW

Un programma LabVIEW è composto da:

- Pannello frontale (Front Panel); è l'interfaccia grafica di ingresso-uscita, costituita da:
  + Controllori (input): tasti, manopole, cursori, . . .
  + Indicatori (output): display numerici, spie, diagrammi, . . .
- Diagramma a blocchi (Block Diagram); è la forma in cui si presenta il codice:
  + Nodi (blocchi): effettuano le operazioni richieste dal programma (appaiono come icone);
  + Collegamenti: permettono lo scambio di informazioni tra i nodi (appaiono come fili che collegano i nodi).

- (Coppia Icona/Connettore *Icon/Connector*)

  permette di trasformare un programma in un oggetto (sub-VI) impiegabile all'interno di un altro programma VI.

LabVIEW - pannello frontale e diagramma a blocchi

Pannello frontale:

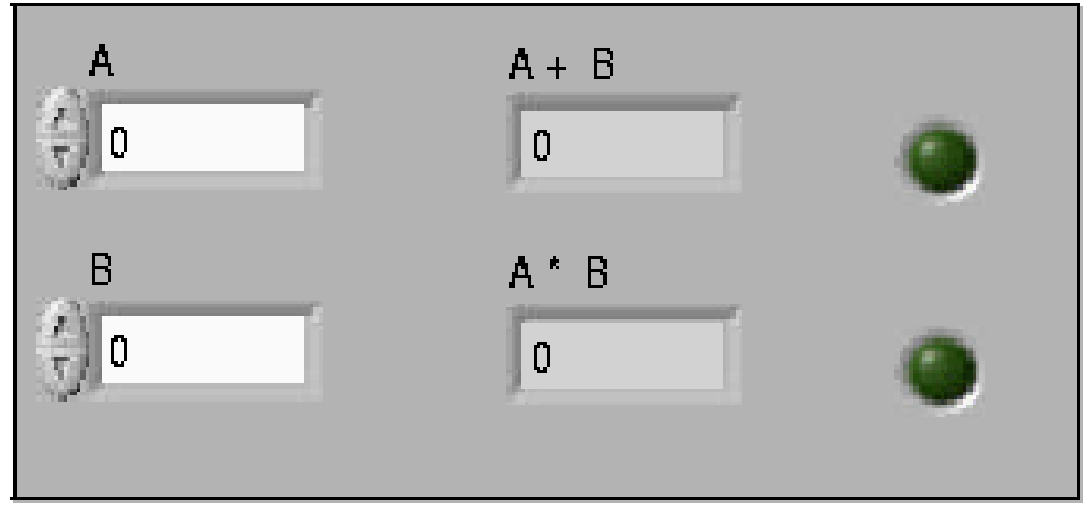
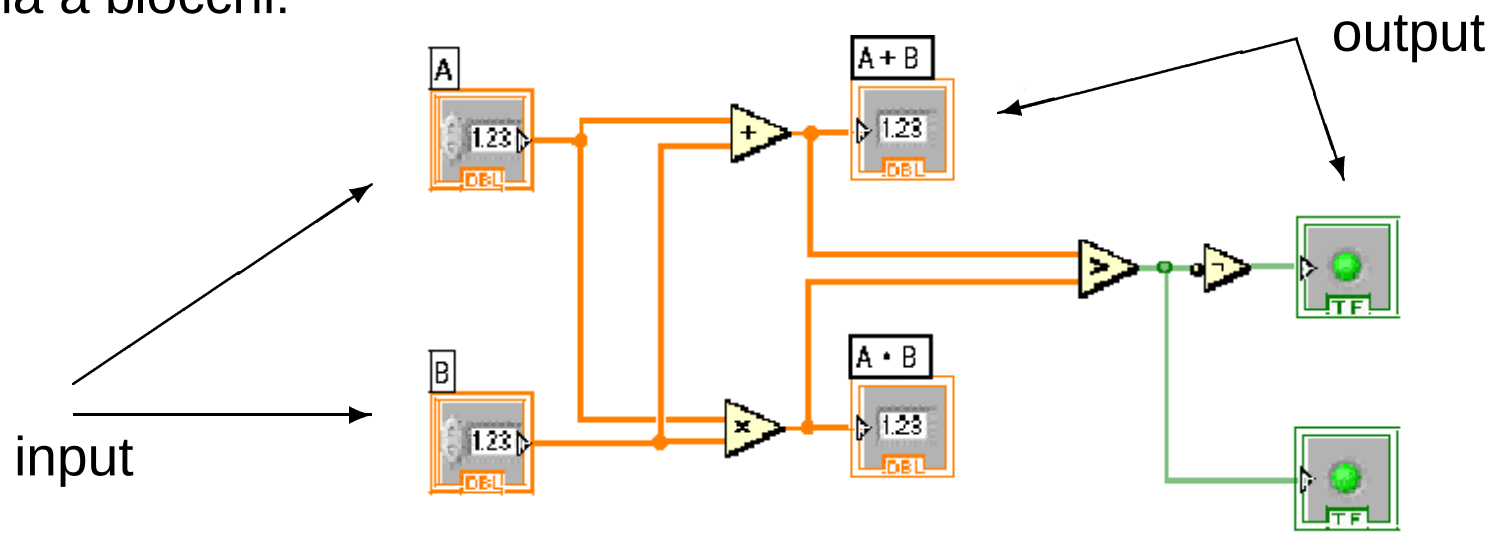
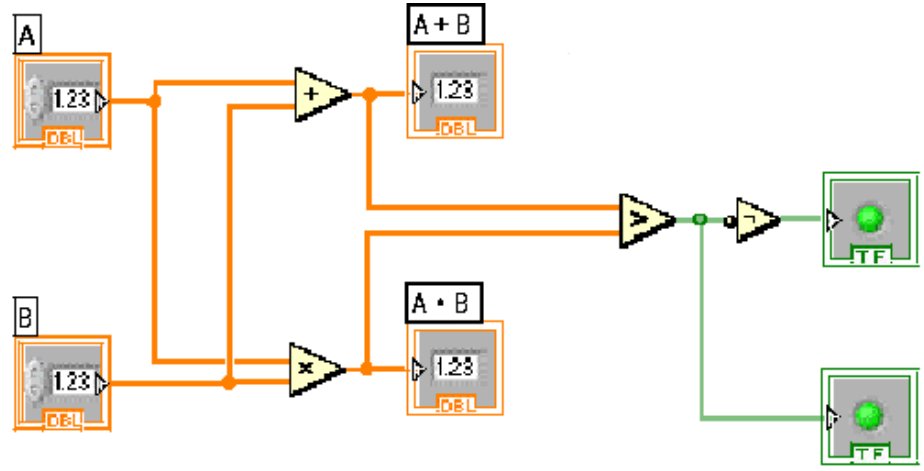


Diagramma a blocchi:



linguaggio G vs. C

```
float a,b;
int i=0,j=0;
   scanf("%f",&a);
   scanf("%f",&b);
   printf("somma=%7.3f;\n",a+b);
   printf("prodotto=%7.3f;\n",a*b);
```




```
if (a+b>a*b)
   {i=1;}
else
   {j=1;}
   printf("somma > prodotto: %d\n",i);
   printf("prodotto > somma: %d\n",j);
```

Linguaggio G (2)

Tipi di dati: (Corrispondenza tipo di dato $\leftrightarrow$ colore)

- numerici:
  - + interi con e senza segno, a 8, 16, 32, 64 bit;
  - + floating point a precisione singola (32 bit), doppia (64 bit) ed estesa (128 bit);
  - + complessi;
- booleani;
- stringa (formato ASCII);
- “list and ring”/enumerato (numero intero associato a stringa): consentono la gestione p. es. di menù a tendina;
- array;
- cluster: collezione di dati anche di tipo diverso.
- . . .

Linguaggio G (3)

Strutture di controllo

Consentono il controllo del flusso di esecuzione

- **for loop**: ripete l'esecuzione n volte (v. `cont-for.vi`);
- **while loop**: ripete l'esecuzione fino al verificarsi di una determinata condizione (v. `cont-while.vi`);
- **case**: permette l'esecuzione condizionata di una fra n porzioni di programma ("if", "select case") (v. `cont-if.vi, cont-case.vi`);
- **sequence**: permette l'esecuzione sequenziale di un certo numero di porzioni di programma; appare come insieme di fotogrammi (flat sequence/stacked sequence)
- **formula node**: permette di scrivere espressioni matematiche in modo "naturale".
- **ritardo**: attende un certo tempo prima di procedere con l'esecuzione

Linguaggio G (4)

Altre caratteristiche:

- Variabili locali e variabili globali;
- Property node;
- **Polimorfismo**: i nodi di un diagramma a blocchi si adeguano automaticamente al tipo di dati che ricevono: somma di due vettori, somma di vettore a scalare, ecc. `[polimorfismo.vi]`

Esempi ... (1a)

controlli e indicatori, scalari, vettori, polimorfismo

+ `esempio_01`: lo s.v. legge due numeri (A e B) in ingresso e ne visualizza su p.f. la somma e il prodotto.

$\Rightarrow$ modificare `esempio_01.vi` in modo che vengano calcolati la differenza e il rapporto (invece che somma e prodotto) dei dati in ingresso; `[esempio_01b.vi]`

$\Rightarrow$ completare `esempio_01.vi` con spie come esempio visto a lezione; `[esempio_01a.vi]`

$\Rightarrow$ modificare `esempio_01.vi` in modo che operi sommando due vettori; `[esempio_01c.vi]`

$\Rightarrow$ modificare `esempio_01c.vi` in modo che operi sommando un vettore a uno scalare (polimorfismo); `[esempio_01d.vi]`

$\Rightarrow$ modificare `esempio_01c.vi` in modo che mostri su p.f. la lunghezza (size) del vettore risultato; `[esempio_01e.vi]`

da fare ... (1b)

- $\Rightarrow$ modificare `esempio_01e.vi` in modo che (in più) estragga e mostri su p.f. il j-esimo elemento del vettore risultato; `[esempio_01f.vi]`
- $\Rightarrow$ modificare `esempio_01e.vi` in modo che (in più) estragga e mostri su p.f. la somma tra il j-esimo e il k-esimo elemento del vettore somma; `[...]`
- $\Rightarrow$ ... che inoltre accenda una spia rossa se il j-esimo elemento è maggiore del k-esimo e una spia verde in caso contrario; `[...]`
- $\Rightarrow$ ... che estragga e mostri su p.f. i valori minimo e massimo tra gli elementi del vettore somma e i relativi indici; `[...]`

⚠ utilizzare opportune etichette accanto agli indicatori e ai controlli

⚠ realizzare diagrammi a blocchi ordinati

⚠ evitare, per quanto possibile, incroci di fili.

da fare ... (2a)

controllo del flusso di programma (for, while, pause, if, . . .)

+ `esempio_02a`: visualizzare su p.f., a intervalli di 2 s, la radice quadrata dei numeri interi da 0 a 5.

$\Rightarrow$ modificare `esempio_02a.vi` in modo che il numero di iterazioni e il tempo di attesa siano impostabili da pannello frontale;

+ `esempio_02b`: visualizzare su p.f., a intervalli di 0.1 s, la radice quadrata dei numeri interi da 0 in poi, arrestando la procedura quando il risultato dell'operazione raggiunge il valore 10.

$\Rightarrow$ modificare `esempio_02b.vi` in modo che venga visualizzato anche l'indice corrente del ciclo;

$\Rightarrow$ modificare `esempio_02b.vi` in modo che la radice quadrata dell'indice corrente del ciclo venga visualizzata solo alla fine dell'esecuzione;

da fare ... (2b)

controllo del flusso di programma (for, while, pause, . . . + cluster)

⇒ modificare `esempio_02b.vi` in modo che l'uscita dal `while` avvenga quando si verifica la condizione `3 * i + 5 >= 20`.

⇒ modificare `esempio_02b.vi` in modo che l'uscita dal `while` avvenga quando si verifica la condizione `3 * i + 5 >= x`, con `x` impostabile da pannello frontale.

\+ `esempio_02c`: come 02b, ma all'uscita dal `while` viene visualizzato il vettore di due colonne la prima delle quali contiene i valori dell'indice (i) accumulati durante l'esecuzione, mentre la seconda i valori di $\sqrt{i}$; la condizione di uscita dal `while` è la stessa.

\+ `esempio_02d`: visualizzare su p.f., a intervalli di 1 s, i numeri interi (i) da 0 a 30; sul p.f. è acceso un LED verde se $i \leq 15$; se invece $i > 14$, il LED è spento e ad ogni incremento dell'indice viene prodotto un "bip".

da fare . . . (2c)

$\Rightarrow$ modificare `esempio_02d.vi` in modo che la condizione di discrimine tra vero e falso sia il fatto che l'indice corrente superi o meno la metà del valore di N, con N impostabile da p.f.

$\Rightarrow$ modificare `esempio_02d.vi` in modo che si accenda un LED rosso se l'indice corrente supera la metà del valore di N (con N impostabile da p.f.) o un LED verde in caso contrario.