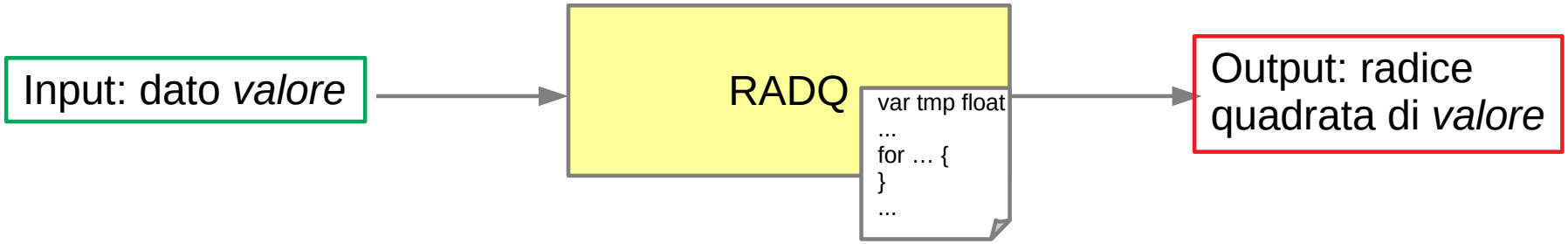
Programmazione I

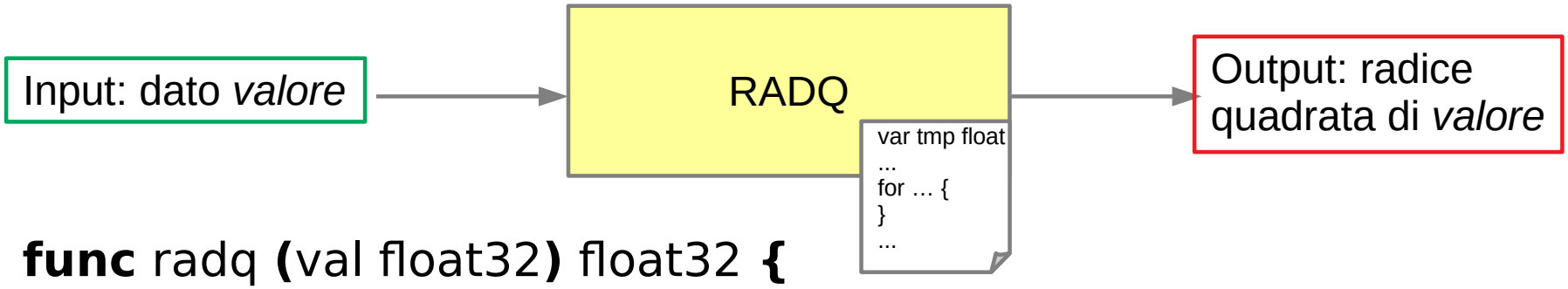
Lezione 12

Recap: il concetto di sottoprogramma

- Idea:
  - isolare parti di codice che svolgono compiti specifici
  - associare un nome e una “interfaccia”, operando *astrazione di processo*



Passaggio di parametri a sottoprogramma

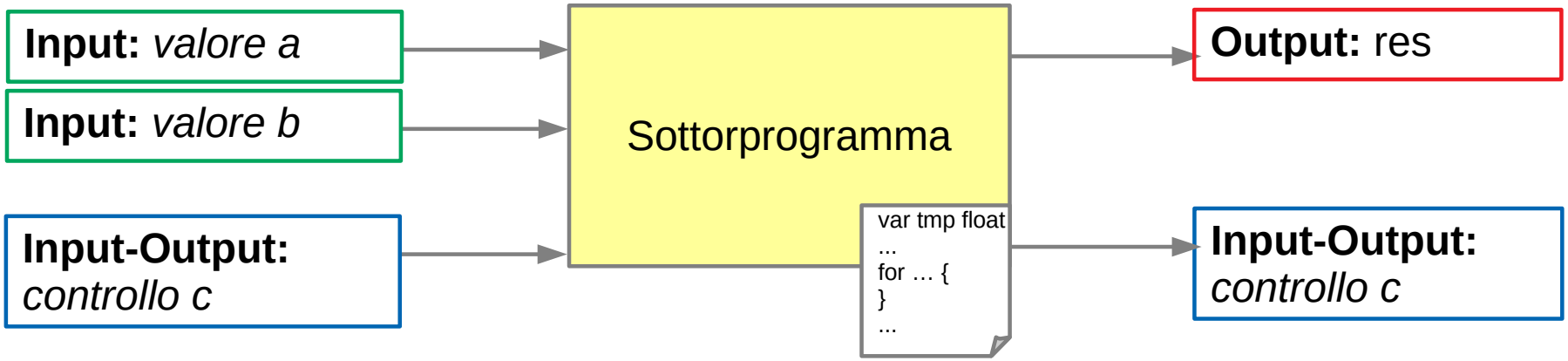



```
func radq (val float32) float32 {
    ...
}
...
var lunghezza, perimetro float32
lunghezza = 10.0
perimetro += radq(lunghezza)
```

Chiamata

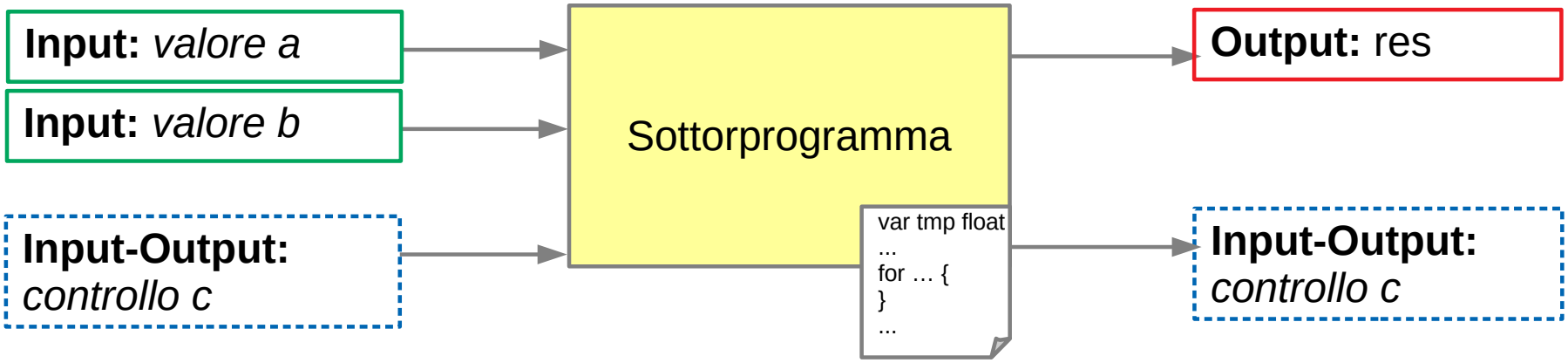
Ruolo logico dei parametri

- Ruolo dei parametri (agli occhi del programmatore): Input, Output, Input-Output



Ruolo logico dei parametri

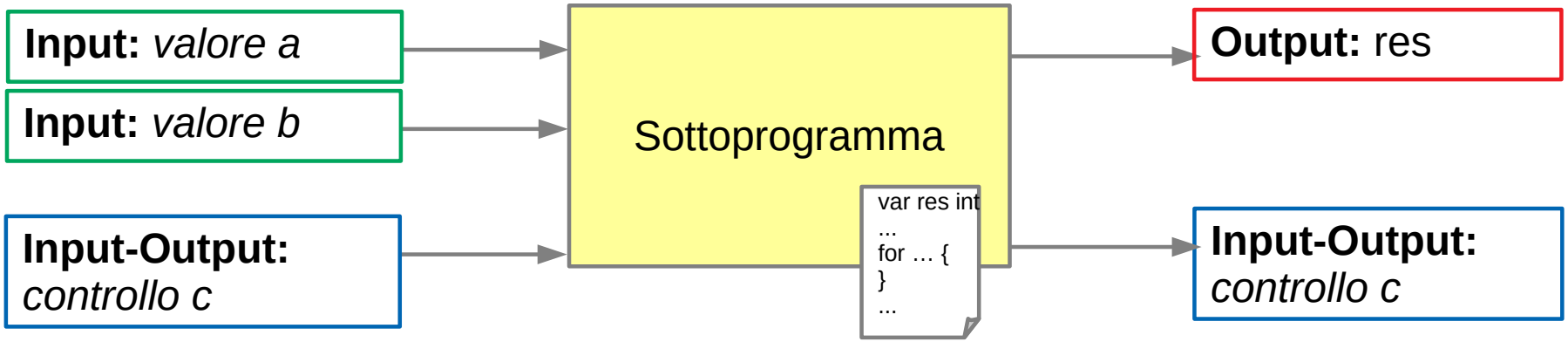
- Ruolo dei parametri (agli occhi del programmatore): Input, Output, Input-Output



- Esempio in GO: media con parametro di uscita dichiarato nella signature

Recap: il concetto di sottoprogramma

- Ruolo dei parametri (agli occhi del programmatore): Input, Output, Input-Output



Recap: gestione della memoria e parametri

- Parametri di input e di output e function call frame (agli occhi del compilatore)
- Famiglie di meccanismi di passaggio dei parametri:
  - Passaggio di parametri **per copia**
  - Passaggio di parametri **per riferimento**

Scope delle variabili

- Formalizzazione: scope e *referencing environment*
- Scope statico:

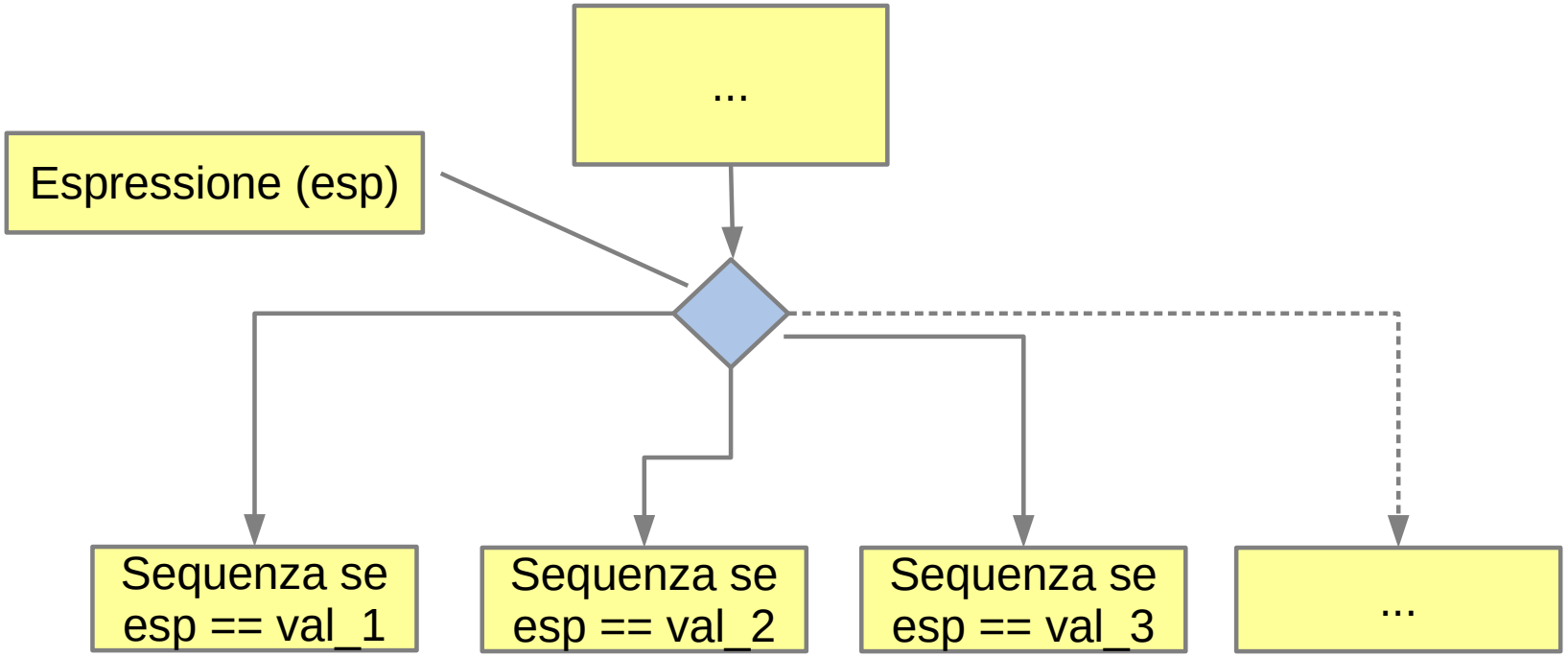
  la dichiarazione a cui fa riferimento ogni identificatore di variabile può essere individuata a compile time

  (codice di esempio costruito insieme)
- Discussione: effetto *shadowing*
- *Scope dinamico*

Sottoprogrammi in GO

- Domanda: se in un sottoprogramma modifico il valore di un parametro di input, la modifica si ripercuote nel chiamante? (discussione)
- Sottoprogrammi GO: formalmente *funzioni*
  - Obiettivo: stesso input, stesso output
  - Domanda: cosa può cambiare il comportamento di una *funzione*? (i.e. stesso input, diverso output in punti diversi del codice)
- Sottoprogrammi che restituiscono più valori
  - assegnamento multiplo
  - dichiarazione e inizializzazione multipla “inline”
  - “comma-ok” idiom e variabili senza nome (blank identifier)
- Esempio: calcola se un numero è primo, e se non lo è restituisci un certificato di “non primalità”

Selezione multi-aria



- Costrutto GO **switch - case**

- Sintassi

  switch <espressione testa> {

  case <espressione 1a>, <espressione 1b>:

  <seq. 1, attivata se
  val. espressione testa == val. espressione 1a, oppure

  val. espressione testa == val. espressione 1b ... >

  case <espressione 2>:

  <seq. 2, attivata se val. espressione testa == val. espressione 2>

  ...

  default:

  <sequenza generica, attivata se nessuna delle precedenti è attivata>

  }

- NB eseguito un qualsiasi “case” si esce dal costrutto
- NB la sezione **default** è opzionale

Esempio

```
switch <variabile> {
case <costante 1>:
    <sequenza 1, attivata se variabile == costante 1>
case <costante 2>:
    <sequenza 2, attivata se variabile == costante 2>
...
default:
    <sequenza generica, attivata se variabile diversa da tutte le costanti>
}
```

Esempio

switch {

case <espressione booleana 1>:

<sequenza 1, attivata se espressione booleana 1 è true>

case <espressione booleana 2>:

<sequenza 1, attivata se espressione booleana 2 è true>

...

default:

<sequenza generica, attivata se nessuna delle espressioni booleane è true >

}

NB se non specificata, l’espressione testa è **true**

Esempio numerico

```
var val int = 8
switch val {
case pow(2,2):
    <sequenza 1, attivata se val == 2² (false)>
case pow(2,3):
    <sequenza 2, attivata se val == 2³ (true)>
}
```

Esempio esoterico

```
switch false {
case 2 == 3:
    // sequenza 1
case 2 == 2:
    // sequenza 2
}
```

- Che effetto ha la porzione di codice qui sopra?

Esempio esoterico

```
switch false {
case 2 == 3:
    // sequenza che viene attivata
case 2 == 2:
    // sequenza non attivata
}
```