Università degli Studi di Padova
Corso di Laurea in Matematica
A.A. 2008-2009

INTRODUZIONE ALLA PROGRAMMAZIONE

Barbara Di Camillo

Si ringraziano il Dott. Enrico Grisan e il Dott. Fabio Aiolli per il materiale didattico fornito

Programmazione

- Un programma descrive al computer, in estremo dettaglio, la sequenza di passi necessari a svolgere un particolare compito
- L'attività di progettare e realizzare un programma è detta programmazione

Cos'è un algoritmo

- Si dice algoritmo la descrizione di un metodo di soluzione di un problema che sia
  - Eseguibile
  - Priva di ambiguità
  - Arrivi ad una conclusione in tempo finito

Un esempio di algoritmo

- Problema: avendo depositato 10000 euro in un conto bancario, che produce il 5% di interessi all'anno, capitalizzati annualmente,quanti anni occorrono affinchè il saldo del conto arrivi al doppio della cifra iniziale?

Algoritmo

1. L'anno attuale è 0; il saldo attuale è 10.000
2. Ripetere i passi 3 e 4 finchè il saldo è minore di 20.000, poi passare al passo 5
3. Aggiungere 1 al valore dell'anno attuale
4. Il nuovo saldo attuale è il valore del saldo attuale moltiplicato per 1.05
5. Il risultato è il valore dell'anno attuale

Un esempio di algoritmo

- Il metodo di soluzione proposto:
  - È non ambiguo, perché fornisce precise istruzioni su cosa bisogna fare ad ogni passo e su quale deve essere il passaggio successivo
  - È eseguibile, perché ciascun passaggio può essere eseguito concretamente
  - Arriva a conclusione in un tempo finito

Altro esempio

Carica due valori dalla RAM, li somma e memorizza il risultato al posto del maggiore dei 2 numeri sommati (nel caso siano uguali, in uno qualsiasi dei 2 indirizzi RAM)

Codice assembler algoritmo 1

```
X: INT 38;
Y: INT 8;
    LOAD R0 X;
    LOAD R1 Y;
    COMP R0 R1;
    BRGE maggiore;
    ADD R0 R1;
    STORE R0 Y;
    STOP;
maggiore:
    ADD R0 R1;
    STORE R0 X;
    STOP;
```

Codice assembler algoritmo 2

```
X: INT 38;
Y: INT 8;
    LOAD R0 X;
    LOAD R1 Y;
    LOAD R2 X;
    ADD R2 R1;
    COMP R0 R1;
    BRGE maggiore;
    STORE R2 Y;
    STOP;
maggiore:
    STORE R2 X;
    STOP;
```

A cosa servono gli algoritmi?

- L'identificazione di un algoritmo è requisito indispensabile per risolvere un problema con il computer
- La scrittura di un programma consiste nel tradurre un algoritmo in qualche linguaggio comprensibile al computer
- Prima di scrivere un programma è necessario conoscere l'algoritmo che risolve il problema!

Linguaggi assembly

- Programmazione con le istruzioni base della CPU (operazioni aritmetiche, trasferimento, confronto, salto, I/O)

```
X: INT 38;
Y: INT 8;
   LOAD R0 X;
   LOAD R1 Y;
   LOAD R2 X;
   ADD R2 R1;
   COMP R0 R1;
   BRGE maggiore;
   STORE R2 Y;
   STOP;
maggiore: STORE R2 X;
   STOP;
```

Linguaggi di alto livello

- Scriviamo un programma in maniera quasi naturale:

```
X: INT 38;
Y: INT 8;
SE(x>y)
        y=x+y
ALTRIMENTI
        x=x+y
```

- Bisogna tradurre questo programma in linguaggio macchina: compilazione

Compilatore

- Il compilatore riceve in ingresso un programma specificato ad alto livello e restituisce lo stesso programma in linguaggio macchina

- La funzione calcolata dai due programmi deve essere la stessa

Compilatore

- Il programma originale (codice sorgente) è un file di testo
- Il compilatore procede attraverso le fasi di front end e di back end per poter restituire lo stesso programma in linguaggio macchina

Stadio di *front end*

- ✓Analisi lessicale (vengono analizzate le istruzioni)
- ✓Analisi sintattica del codice sorgente (vengono individuati i costrutti)
- ✓Analisi semantica (rilevazioni errori semantici e traduzione di espressioni in sequenze elementari)
- ✓Generazione del codice intermedio

Stadio di *front end*

- Analisi lessicale il compilatore divide il codice sorgente in tanti pezzetti chiamati token. I token sono gli elementi minimi (non ulteriormente divisibili) di un linguaggio, ad esempio parole chiave (for, while), nomi di variabili (pippo), operatori (+, -, «).

Stadio di *front end*

- Analisi sintattica L'analisi sintattica prende in ingresso la sequenza di token generata nella fase precedente ed esegue il controllo sintattico, attraverso un grammatica. Il risultato di questa fase è un albero di sintassi.

Stadio di *front end*

- Analisi semantica L'analisi semantica si occupa di controllare il significato delle istruzioni presenti nel codice in ingresso. Controlli tipici di questa fase sono il type checking, ovvero il controllo di tipo, controllare che gli identificatori siano stati dichiarati prima di essere usati e cosi via.

Stadio di *front end*

- Generazione del codice intermedio:
  Dall'albero di sintassi viene generato il codice intermedio.

Stadio di *back end*

Anche lo stadio di back end si divide in più fasi:

- Ottimizzazione del codice intermedio.
- Generazione del codice target: in questa fase viene generato il codice nella forma del linguaggio target. Spesso il linguaggio target è un linguaggio macchina.

Linker

- Le funzioni nel codice oggetto non sono tutte conosciute dal compilatore
- Alcune possono essere riposte in una libreria di funzioni
- Una libreria è un catalogo di funzioni utilizzate dal programma

Linker

- Il linker aggiunge al codice oggetto le funzioni di libreria che vengono utilizzate dal programma
- Aggiunge delle fasi di preambolo e postambolo per caricare ed eseguire correttamente il programma
- Genera un programma eseguibile

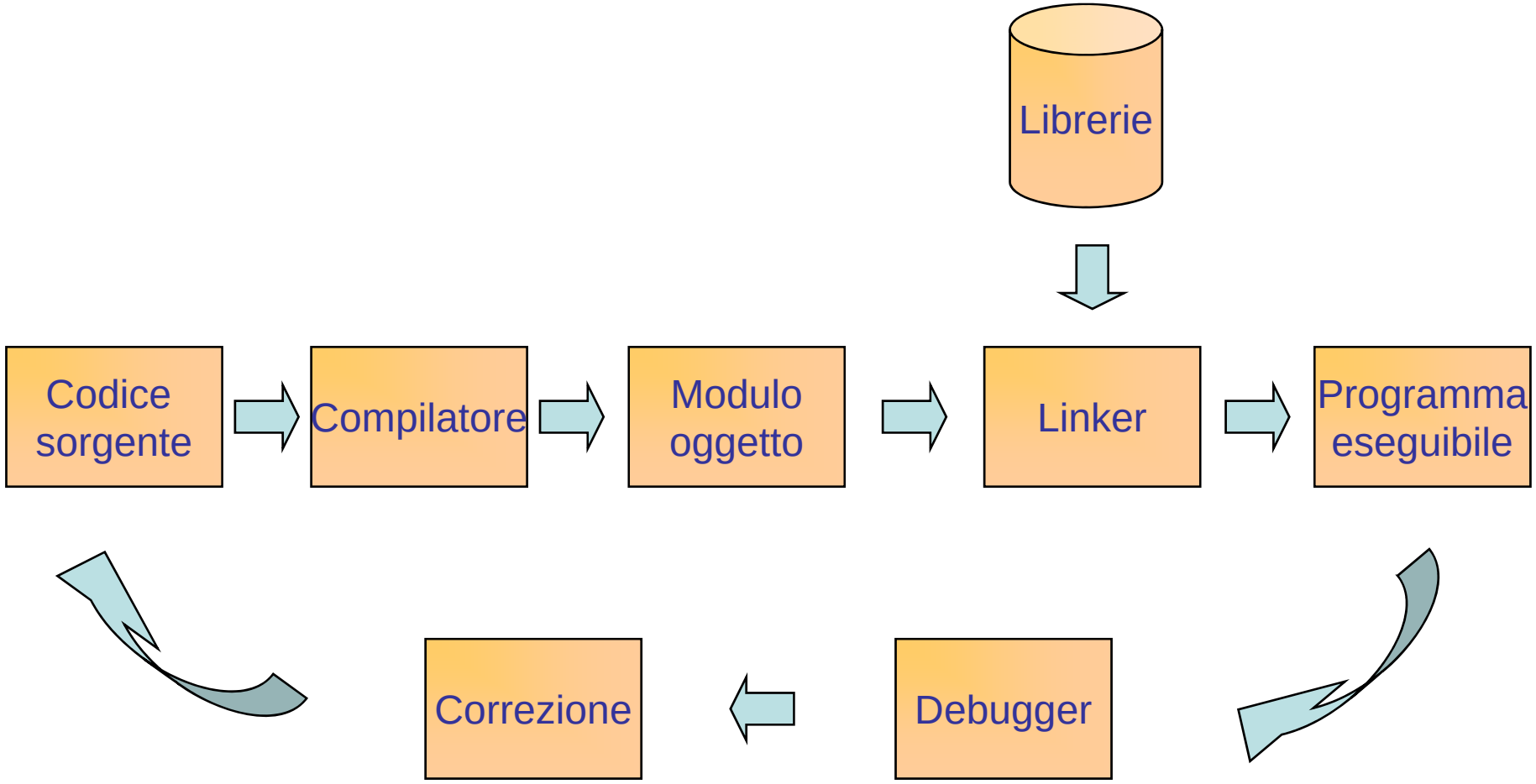
Debugger

- Anche quando tutti gli errori sintattici sono stati corretti, non è detto che il programma funzioni!
  - Errori "runtime" che interrompono il programma o portano a cicli infiniti
  - Errori che portano il programma a fornire risultati errati

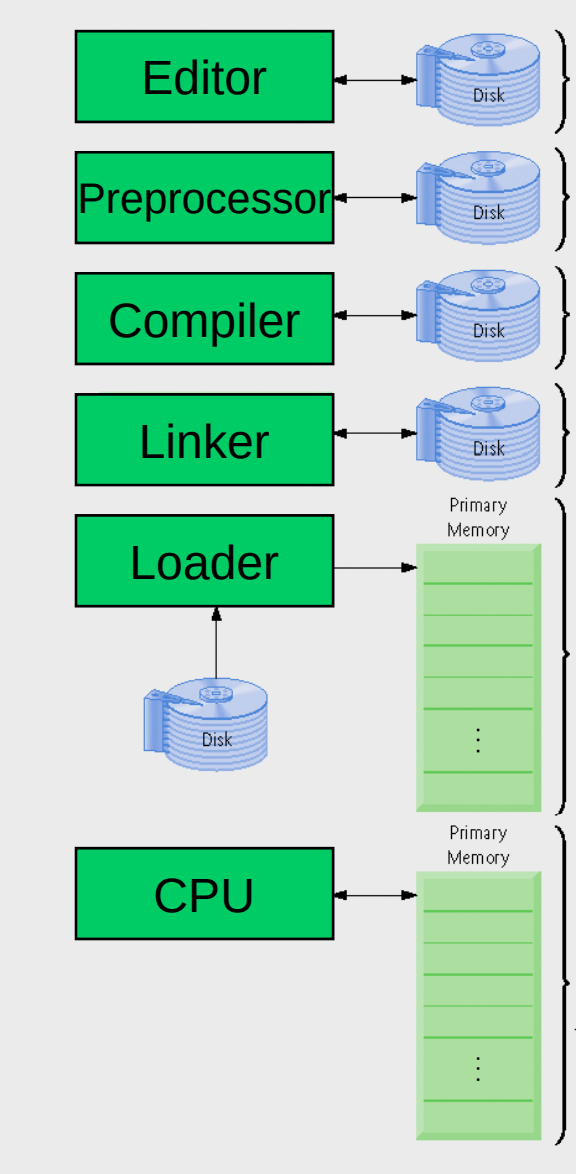
Debugger

- Un errore in un programma è chiamato "bug"
- Il de-bugger serve a eliminare questi errori analizzando:
  - Il valore delle variabili passo a passo
  - Interrompendo l'esecuzione al verificarsi di alcuni eventi

Dal codice all'eseguibile



Dal codice all'eseguibile



Si scrive il programma in un editor e si salva sul disco

Il codice viene pre-elaborato

Il compilatore crea un file oggetto e lo salva sul disco

Il linker collega il modulo oggetto con le librerie, crea un file eseguibile e lo salva su disco

Il loader carica il programma nella memoria principale

La CPU carica una istruzione alla volta e la esegue

Preprocessore o precompilatore

- Un preprocessore o precompilatore è un programma che effettua sostituzioni testuali sul codice sorgente di un programma per espandere in-line dichiarazioni di funzioni di libreria utilizzate e piccole routine.
- Tipicamente, il preprocessore viene lanciato nel processo di compilazione di un software, e il file risultante verrà preso in input da un compilatore.
- Le istruzioni inserite nel codice per essere elaborate dal preprocessore sono solitamente dette "direttive". Per evitare al preprocessore l'onere di comprendere il linguaggio di programmazione ospitante, le direttive devono essere facilmente distinguibili dal resto del codice. Per esempio, nel preprocessore del C, le direttive sono righe che iniziano con il carattere "#".

Preprocessore o precompilatore

- Un altro scopo comune dei preprocessori è permettere l'inserimento di codice di un linguaggio di alto livello, per esempio SQL, all'interno di un programma scritto in un linguaggio di più basso livello, per esempio C. In tal caso, il preprocessore traduce le istruzioni del linguaggio di più alto livello in istruzioni del linguaggio di livello inferiore.
- L'uso massiccio di funzioni di preprocessore può rendere molto caotico il codice, pertanto i linguaggi di programmazione di formulazione recente forniscono caratteristiche che riducono o eliminano la necessità di usare preprocessori.

Linguaggi compilati

- Fino ad ora abbiamo visto il meccanismo di funzionamento dei linguaggi compilati: un compilatore non esegue il programma che riceve in ingresso, ma lo traduce in linguaggio macchina (memorizzando su file il codice oggetto pronto per l'esecuzione diretta da parte del processore).

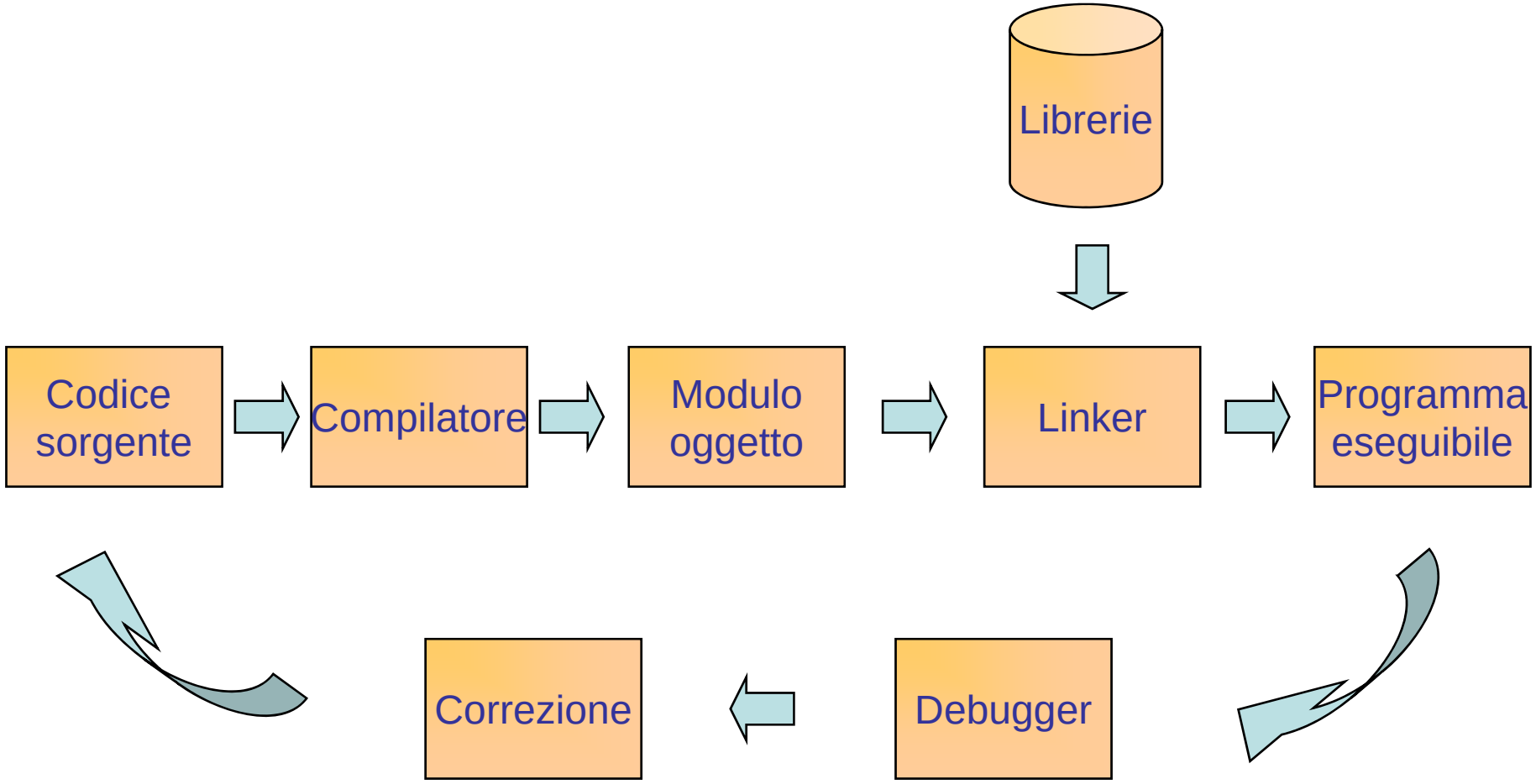
Linguaggi interpretati

- Un linguaggio interpretato è invece un linguaggio di programmazione i cui programmi vengono eseguiti da un interprete.
- Un programma interpretato, in esecuzione, richiede più memoria ed è meno veloce. Durante l'esecuzione, l'interprete deve infatti analizzare le istruzioni a partire dal livello sintattico, identificare le azioni da eseguire, trasformarle in linguaggio macchina ed eseguirle; mentre le istruzioni del codice compilato, già in linguaggio macchina, vengono caricate e istantaneamente eseguite dal processore.

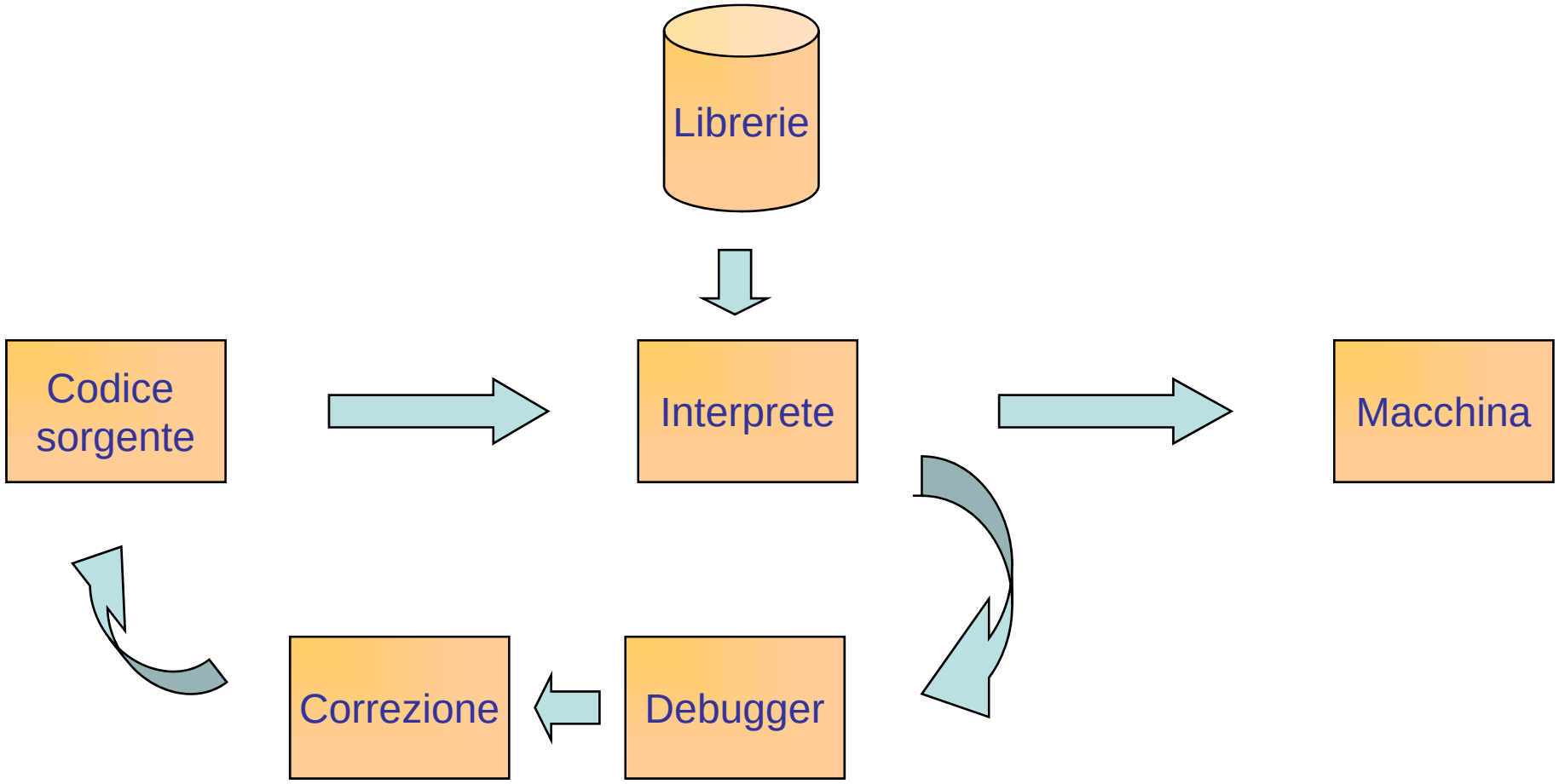
Linguaggi interpretati

- In compenso, l'interpretazione di un programma può essere più rapida del ciclo compilazione/esecuzione. Questa differenza può costituire un vantaggio durante lo sviluppo di un programma.
- Inoltre, la maggior parte degli interpreti consentono all'utente di agire sul programma in esecuzione sospendendolo, ispezionando o modificando i contenuti delle sue variabili, e così via, in modo spesso più flessibile e potente di quanto si possa ottenere, per il codice compilato, da un debugger.

Linguaggi compilati



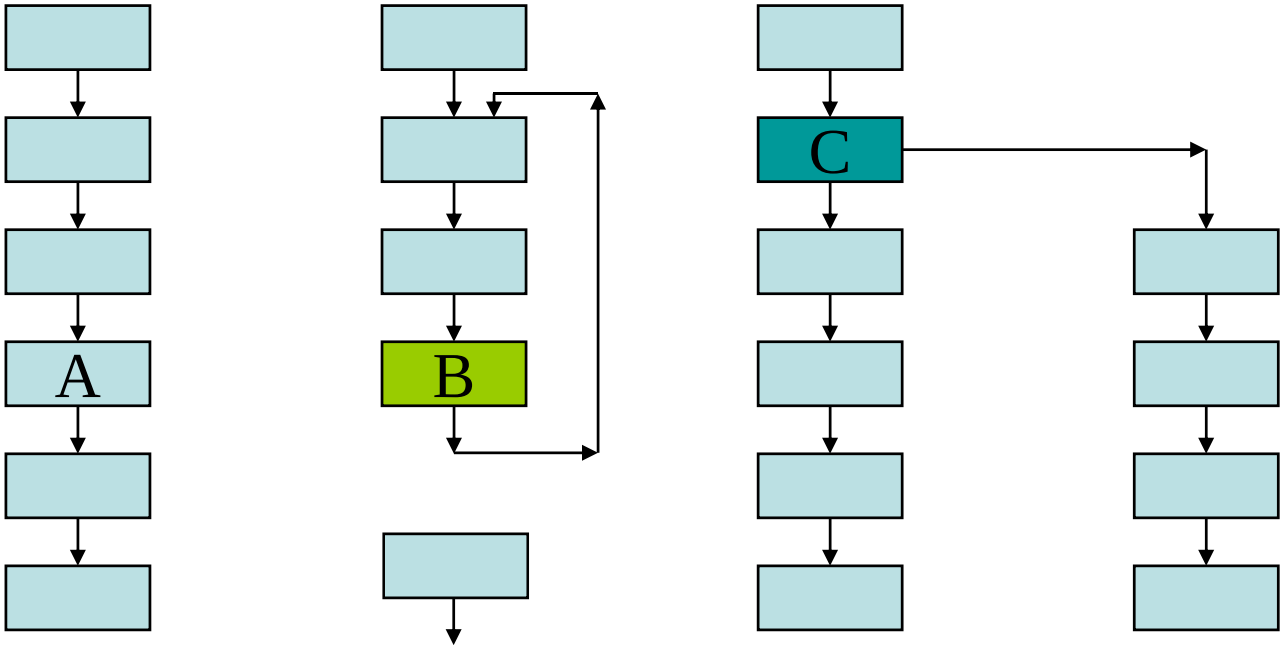
Linguaggi interpretati



Programmi e grafi

- Se associamo ad ogni *istruzione* un *vertice* di un grafo e colleghiamo ogni istruzione alla istruzione che la segue con un *arco orientato* otteniamo il *grafo di flusso* associato al programma
- Le istruzioni di salto condizionato sono caratterizzate da *due archi* in uscita che verranno percorsi in alternativa in funzione del verificarsi o meno della condizione di salto

Programmi e grafi



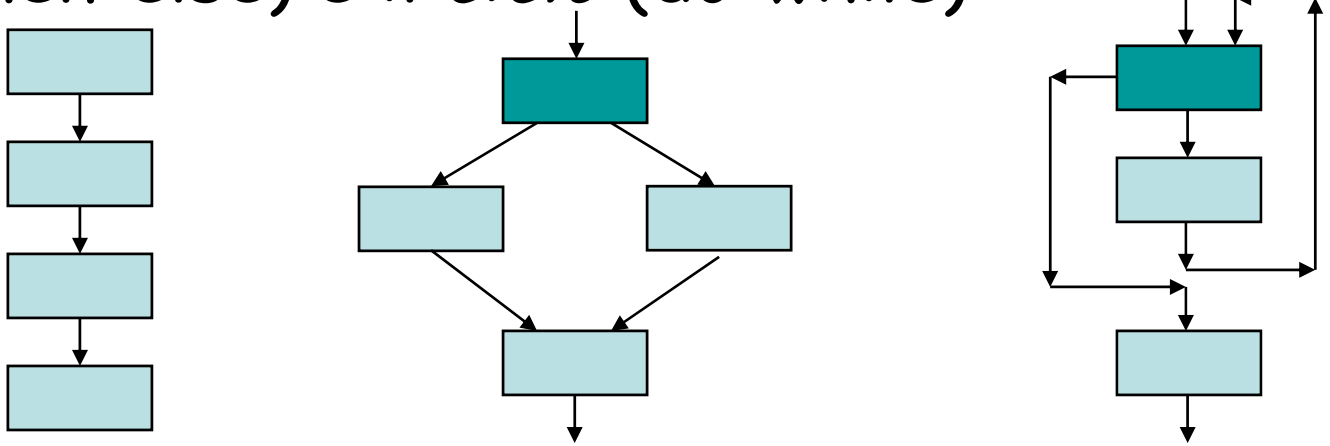
istruzioni in sequenza

salto

salto condizionato

La programmazione strutturata

- Negli anni '60 fu dimostrato che qualsiasi grafo di flusso poteva essere trasformato in un grafo *strutturato* equivalente (dal punto di vista del risultato) composto solo da *sequenze* di istruzioni combinate con le due strutture fondamentali la *scelta* (if-then-else) e il *ciclo* (do-while)



Istruzioni di salto

- Le istruzioni che compongono un programma vengono eseguite in *sequenza*
- Istruzioni di *salto* (JUMP o GOTO) consentono di alterare l'ordine di esecuzione
- Le istruzioni di salto possono essere *condizionate* dal valore di un dato o di un'espressione
- Le istruzioni di salto condizionato consentono di scrivere programmi il cui comportamento dipende dai dati
- Un uso sconsiderato delle istruzioni di salto produce programmi di cui è difficile verificare la correttezza (detti programmi *spaghettata*)

L'istruzione `if-then-else`

Scelta tra due alternative, se una condizione e' vera

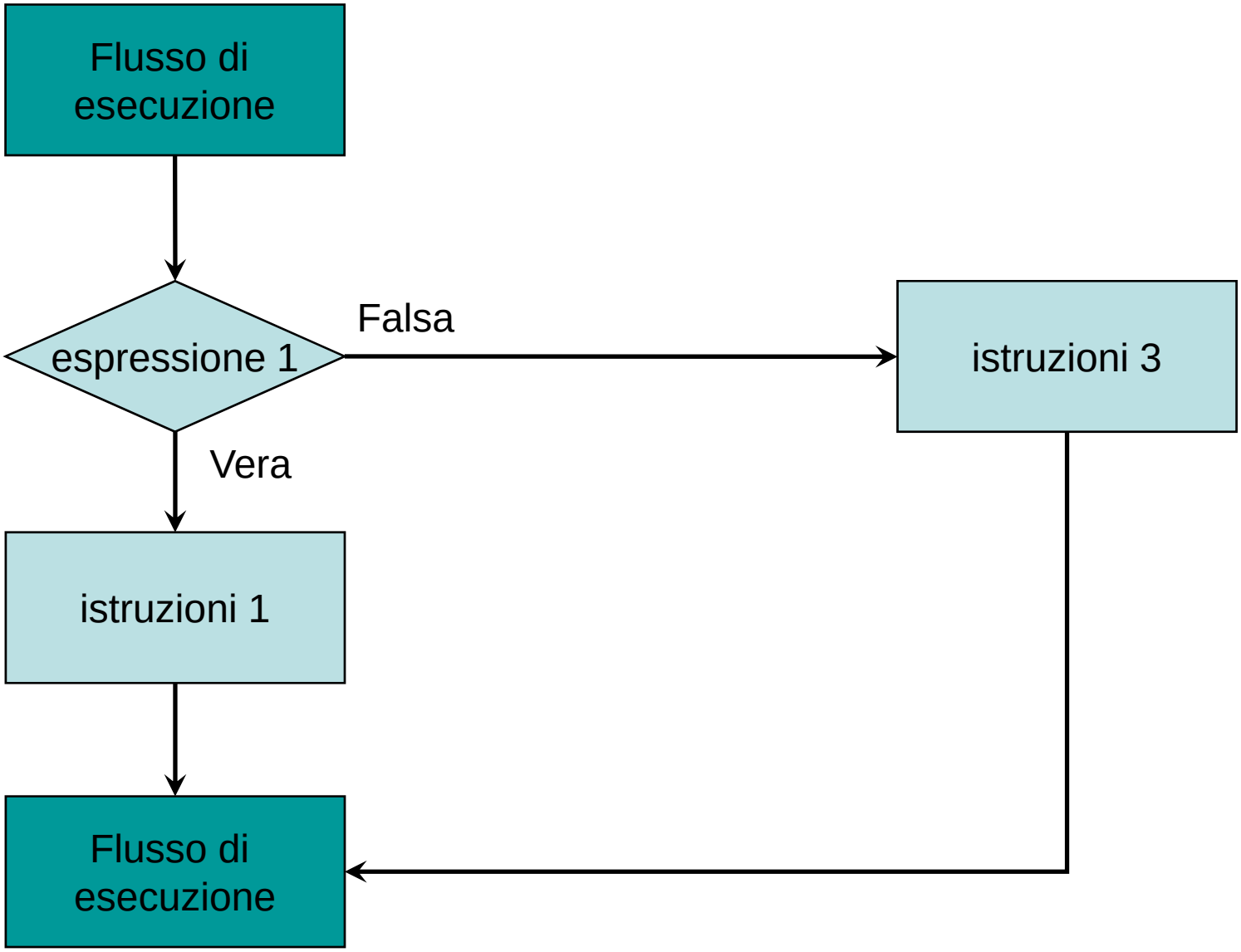
- If (condizione) then (`istruzioni1`) else (`istruzioni2`)
- If (condizione) then (`istruzioni1`)

Es.: if (ci sono biglietti) then (compra un biglietto)

Semantica

- L'esecuzione di `istruzioni1` avviene solo se l'espressione booleana 1 è vera, viene saltata se l'espressione è falsa
- L'esecuzione di `istruzioni2` avviene solo se l'espressione booleana 1 è false, mentre l'espressione booleana 2 è vera
- In tutti gli altri casi viene eseguito il blocco di istruzioni istruzioni3, prima di proseguire il flusso di esecuzione

Flusso di esecuzione



Il ciclo while do

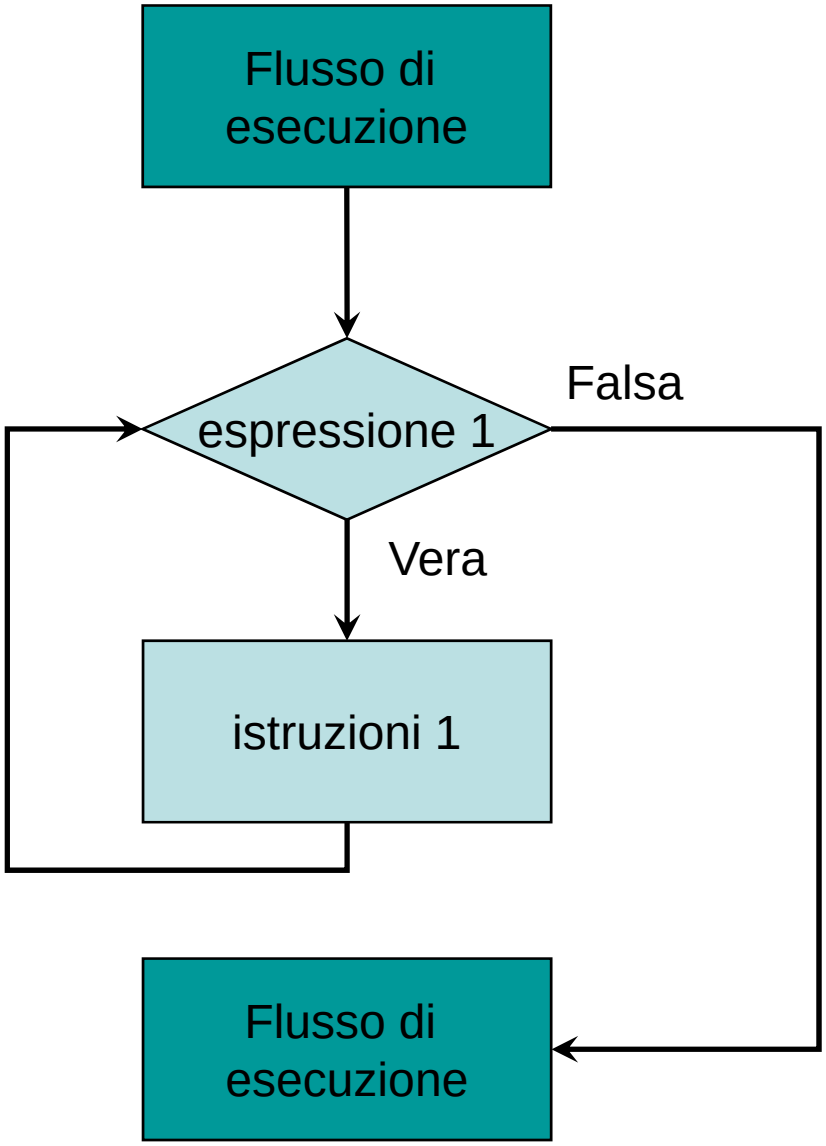
Eseguire un'attivita' purche' una condizione rimanga vera:
While (condizione) do (istruzioni)

Es.: while (ci sono biglietti) do (vendi un biglietto)

Semantica del ciclo `while do`

- All'esecuzione viene valutata l'`espressione booleana`, se è vera vengono eseguite le `istruzioni` che seguono il do. Si valuta poi di nuovo l'`espressione booleana`; se l'espressione è falsa si procede con l'esecuzione del programma
- ATTENZIONE!! Se le `istruzioni` non alterano il valore dell'`espressione booleana` il ciclo non terminerà mai

Flusso di esecuzione: while



Fasi del ciclo relative alla condizione di terminazione

- Inizializzazione: stato iniziale, che verra' modificato dall'azione
- Controllo della condizione di terminazione: confronto tra stato corrente e condizione, terminazione se uguali
- Modifica dello stato: per andare verso la condizione di terminazione

Fasi del ciclo relative alla condizione di terminazione

Esempio di pseudocodice

```
Conta <-3;
While (Conta > 0) do
(stampa il messaggio "Saluti" e
Conta <- Conta - 1)
```

Inizializzazione: Conta <- 3
Condizione di terminazione: Conta <0 o Conta =0
Modifica stato: Conta <- Conta -1

Attenzione alla condizione di terminazione

numero <- 1;

while (numero ≠ 6) do

(numero <- numero +2)

- Condizione di terminazione: numero = 6
- Non verra' mai raggiunta!

Ciclo while

- Il ciclo while e' un ciclo condizionale in cui un blocco di istruzioni viene indefinitamente ripetuto finche' non si verifica una condizione che determina l'uscita dal ciclo
- Prima di entrare nel ciclo while, in generale ci sono delle istruzioni che inizializzano le variabili coinvolte nella condizione (cioe' una espressione booleana) che governa il ciclo. Il ciclo inizia verificando la condizione. Se la condizione si valuta a true allora viene eseguito il corpo del ciclo while e di seguito il controllo ritorna all'inizio del ciclo, ovvero viene nuovamente verificata la condizione, il cui valore potrebbe essere cambiato in virtu' dell'esecuzione del corpo. Questo processo iterativo e' ripetuto fino a quando la condizione si valuta a false: in tal caso, si esce dal ciclo ed il controllo passa all'istruzione che immediatamente segue il ciclo while
- Due considerazioni molto importanti: (1) la condizione del ciclo deve essere inizializzata: se la condizione non e' inizializzata, il ciclo potrebbe usare valori errati o addirittura indefiniti; (2) il valore della condizione deve mutare durante il ciclo: se il valore della condizione non cambia durante il ciclo, non ci sara' alcuna possibilita' di uscita dal ciclo, che continuera' indefinitamente

Esempio

Verificare se un elemento e' presente in un elenco di elementi

- Supponiamo ordine crescente (alfabetico o numerico)
- Scorriamo tutto l'elenco dall'inizio alla fine finche'
  - troviamo l'elemento
  - o non ci sono piu' elementi da guardare
  - o gli elementi rimasti sono maggiori

Esempio

```
Procedure Cerca (lista, valorecercato)
if (elenco vuoto)
then (stampa "no")
else
  (valore<-primo-elemento;
   while (valorecercato>valore e ci sono ancora
          elementi)
   do ( valore <- elemento successivo
        if (valorecercato = valore)
        then (stampa si)
        else (stampa no)
      )
  )
```

Il ciclo for

Eseguire un'attivita' finché un contatore ct, inizializzato al valore val_min non è superiore a val_max:

for (ct=val_min; ct≤val_max; ct<-ct+1) (istruzioni)

Es.:

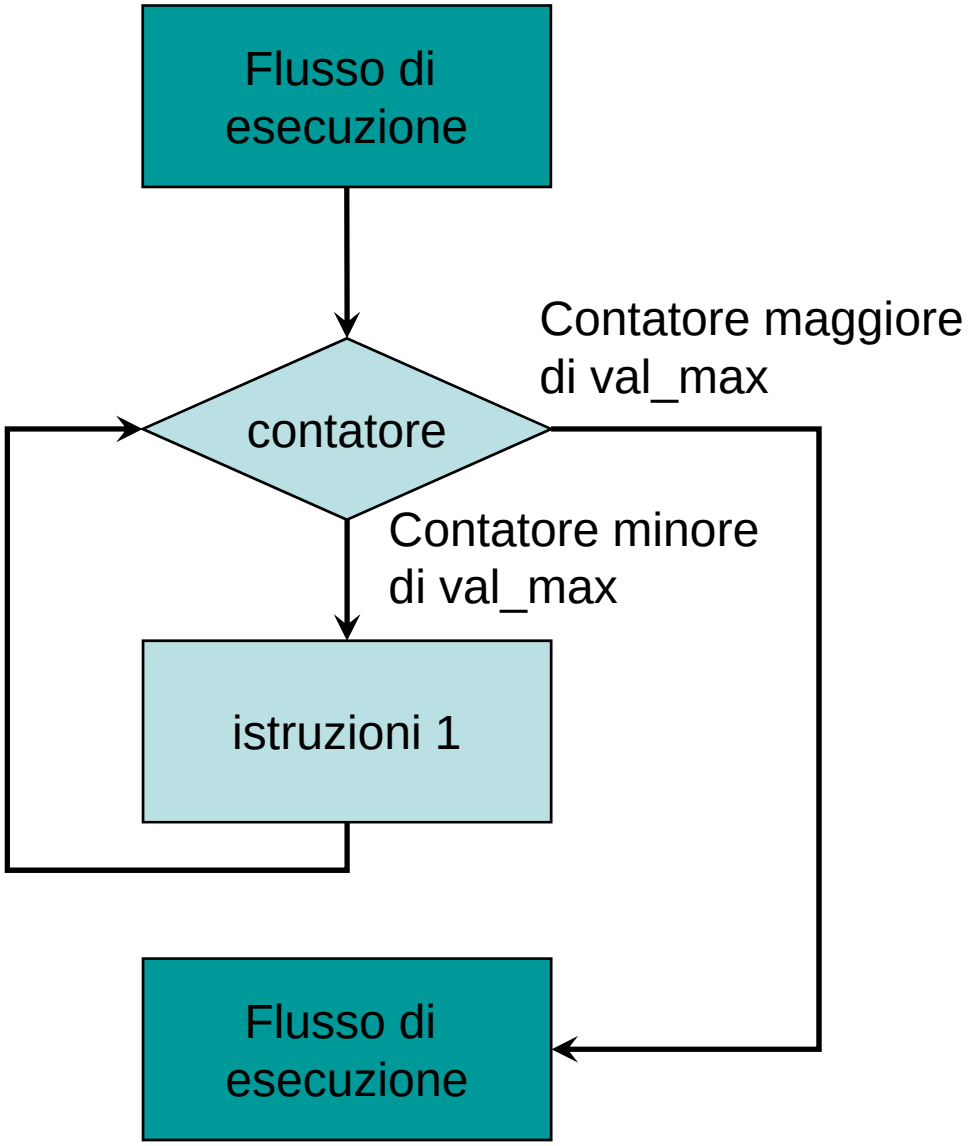
x<-0

for (i=1; i≤5; i<-i+1) (x<-x+i)

Semantica del ciclo `for`

- Data un valore iniziale *`val min`* ed un valore finale *`val_max`*, il ciclo *`for`* prima inizializza la variabile *`ct`* al valore di *`val min`*, quindi incrementa (aumenta il valore di) *`ct`* di una unità fino a che *`ct`* non diventa maggiore di *`val_max`*.
- Ad ogni icremento della variabile contatore ct vengono eseguite le istruzioni comprese fra l'inizio del ciclo *`for`* e la sua fine identificata

Flusso di esecuzione: for



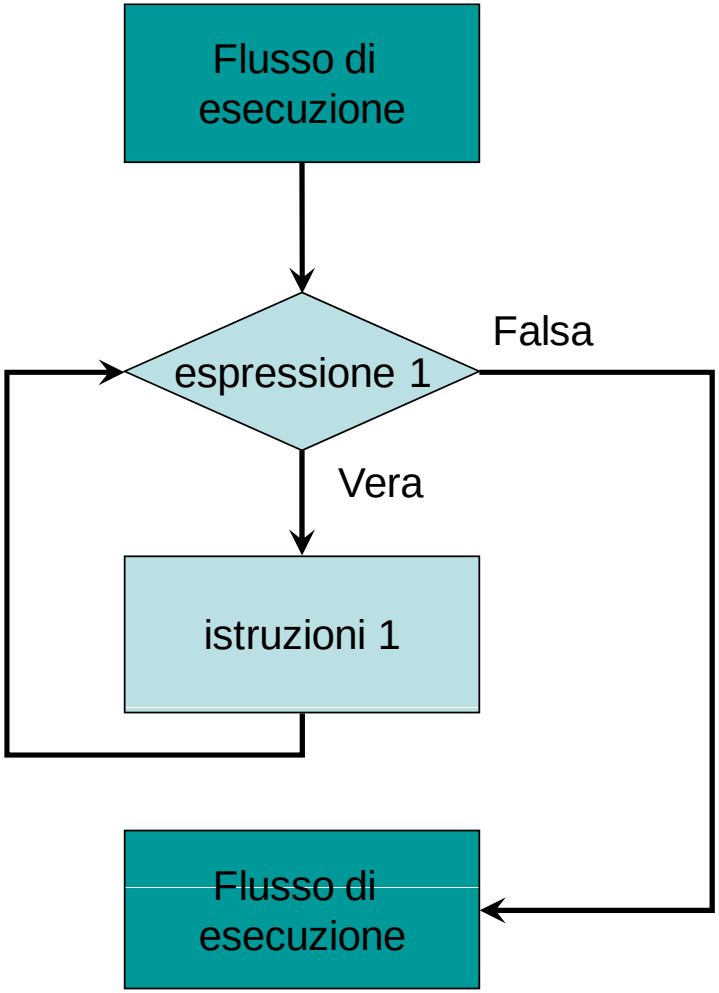
Fasi del ciclo relative alla condizione di terminazione

- Inizializzazione: stato iniziale, che verra' modificato dall'azione
- Controllo della condizione di terminazione: confronto tra stato corrente e condizione, terminazione se uguali
- Modifica dello stato: per andare verso la condizione di terminazione

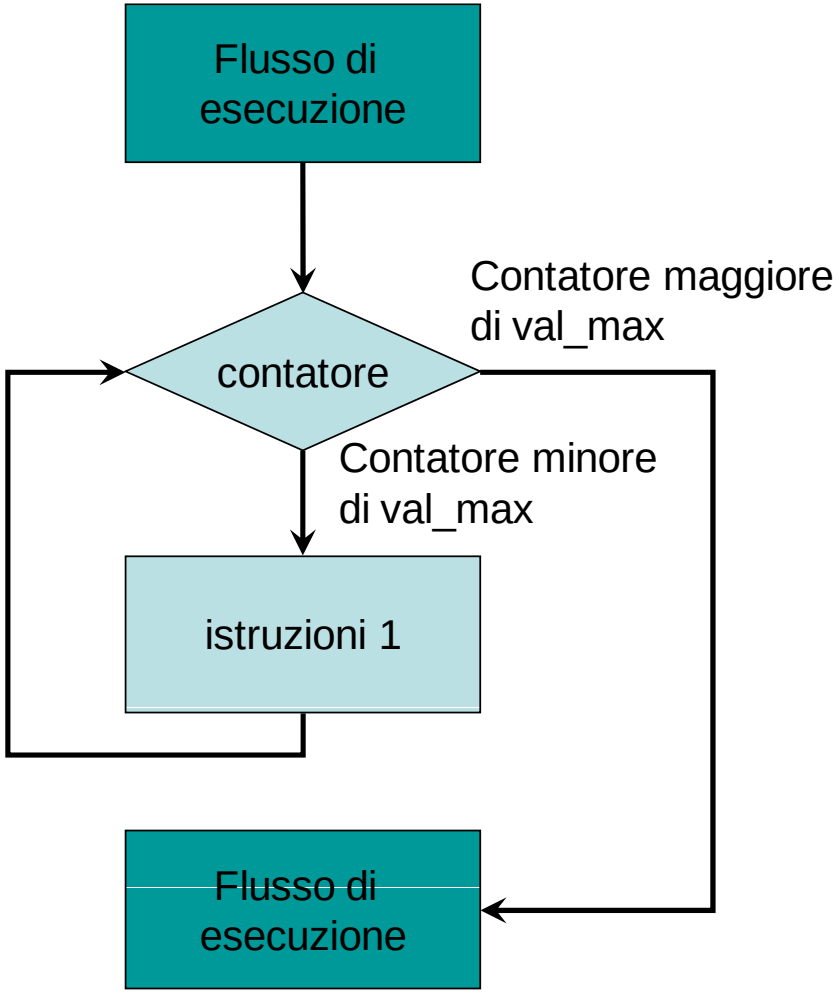
Ciclo for

- La parte *inizializzazione introduce e/o inizializza una variabile di ciclo* (in generale ci potrebbero essere piu` variabili di ciclo) che deve essere **testata nella parte *controllo* e *aggiornata nella parte aggiornamento***.
- La parte *controllo consiste in una espressione booleana che fornisce la* **condizione di terminazione del ciclo**, **basato sul valore corrente (ovvero a quel** punto della ripetizione del ciclo) della variabile di ciclo. Questo confronto viene eseguito ad ogni iterazione del ciclo
- La parte *aggiornamento e' generalmente costituita da assegnazioni che* **modificano il valore della variabile di ciclo**, **ogni volta che il ciclo viene** eseguito
- Il *corpo del ciclo e' costituito da una sequenza di istruzioni, che viene eseguita* dopo il controllo

Ciclo while



Ciclo for



Ciclo while vs ciclo for

La parte ***inizializzazione viene eseguita una sola volta all'inizio del ciclo.***

La sequenza di esecuzione delle tre altre parti del ciclo for e':

(1) controllo,

(2) corpo,

(3) aggiornamento

La sequenza di esecuzione del ciclo while e'invece:

(1) controllo,

(2) corpo e aggiornamento (l'aggiornamento deve essere contenuto nel corpo delle istruzioni)