

# Linguaggi di Programmazione

## CORSO DI Laurea IN "INFORMATICA"

### Strutturare il controllo

Valeria Carofiglio

(Questo materiale è una rivisitazione del materiale prodotto da Nicola Fanizzi)

## Contenuti

Come si realizzano le sequenze di controllo del flusso nei linguaggi ad alto livello:

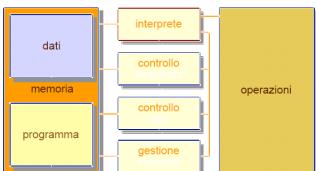
(ovvero)

come l'interprete nella macchina astratta gestisce il flusso di esecuzione delle istruzioni di un programma scritto nel linguaggio comprensibile a quella macchina

- Dalla Sintassi (notazione) alla Semantica per valutare
  - Espressioni
  - Comandi
    - assegnazioni
  - Controllo della sequenza
    - esplicativi, condizionali, iterativi
    - programmazione strutturata
  - Ricorsione
    - ricorsione in coda

## Controllo del flusso di esecuzione...

Funzionalità dell'interprete della macchina astratta



Realizzazione del flusso di esecuzione in un programma

- COME l'interprete nella macchina astratta gestisce il flusso di esecuzione delle istruzioni di un programma scritto nel linguaggio comprensibile da quella macchina

## ..livello di macchina astratta

### Tipologie

#### • Macchina astratta di basso livello

- Gestione semplice: aggiornamento del valore del PC (program counter)

#### • Macchina astratta di alto livello

- Gestione complicata:  
uso di opportuni costrutti per strutturare il controllo (in dipendenza del linguaggio)
  - Controllo della sequenza di valutazioni di Espressioni: Dalla Sintassi (notazione) alla Semantica (valutazione)
  - Comandi: assegnazioni
  - Controllo della sequenza di istruzioni: esplicativi, condizionali, iterativi, programmazione strutturata
  - Ricorsione: ricorsione in coda

## Espressioni

Costituenti di tutti i linguaggi di programmazione

### Definizione:

entità sintattica **da valutare**;  
se la valutazione termina produce un **valore**,  
altrimenti il valore è considerato **indefinito**

Esempio:  $4 + 3 * 2$

- pare ovvio che la valutazione produca il valore 10;
- assunzione implicita: precedenza di **\*** su +  
altrimenti il risultato sarebbe stato diverso

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni

Costituenti di tutti i linguaggi di programmazione

### Composta da

- una singola entità: costante, variabile
  - es.: true, 12.45E4, ipotenusa, contoCorrente, ....
- operatore applicato ad una espressione
  - es.: not(flag) - saldoConto
- operatore applicato a due (o più) espressioni
  - es: a+3/4, (n<0) and not(ordinato)

### GRAMMATICA LIBERA

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Come valutare Espressioni?

Sappiamo che la semantica è guidata dalla sintassi....

Tramite la seguente grammatica ...

$\langle \text{Expr} \rangle ::= \langle \text{Num} \rangle \mid \langle \text{Var} \rangle \mid \langle \text{Expr} \rangle \llcorner \langle \text{Expr} \rangle \mid \langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle$   
 $\langle \text{Num} \rangle ::= \{1\} \dots$   
 $\langle \text{Var} \rangle ::= X_1 \mid X_2 \dots$

Grammatica: Per definire cosa sia una stringa "espressione"

Generiamo la stringa sintatticamente corretta

(con struttura astratta)

$X_1 + 8 - X_2$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle \mid \langle \text{Var} \rangle \mid \langle \text{Expr} \rangle \llcorner \langle \text{Expr} \rangle \mid \langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle$

$\langle \text{Var} \rangle \rightarrow X_1 \mid X_2 \dots$

$\langle \text{Expr} \rangle \llcorner \langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle \mid \langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle \llcorner \langle \text{Expr} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Num} \rangle \mid \langle \text{Var} \rangle \mid \langle \text{Expr} \rangle \llcorner \langle \text{Expr} \rangle \mid \langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle$

$\langle \text{Num} \rangle \rightarrow \{1\} \dots$

Albero di derivazione: Per esprimere la struttura logica che la grammatica assegna alla stringa (conveniente per valutare)

Volutiamo la semantica della struttura astratta  $X_1 + 8 - X_2$  a partire da uno stato in cui associa a  $X_1$  il valore 5 e ad  $X_2$  il valore 3.

Ricordate che la stringa, in realtà è l'albero di derivazione.

1:  $\alpha(X) = n$        $\rightarrow (X_1 + 8 - X_2, \alpha \rightarrow (\_)) \text{ 10}$

1:  $\alpha(X) = n$        $\rightarrow (X_1 + 8 - X_2, \alpha \rightarrow (\_)) \text{ 10}$

1:  $\alpha(X) = n$        $\rightarrow (X_1 + 8 - X_2, \alpha \rightarrow (\_)) \text{ 10}$

1:  $\alpha(X) = n$        $\rightarrow (X_1 + 8 - X_2, \alpha \rightarrow (\_)) \text{ 10}$

1:  $\alpha(X) = n$        $\rightarrow (X_1 + 8 - X_2, \alpha \rightarrow (\_)) \text{ 10}$

1:  $\alpha(X) = n$        $\rightarrow (X_1 + 8 - X_2, \alpha \rightarrow (\_)) \text{ 10}$

1:  $\alpha(X) = n$        $\rightarrow (X_1 + 8 - X_2, \alpha \rightarrow (\_)) \text{ 10}$

Notazione lineare: Uso conveniente delle espressioni nel testo di un programma

Ma come rappresentare la priorità degli operatori?

Dipende dai LdP

N. Fanizzi

## Espressioni: sintassi

notazione postfissa (o polacca inversa)

il simbolo dell'operatore segue le espressioni che rappresentano i suoi operandi

- es:  $a \ b + c \ d + *$  è la scrittura dell'espressione precedente

Nei LdP

Usata dal codice intermedio prodotto da alcuni compilatori

## Espressioni: sintassi

notazione infissa

il simbolo dell'operatore binario è posto fra le espressioni che rappresentano i suoi operandi

es: moltiplicazione per z della somma di x e y:  $(x+y)*z$

• notazione più comune (dalla matematica)

• operatori non binari: notazione simile.

• per evitare ambiguità

- parentesi
- regole di precedenza

• per alcuni LdP la notazione infissa è solo un'abbreviazione (*syntactic sugar*) per la notazione prefissa:

- in C++:  $x * y$  è abbreviazione di:  $x.\text{operator}*(y)$

- in Ada:  $*(x,y)$

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: sintassi

notazione polacca prefissa/inversa

Vantaggi:

- si possono usare per rappresentare facilmente operatori di qualunque arietà (al contrario della n. infissa)
- si possono evitare le parentesi
  - utili solo per motivi di leggibilità, non per l'ordine di valutazione
- la valutazione di una espressione in queste notazioni è più semplice

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: sintassi

notazione polacca prefissa

il simbolo dell'operatore precede le espressioni che rappresentano i suoi operandi

es. somma:  $+(x \ y)$  oppure semplicemente  $+ \ x \ y$

es: appl. funzioni  $f(x \ y)$  oppure semplicemente  $f \ x \ y$

• si applica sempre l'operatore che precede immediatamente gli operandi

• es:  $* \ + \ a \ b \ + \ c \ d$  significa, in notazione infissa:  $(a+b)*(c+d)$

• NB: non servono parentesi né regole di precedenza perché sia nota l'aritetà di ogni funzione/operatore:

- I LdP (la maggior parte) la usano per gli operatori unari e le funzioni definite dall'utente

- LISP: notazione polacca di Cambridge include anche gli operatori tra parentesi:  $(* (+ a b) (+ c d))$

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: semantica...

Influenzata dalla rappresentazione dell'espressione

La rappresentazione di una espressione influenza il modo in cui si determina la sua semantica....

....e quindi il modo in cui l'espressione stessa si valuta

- Notazione infissa: Chiarire la precedenza (uso di parentesi /regola di associatività) tra tutti gli operatori

- es.:  $4+3*5$  deve dare 19 e non 35

- es.:  $x=4$  and  $y=5$  senza parentesi porta ad un errore di tipo in Pascal perché interpretato come  $x=(4$  and  $y)=5$

- es.:  $30-(7-5)$  oppure  $(30-7)-5$ ? (associatività del -: da sinistra a destra. NO APL)

- Non è possibile valutare l'espressione con una sola scansione (es: valutare  $5+3*2$  → si valuta prima a destra, poi si torna indietro )

- Notazione prefissa/postfissa:

- Valutare l'espressione con una sola scansione da sinistra a destra usando una pila

- Algoritmi diversi

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: semantica... ...della notazione prefissa

Scansione da sin. A destr. Con una pila: Algoritmo

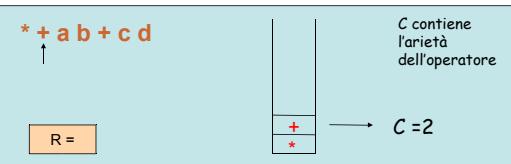
1. Leggi prossimo termine e mettilo sulla pila
2. Se il simbolo è un operatore
  - Inizializza  $C$  alla sua arietà e torna al passo 1.
  - altrimenti (operando) decrementa  $C$
3. Se  $C \neq 0$  torna al passo 1.  
altrimenti
  - a. Applica l'ultimo operatore inserito sulla pila agli operandi successivamente inseriti, memorizzando il risultato in  $R$
  - b. Rimuovi dalla pila operandi e operatore, inserisci  $R$
  - c. Se simboli sulla pila sono terminati allora salta al passo 5.
  - d. Poni  $C = n-m$ , con  $n$  arietà dell'operatore in cima;  $m$  n.ro di operandi presenti sopra
  - e. Torna al passo 4.
3. Se la sequenza di termini da leggere non è vuota torna a 1.

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

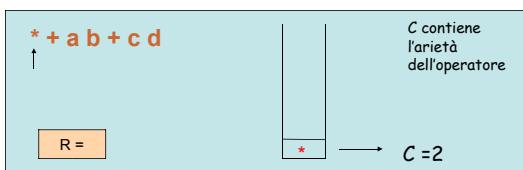
Scansione da sinistra a destra e uso di una pila



## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

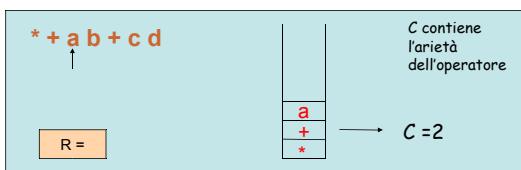
Scansione da sinistra a destra e uso di una pila



## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

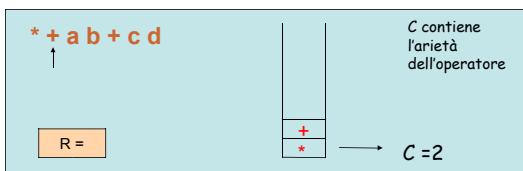
Scansione da sinistra a destra e uso di una pila



## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

Scansione da sinistra a destra e uso di una pila



## Espressioni: semantica... ...della notazione prefissa

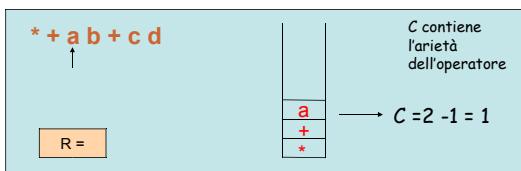
Valutazione semplice:

Scansione da sinistra a destra e uso di una pila

Se il simbolo è un operatore

Inizializza  $C$  alla sua arietà e torna al passo 1.

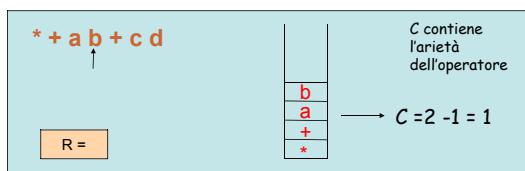
altrimenti (operando) decrementa  $C$



## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

Scansione da sinistra a destra e uso di una pila

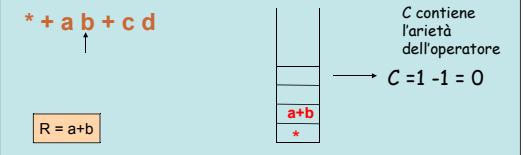


## Espressioni: semantica... ...della notazione prefissa

- Se  $C \neq 0$  torna al passo 1.  
altrimenti

- Applica l'ultimo operatore inserito sulla pila agli operandi successivamente inseriti, memorizzando il risultato in  $R$
- Rimuovi dalla pila operandi e operatore, inserisci  $R$
- Se simboli sulla pila sono terminati allora salta al passo 5.

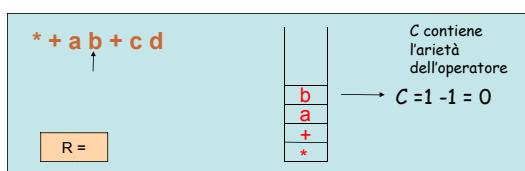
- Poni  $C = n-m$ , con  
n'arietà dell'operatore in cima; m n.ro di operandi presenti sopra



## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

Scansione da sinistra a destra e uso di una pila

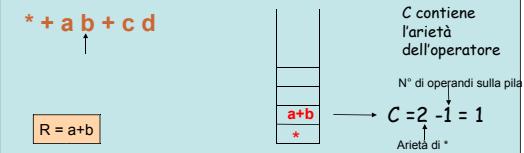


## Espressioni: semantica... ...della notazione prefissa

- Se  $C \neq 0$  torna al passo 1.  
altrimenti

- Applica l'ultimo operatore inserito sulla pila agli operandi successivamente inseriti, memorizzando il risultato in  $R$
- Rimuovi dalla pila operandi e operatore, inserisci  $R$
- Se simboli sulla pila sono terminati allora salta al passo 5.

- Poni  $C = n-m$ , con  
n'arietà dell'operatore in cima; m n.ro di operandi presenti sopra

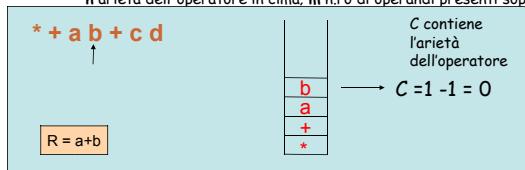


## Espressioni: semantica... ...della notazione prefissa

- Se  $C \neq 0$  torna al passo 1.  
altrimenti

- Applica l'ultimo operatore inserito sulla pila agli operandi successivamente inseriti, memorizzando il risultato in  $R$
- Rimuovi dalla pila operandi e operatore, inserisci  $R$
- Se simboli sulla pila sono terminati allora salta al passo 5.

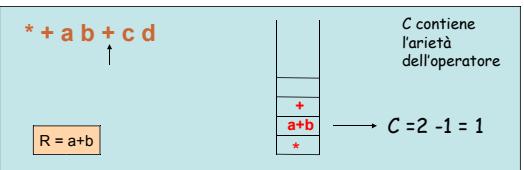
- Poni  $C = n-m$ , con  
n'arietà dell'operatore in cima; m n.ro di operandi presenti sopra



## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

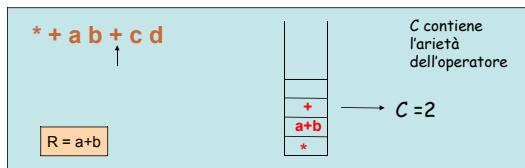
Scansione da sinistra a destra e uso di una pila



## Espressioni: semantica... ...della notazione prefissa

Se il simbolo è un operatore

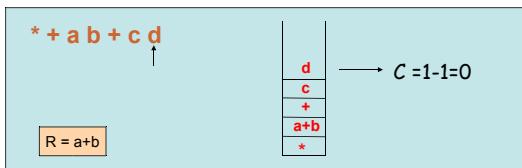
Inizializza  $C$  alla sua arietà e torna al passo 1.  
altrimenti (operando) decrementa  $C$



## Espressioni: semantica... ...della notazione prefissa

Se il simbolo è un operatore

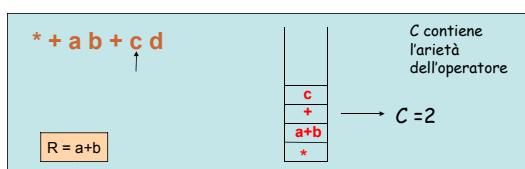
Inizializza  $C$  alla sua arietà e torna al passo 1  
altrimenti (operando) decrementa  $C$



## Espressioni: semantica... ...della notazione prefissa

Se il simbolo è un operatore

Inizializza  $C$  alla sua arietà e torna al passo 1.  
altrimenti (operando) decrementa  $C$

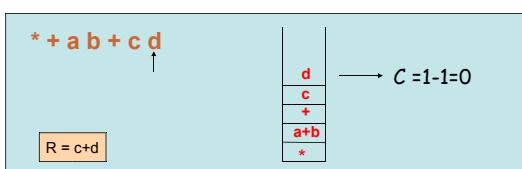


## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

Scansione da sinistra a destra e uso di una pila

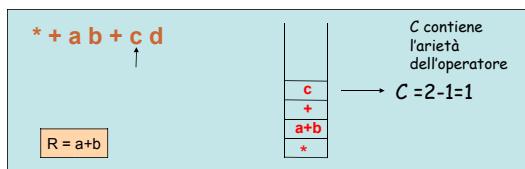
1. Se  $C \neq 0$  torna al passo 1.  
altrimenti
  - a. Applica l'ultimo operatore inserito sulla pila agli operandi successivamente inseriti, memorizzando il risultato in  $R$



## Espressioni: semantica... ...della notazione prefissa

Se il simbolo è un operatore

Inizializza  $C$  alla sua arietà e torna al passo 1.  
altrimenti (operando) decrementa  $C$

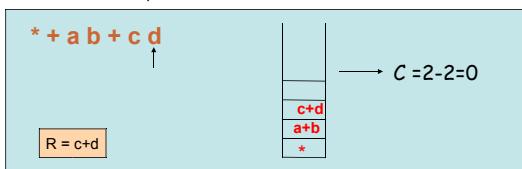


## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

Scansione da sinistra a destra e uso di una pila

- a. Rimuovi dalla pila operandi e operatore, inserisci  $R$
- b. Se simboli sulla pila sono terminati allora salta al passo 5.
- c. Pon  $C$  a  $n-m$ , con  
 $n$  arietà dell'operatore in cima;  $m$  n.ro di operandi presenti sopra
- d. Torna al passo 4.



## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

Scansione da sinistra a destra e uso di una pila

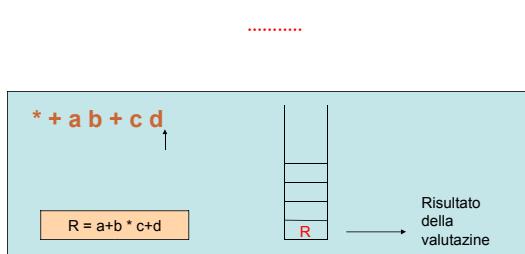
1. Se  $C \neq 0$  torna al passo 1.  
altrimenti
  - a. Applica l'ultimo operatore inserito sulla pila agli operandi successivamente inseriti, memorizzando il risultato in  $R$
  - b. Rimuovi dalla pila operandi e operatore, inserisci  $R$
  - c. Se simboli sulla pila sono terminati allora salta al passo 5.
  - d. Pon  $C = n-m$ , con  
 $n$  arietà dell'operatore in cima;  $m$  n.ro di operandi presenti sopra
  - e. Torna al passo 4.
2. Se la sequenza di termini da leggere non è vuota torna a 1.

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

Scansione da sinistra a destra e uso di una pila



## Espressioni: semantica... ...della notazione prefissa

Valutazione semplice:

Scansione da sinistra a destra e uso di una pila

MA:

- 1) Necessario conoscere l'arietà di un operatore (distinzione sintattica per operatori unari/binari)
- 2) Necessario controllare che ci siano abbastanza operandi sulla pila per effettuare l'operazione
 

*Se  $C \neq 0$  torna al passo 1.*  
altrimenti

  - a. Applica l'ultimo operatore inserito sulla pila agli operandi successivamente inseriti, memorizzando il risultato in  $R$
  - b. Rimuovi dalla pila operandi e operatore, inserisci  $R$
  - c. Se simboli sulla pila sono terminati allora salta al passo 5.
  - d. Pon  $C = n-m$ , con  
 $n$  arietà dell'operatore in cima;  $m$  n.ro di operandi presenti sopra
  - e. Torna al passo 4.

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: semantica... ...della notazione postfissa

Scansione da sinistra a destra e uso di una pila

• Più semplice:

- non serve controllare la presenza degli operandi per l'operatore inserito dato che vengono letti prima

• MA

- Serve conoscere l'arietà'

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: semantica... ...della notazione postfissa

Scansione da sinistra a destra e uso di una pila: Algoritmo

1. Leggi prossimo termine e mettilo sulla pila
2. Se il simbolo è un operatore
  1. Applicallo agli operandi immediatamente precedenti sulla pila, memorizza il risultato in  $R$
  2. Elimina operandi ed operatore dalla pila
  3. Memorizza  $R$  sulla pila
3. Se la sequenza ancora da leggere non è vuota torna a 1.
4. Se il simbolo letto è un operando torna a 1

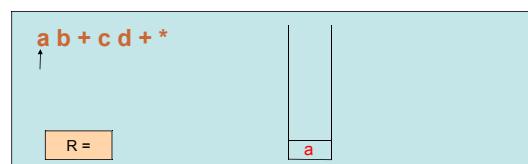
N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: semantica... ...della notazione postfissa

Scansione da sinistra a destra e uso di una pila: Algoritmo

Se il simbolo è un operatore

1. Applicallo agli operandi immediatamente precedenti sulla pila

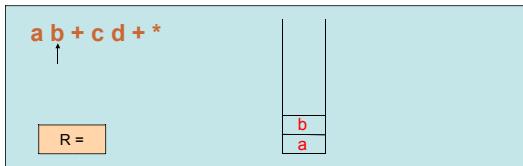


N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: semantica...

...della notazione postfissa

Scansione da sinistra a destra e uso di una pila: Algoritmo



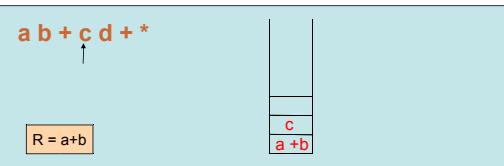
## Espressioni: semantica...

...della notazione postfissa

Scansione da sinistra a destra e uso di una pila: Algoritmo

Se il simbolo è un operatore

1. Applicallo agli operandi immediatamente precedenti sulla pila, memorizza il risultato in R



## Espressioni: semantica...

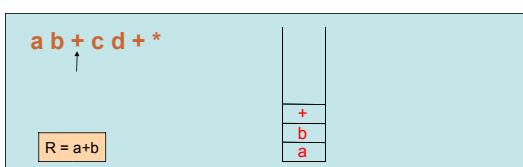
...della notazione postfissa

Scansione da sinistra a destra e uso di una pila: Algoritmo

1. Leggi prossimo termine e mettilo sulla pila

2. Se il simbolo è un operatore

1. Applicallo agli operandi immediatamente precedenti sulla pila, memorizza il risultato in R



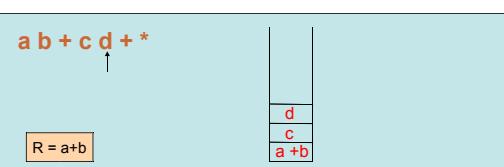
## Espressioni: semantica...

...della notazione postfissa

Scansione da sinistra a destra e uso di una pila: Algoritmo

Se il simbolo è un operatore

1. Applicallo agli operandi immediatamente precedenti sulla pila, memorizza il risultato in R



## Espressioni: semantica...

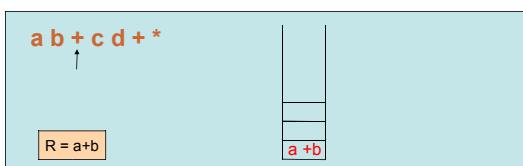
...della notazione postfissa

Scansione da sinistra a destra e uso di una pila: Algoritmo

1. Elimina operandi ed operatore dalla pila

2. Memorizza R sulla pila

1. Se la sequenza ancora da leggere non è vuota torna a 1.



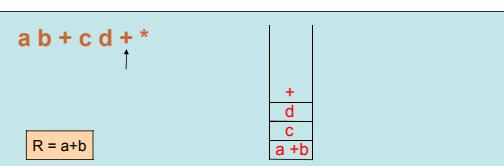
## Espressioni: semantica...

...della notazione postfissa

Scansione da sinistra a destra e uso di una pila: Algoritmo

Se il simbolo è un operatore

1. Applicallo agli operandi immediatamente precedenti sulla pila, memorizza il risultato in R



## Espressioni: semantica...

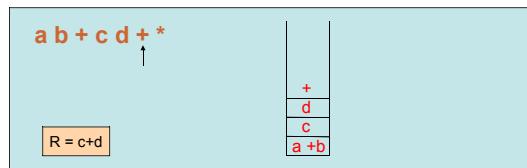
...della notazione postfissa

Scansione da sinistra a destra e uso di una pila: Algoritmo

1. Leggi prossimo termine e mettilo sulla pila

2. Se il simbolo è un operatore

1. Applico agli operandi immediatamente precedenti sulla pila, memorizza il risultato in R



## Espressioni: semantica...

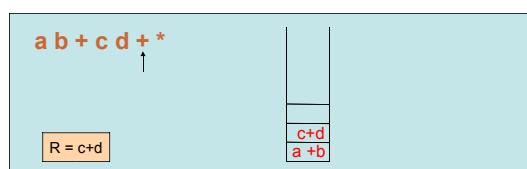
...della notazione postfissa

Scansione da sinistra a destra e uso di una pila: Algoritmo

1. Elimina operandi ed operatore dalla pila

2. Memorizza R sulla pila

1. Se la sequenza ancora da leggere non è vuota torna a 1. ....ecc.



## Espressioni: valutazione...

...ordine di valutazione delle sotto-espressioni

Cosa valutare prima?

nessuna indicazione su:

- 1) quale ordine seguire per valutare nodi dello stesso livello  
( $a + f(b)$ ) oppure ( $c + f(b)$ )

- 2) Valutare prima operando o operatore  
Ese:  $a ++ \rightarrow a++ \leftrightarrow ++a$   
espressioni booleane

Trascrivibile in matematica / Importante in informatica

Perché?

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: valutazione...

...ordine di valutazione delle sotto-espressioni

### Effetti collaterali:

azione che influenza i risultati parziali o finali senza restituire esplicitamente un risultato

Se la valutazione di f influenzasse il valore del suo operando (effetto collaterale)

L'ordine di esecuzione (tra A e B) sarebbe importante



STRATEGIE nei LdP  
-assenti nei ling. Dichiаративи puri;  
ove presenti la def del linguaggio  
specifico l'ordine  
- es. Java impone l'ordine  
"sinistra → destra"  
- Es C non impone ordini

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: valutazione

come calcolare il valore di una espressione

- Notazioni lineari (infissa, prefissa, postfissa):
- Risultato dell'analisi sintattica (nella compilazione)  
→ **alberi sintattici** (struttura logica della stringa secondo la grammatica per la notazione lineare)
  - Nodi interni: operatori
  - Nodi foglia: operandi semplici
  - Sotto-alberi radicati in N: operandi per espressioni di livello inferiore
- Alberi sintattici (nelle fasi succ. della compilazione) → per generare codice oggetto (diverso a seconda del tipo di macchina per la quale il compilatore è costruito) che "valuta" (operazioni che la macchina astratta esegue per computare) l'espressione a run time.

La rappresentazione ad albero chiarisce precedenza e associatività

MA...

N. Fanizzi ° Llinguaggi di Programmaazione (c) 2006

## Espressioni: valutazione...

...ordine di valutazione delle sotto-espressioni

### Aritmetica finita

L'insieme dei numeri rappresentabili in un calcolatore è finito

Riordinando i termini delle espressioni si possono causare

- Errori di overflow
  - Es: a massimo intero rappresentabile,  $b > c$
  - $(a-b+c)$  ok,  $(a+c-b)$  overflow
- Errori dovuti alla precisione: risultati differenti per operandi troncati in modo diverso (es. op. In virgola mobile)

N. Fanizzi ° Llinguaggi di Programmaazione (c) 2006

## Espressioni: valutazione...

...ordine di valutazione delle sotto-espressioni  
Operandi non definiti

Espressioni definite anche se alcuni operandi che vi compaiono non lo sono

(Non si sa a priori quale sia l'operando a cui applicare un operatore )

ad esempio i condizionali. :

$a==0 ? b : b/a$   
( $b/a$  se  $a$  è diverso da 0, altrimenti  $b$ . in C)

### Strategie nei LdP:

Valutazione eager

Valutazione Lazy

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: valutazione...

...ordine di valutazione delle sotto-espressioni

### Valutazione Lazy (corto-circuitata):

non usata da tutti I tipi di linguaggi (attenzione!!!)

Esempio: ERRORE in Pascal se volessimo scorrere la lista fino al termine oppure sino a trovare il valore 3

```
P:= lista;
While (p<>nil) and p^.valore <> 3 do
  P:=P^.prossimo;
```

Pascal non usa la valutazione lazy corto-circuitata--> entrambi gli elementi sarebbero valutati anche se p=nil. Si avrebbe errore.

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: valutazione...

...ordine di valutazione delle sotto-espressioni

### Valutazione eager:

prima valutare gli operandi poi applicare l'operatore ai valori ottenuti

### esempi

$a == 0 ? b : b/a$

(problema:  $b/a$  valutata anche se  $a = 0$ ) → inutile

$a==0 || b/a>2$

(problema: valutati entrambi gli operandi anche se il valore di verità dell'espressione dipende dal valore del primo operando) → inutile

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: valutazione...

...ordine di valutazione delle sotto-espressioni

### Ottimizzazione del codice oggetto:

L'ordine di valutazione potrebbe influenzare l'efficienza dell'esecuzione del codice

a = vettore[i];  
b = a\*a + c\*d;

a\*a più accessi in memoria

- nella seconda assegnazione meglio valutare prima  $c*d$  perché il valore per a (da memoria) potrebbe essere ancora non disponibile
- Messa in atto dai compilatori purché la semantica sia preservata (senza specifica a livello di semantica del linguaggio)

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: valutazione...

...ordine di valutazione delle sotto-espressioni

### Valutazione Lazy (corto-circuitata):

Per decidere quali operandi:

Non valutare gli operandi prima dell'operatore, ma al momento della valutazione dell'operatore stesso

Costosa e Poco usata dai LdP imperativi  
(solo nelle espressioni condizionali/booleane)

### Valutazione corto-circuitata delle espressioni booleane

$a==0 || (b/a)>2$

(primo operando di una disgiunzione/congiunzione ha valore vero/falso, il secondo non viene valutato (dato che il globale ha risultato vero/falso)

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Espressioni: valutazione...

...ordine di valutazione delle sotto-espressioni

Spesso: la def precisa del metodo di valutazione delle espressioni non è specificato nella descrizione semantica del linguaggio

### Una osservazione finale

- **Pragmatica:** particolari sulla semantica del linguaggio sono spesso:

- 1) nascosti,
- 2) poco chiari,
- 3) dipendenti dall'implementazione (compilatore)

usare tutti i mezzi - (" ") op. Booleani specifici, variabili-

per eliminare le fonti di ambiguità

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006





## Assegnamento (cont.)

ancora: sintassi, semantica e valutazione (ling.imperativi)

### In alcuni linguaggi (C e derivati)

- Assegnamenti abbreviati, ottimizzati dal compilatore
  - Incremento/decremento:
    - pre  $++X$ ,  $--X \leftrightarrow X = X+1$
    - Incrementa/decremente  $X$  e restituisce il nuovo r-valore
  - post  $X++, X--$ 
    - Restituisce l'r-valore di  $X$  e Incrementa/Decrementa  $X$
  - Auto <var> <op= <expr> ( $x+=y$ )
    - Incrementa/decremente  $X$  della quantità data dall'r-valore di  $Y$  e restituisce il nuovo r-valore di  $X$

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Controllo esplicito

### Comando sequenziale (quasi sempre indicato con ;)

- Specifica la sequenza tra due comandi più semplici  
 $C1; C2$ 
  - L'esecuzione di  $C2$  comincia dopo la terminazione di  $C1$
- Operatore associativo a sinistra

### Comando composto

- Ragggruppamento di una sequenza usando delimitatori
  - begin...end
  - oppure
  - {...}

usabile in luogo di un comando semplice. Possibili annidamenti

N. Fanizzi ° Llinguaggi di Programmaazione (c) 2006

## Assegnamento (cont.)

ancora: sintassi, semantica e valutazione (ling. Ad Oggetti)

### Linguaggi con modello a riferimenti (Paradigma ad oggetti)

- <var> = <expr>
  - L'espressione <expr> deve essere valutata producendo un riferimento da assegnare alla variabile riferimento:
    - Es.  $x:=expr$ 
      - »  $x$  è un riferimento per l'oggetto ottenuto dalla valutazione di  $expr$
      - » Diverso da copiare un valore in una locazione di memoria
    - Es.  $x:=y$ 
      - »  $x$  e  $y$  sono due riferimenti allo stesso oggetto
  - Utilizzato in Java per gli oggetti
    - Mentre per i tipi primitivi le variabili sono modificabili

N. Fanizzi ° Llinguaggi di Programmaazione (c) 2006

## Controllo esplicito (cont.)

### Comando di salto (con goto)

- Deriva dal salto incondizionato dell'Assembly
  - Esempio: goto A
    - l'esecuzione del salto trasferisce il controllo al punto etichettato da A (potrebbe anche essere un numero di riga)
- Molto avversato. Dijkstra "Go to statement considered harmful", Communications of the ACM, 11(3), 1968
  - Contro il goto: Cosa succede al RdA di un blocco se c'e' un salto all'interno del blocco?
  - Contro il goto: I salti a grandi distanze diminuiscono la leggibilità quindi il riuso e la manutenibilità
- Non essenziale, grazie al teorema di Böhm e Jacopini: "ogni programma può essere tradotto in uno equivalente senza il GOTO"
  - Bandito dai nuovi LdP (java è stato il primo) che hanno costruiti esplicativi per uscita da cicli, ritorno da sottoprogrammi...

N. Fanizzi ° Llinguaggi di Programmaazione (c) 2006

## Controllo della sequenza

Comandi che permettono di specificare l'ordine con cui effettuare le modifiche di stato espresse dagli assegnamenti

- Comandi esplicativi:
  - Composizione (blocco)
  - Salto (goto)
- Comandi condizionali (o di selezione)
  - Permettono di specificare alternative con cui continuare la computazione in base al verificarsi di certe condizioni
- Comandi iterativi
  - Ripetizione di un comando un numero di volte predefinito o dipendente dal verificarsi di certe condizioni

N. Fanizzi ° Llinguaggi di Programmaazione (c) 2006

## Controllo esplicito (cont.)

### Salti controllati a distanze brevi presenti in molti linguaggi

- return
  - ritorno da un sottoprogramma, funzione, metodo
- break
  - uscita da un ciclo o selezione multipla
- continue
  - terminare l'iterazione corrente riprendendo dalla prossima

### Eccezioni

- Salto ad un flusso di controllo parallelo

N. Fanizzi ° Llinguaggi di Programmaazione (c) 2006

## Comandi condizionali

Comandi che esprimono alternative nella computazione

- Comando if (introdotto da ALGOL)
  - if <B-exp> then <cmd> [else <cmd>]

- La presenza di if annidati porta ad ambiguità
  - Generalmente else appartiene all'if più interno (la grammatica lo stabilisce)
  - Oppure si usa un terminatore (endif)
  - Alcuni linguaggi utilizzano esplicitamente l'annidamento con la parola chiave elseif
- Facile implementazione: Attraverso i salti condizionati dell'Assembly

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Comandi condizionali (cont.)

• Comando di selezione multipla: es. case (cont.)

```

    - case <expr> of
      <etichetta_1> : <cmd>
      .....
      <etichetta_N> : <cmd>
      [else <etichetta_N+1>]
    endcase
  
```

- Realizzato in Assembly attraverso una jump table
  - Vantaggio: più efficiente dell'implementazione degli if annidati
  - Svantaggio: la jump table può consumare spazio se i valori sono distanti

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Comandi condizionali (cont.)

- Comando di selezione multipla: es. case

- case <expr> of
  - <etichetta\_1> : <cmd>
  - .....
  - <etichetta\_N> : <cmd>
  - [else <etichetta\_N+1>]
- endcase
- <expr> di tipo compatibile con le etichette
- etichette rappresentate da una o più costanti (o intervalli) -tipi ammessi: scalari
- Più leggibile degli if annidati
- Implementazione: Uso della jump table (assembly)

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Comandi condizionali (cont.)

• Comando di selezione multipla: es. switch (cont.)

```

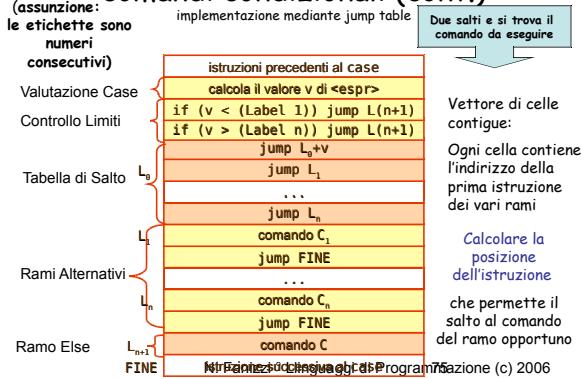
    - switch (<expr>) corpo
      Switch (Expr) {
        case Label1: C1 break;
        case Label2: C2 break;
        .....
        case Labeln: Cn break;
        default: Cn+1 break;
      }
  
```

Scelte multiple molto diverse nei vari LdP

- C, C++, Java: una volta selezionato il ramo, il controllo passa al ramo successivo
- Break: per simulare il case;
- Non ammessi liste o range nelle etichette. (ma simulabili per effetto del case senza break)

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## comandi condizionali (cont.)



## comandi iterativi:

Ripetizione di un comando:

Lunghezza computazione non determinata staticamente

Maggiore espressività  
turing-completezza

- linguaggi a basso livello

- Salti: jump, goto

- linguaggi ad alto livello

- iterazione strutturata. Nella sintassi si distinguono:
  - iterazione determinata (fissato il numero di volte)
  - iterazione indeterminata
- ricorsione
  - iterazione隐式

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## iterazione indeterminata

O Iterazione logicamente controllata è formata da 2 parti  
condizione del ciclo (o guardia): espressione  
corpo: comando

- **sintassi** (Algol e derivati)  
`while (<Espr>) do <cmd>`
  - 1. viene valutata l'espressione (booleana) `<Espr>`
  - 2. se vera si esegue il comando `<cmd>` e si torna al passo precedente
- Implementata sulla macchina fisica con un salto condizionato (facilmente)
- Potenza del costrutto elevata:
  - sequenza + selezione + iterazione indeterminata → Turing completezza del linguaggio (costrutti sufficienti per calcolare)

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006



## Iterazione determinata (cont.)

### semantica

`for <var> = <expr1> to <expr2> by <expr3>  
do <cmd>`

Pascal  
For i=1 to 6 by 2  
do {corpo}

#### Algoritmo

1. valutare `<expr1>`, `<expr2>`, `<expr3>` e ricordare i valori, rispettivamente, `val1` e `val2` e `val3`
2. inizializzare la <var> a `val1`
3. se il valore della `<var>` è maggiore di `val2` termina
  - eseguire il comando `<cmd>` del corpo
  - incrementare il valore di `<var>` del valore di `val3`
  - tornare al passo 3.

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Iterazione indeterminata (cont.)

- In molti linguaggi esiste anche l'istruzione che esegue il controllo dopo aver eseguito il comando
  - Pascal:
    - `repeat <cmd> until <Espr>`
    - corrisponde a:  
`<cmd>;  
while not <Espr> do <cmd>`
  - C, C++, Java:
    - `do <cmd> while <Espr>`
    - corrisponde a:  
`<cmd>;  
while (<Espr>) do <cmd>`

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Iterazione determinata (cont.)

### semantica

#### Osservazioni

- non è possibile ottenere cicli infiniti
- Numero iterazioni (contatore di iterazione- ic): determinato a priori:  $(val2 - val1 + val3)/val3$
- se il passo (`val3`) è negativo, il test al punto 3. è per minore anziché maggiore

#### Algoritmo

1. valutare `<expr1>`, `<expr2>`, `<expr3>` e ricordare i valori, rispettivamente, `val1` e `val2` e `val3`
2. inizializzare la <var> a `val1`
3. se il valore della `<var>` è maggiore di `val2` termina
4. eseguire il comando `<cmd>` del corpo
5. incrementare il valore di `<var>` del valore di `val3`
6. tornare al passo 3.

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Iterazione determinata

Iterazione controllata numericamente  
meno potente di quella indeterminata ma utile (pragmatica)

- **sintassi (generica)**
  - `for <var> = <expr1> to <expr2> by <expr3>  
do <cmd>`
  - `<var>` è la **variabile di controllo** del ciclo
  - `<expr1>` ed `<expr2>` rappresentano, rispettivamente, i valori iniziale e finale assunti dalla variabile
  - `<expr3>` rappresenta l'incremento progressivo della variabile, dopo aver eseguito il **corpo** `<cmd>`
- **vincolo (semantica statica)**
  - la variabile di controllo non può essere modificata nel corpo

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Iterazione determinata (cont.) nei diversi linguaggi

- modificabilità della var. di controllo (alcuni lo permettono → non è realmente it. determinata)
- numero di iterazioni
  - se `val1 > val2` il corpo non viene mai eseguito
    - alcuni linguaggi eseguono il test dopo l'esecuzione del corpo

`for <var> = <expr1> to <expr2> by <expr3>  
do <cmd>`

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Iterazione determinata (cont.)

**nei diversi linguaggi**

- **Val3 (passo negativo)**

- Necessario "costante non nulla" per il passo:  
in Algol e Pascal potrebbe anche essere negativo  
(downto o reverse)
  - `for <var> = <expr1> downto <expr2> by <expr3>`  
`do <cmd>`
- **Fortran:** evita sintassi diverse.
  - Il valore di `ic` (*iteration count*) viene calcolato e usato direttamente per controllare la fine del ciclo (`for`): termina se `ic` negativo (invece che usare il test su `var` ed `expr2`)

- **salto del ciclo**

- in alcuni linguaggi è permesso uscire prematuramente con un `goto`

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Iterazione determinata (cont.)

**espressività**

- Per esprimere n ripetizioni di un comando

- n fissato all'inizio del ciclo (non alla stesura del programma)
- Assegnazioni, sequenze, condizionale e `it.determinata`, non sono sufficienti (No Turing-complettezza: necessità di `It.Indeterminata`)
  - Es (computazione che non termina) uso di `while`, `goto` o chiamata ricorsiva:
    - $f(x) = x$  se  $x$  è pari
    - Non termina se  $x$  è dispari

- **It. indeterminata puo' simulare it.determinata**

N. Fanizzi ° Llinguaggi di Programmarazione (c) 2006

## Iterazione determinata (cont.)

**nei diversi linguaggi**

- **Quale valore finale della variabile di controllo**

- Se la var. di controllo è visibile fuori dal ciclo
  - si possono avere ambiguità o errori
    - Es: I di tipo intervallo e  
`for I =1 to 10 by 1 do`  
`corpo`
  - in Fortran e Pascal il val. è indefinito (dipende dalla particolare implementazione) → non portabilità
  - In Algol W e IV, e anche in Ada la var. è locale al ciclo (l'intestazione dichiara implicitamente la variabile di controllo, in base a `val1` e `val2`)

N. Fanizzi ° Llinguaggi di Programmarazione (c) 2006

## Iterazione determinata (cont.)

**espressività**

- **It. indeterminata puo' simulare it.determinata**

- **Ma:**

- For (invece del while) facilità la leggibilità e la manutenibilità del codice
- Prevenire errori (dimenticare l'inizializzazione, o l'incremento della variabile di controllo)
- For componibile più efficientemente (allocazione dei registri)

N. Fanizzi ° Llinguaggi di Programmarazione (c) 2006

## Iterazione determinata (cont.)

**espressività**

- **It. indeterminata puo' simulare it.determinata**

- **Ma:**
  - For (invece del while) facilità la leggibilità e la manutenibilità del codice
  - Prevenire errori (dimenticare l'inizializzazione, o l'incremento della variabile di controllo)
  - For componibile più efficientemente (allocazione dei registri)

N. Fanizzi ° Llinguaggi di Programmarazione (c) 2006

## Iterazione determinata (cont.)

**in C e suoi successori**

**Sintassi:** `for (<esp1>; <esp2>; <esp3>) <cmd>`

- **Semantica**

1. valuta `esp1`
2. valuta `esp2`; se è nulla termina
3. esegui il comando del corpo
4. valuta `esp3` e riprendi da 2.

- nessun vincolo sui valori delle espressioni né sulla modificabilità delle variabili coinvolte nelle espressioni (like-while)

- tuttavia le variabili di controllo sono spesso locali al ciclo (C++ e Java)

N. Fanizzi ° Llinguaggi di Programmarazione (c) 2006

## Iterazione determinata (cont.) in C e suoi successori

- in C anche un comando produce un valore (può essere visto come espressione)

• **for (<esp1>; <esp2>; <esp3>) <cmd>**

```
for i = inizio; i <= fine; i += passo{  
    comandi;  
}
```

```
i = inizio;  
while (i <= fine){  
    comandi;  
    i += passo  
}
```

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Programmazione strutturata

Metodologia per sviluppare codice (e quindi flusso di controllo) il più possibile strutturato

- progettazione top-down (o comunque gerarchica)
  - Codice sviluppato per raffinamenti successivi
- modularizzazione del codice
  - raggruppare codice che svolge una specifica funzione
    - comandi per l'astrazione del controllo (funzioni/procedure)
- uso di nomi significativi + uso estensivo dei commenti
  - essenziali per la leggibilità, la verifica e il riuso del codice
- uso di tipi strutturati per migliore progettazione e manutenzione del codice
  - aggregati di tipi anche differenti es. Record (raccolta di dati non omogenei: es: record studente)

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Programmazione strutturata

Metodologia per sviluppare codice (e quindi flusso di controllo) il più possibile strutturato

- uso di costrutti strutturati per il controllo
  - 1 punto di ingresso; 1 punto di uscita
  - Scansione lineare del testo ↔ flusso di esecuzione
    - C1;C2 con C1 e C2 arbitrariamente complessi
      - All'unica uscita di C1 il controllo passa all'unica entrata di C2
  - GOTO non permesso; break per anticipare l'uscita da un costrutto strutturato
  - Consente la nidificazione del codice
    - più leggibile e modificabile

Anticipo della programmazione in grande (in dipendenza dei meccanismi di astrazione realizzati, per creare l'indipendenza delle parti- contributo ing.sw)

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Ricorsione

alternativo all'iterazione

- in informatica: sottoprogramma che richiama se stesso
  - direttamente o indirettamente (mutua ricorsione)
    - P chiama Q che a sua volta chiama P (es. mutua ricorsione)

Calcolare il valore dell'n-esimo numero di fibonacci

(ovvero

$Fib(0)=Fib(1)=1$

$Fib(n)=Fib(n-1)+Fib(n-2)$  per  $n > 1$ )

```
int fib(int n) {  
    if (n==0) return 1;  
    else if (n==1) return 1;  
    else return fib(n-1) + fib(n-2);  
}
```

Allocazione e deallocazione di memoria a run-time (non necessariamente tutti i RdA devono essere attivi)

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## Ricorsione: esempio

```
int fatt (int n) {  
    if (n<=1)  
        return 1;  
    else  
        return n*fatt(n-1);  
}
```

puntatore di catena dinamica
indirizzo risultato
parametro n
risultato intermedio (fatt(n-1))

RdA semplificato per fatt(n)

- valore del risultato intermedio nel RdA per f(n) può essere determinato solo al termine della chiamata di f(n-1)

FATT(2)  
- e così via....

- quindi tutti i RdA devono coesistere

puntatore di catena dinamica
indirizzo risultato
parametro n=3
risultato intermedio (fatt(n-1))

FATT(3)  
N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

6

## Ricorsione: esempio

```
int fatt (int n) {  
    if (n<=1)  
        return 1;  
    else  
        return n*fatt(n-1);  
}
```

puntatore di catena dinamica
indirizzo risultato
parametro n
risultato intermedio (fatt(n-1))

RdA semplificato

- valore del risultato intermedio nel RdA per f(n) può essere determinato solo al termine della chiamata di f(n-1)

FATT(2)  
- e così via....

- quindi tutti i RdA devono coesistere

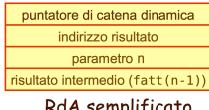
puntatore di catena dinamica
indirizzo risultato
parametro n=2
risultato intermedio (fatt(n-1))

FATT(3)  
N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

6

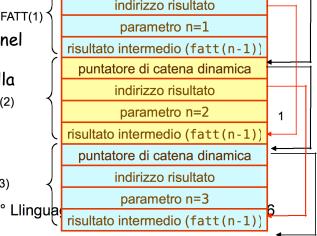
## Ricorsione: esempio

```
int fatt (int n) {
    if (n<=1)
        return 1;
    else
        return n*fatt(n-1);
}
```



- valore del risultato intermedio nel RdA per f(n) può essere determinato solo al termine della chiamata di f(n-1)
  - e così via...
  - quindi tutti i RdA devono coesistere

N. Fanizzi ° Llingua



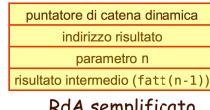
FATT(1)

FATT(2)

FATT(3)

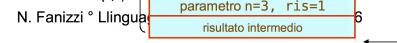
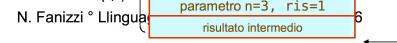
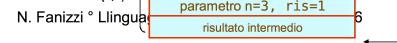
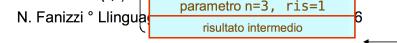
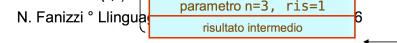
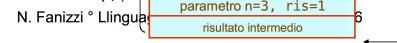
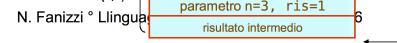
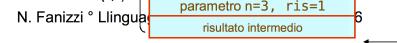
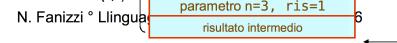
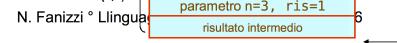
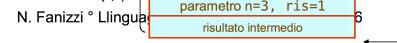
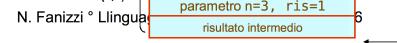
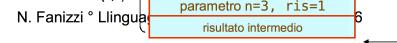
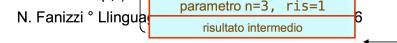
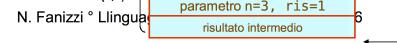
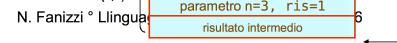
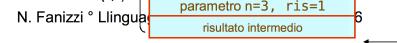
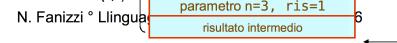
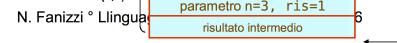
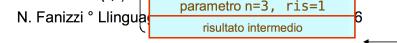
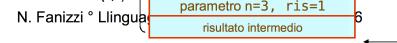
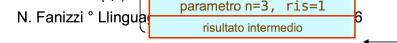
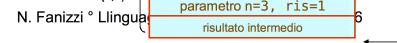
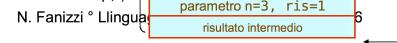
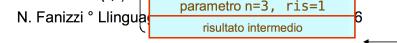
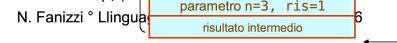
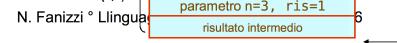
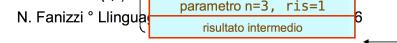
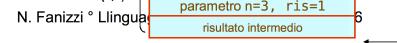
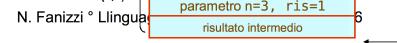
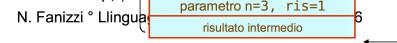
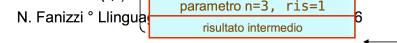
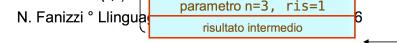
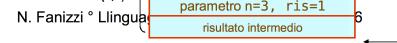
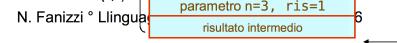
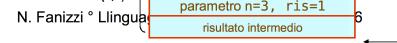
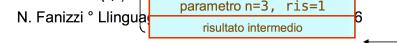
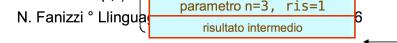
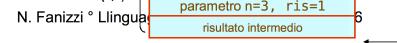
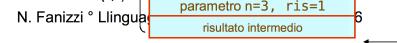
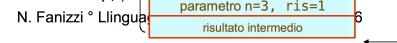
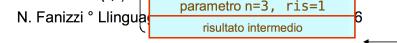
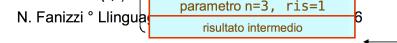
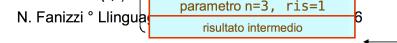
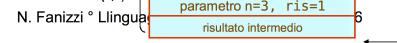
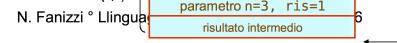
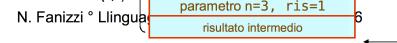
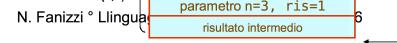
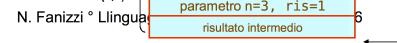
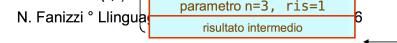
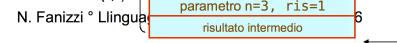
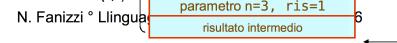
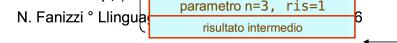
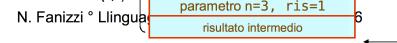
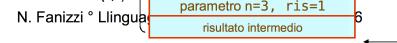
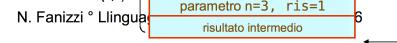
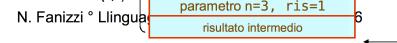
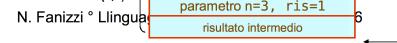
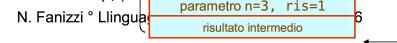
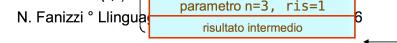
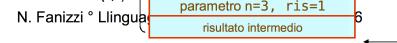
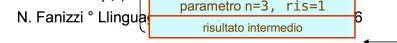
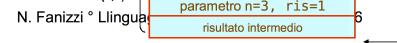
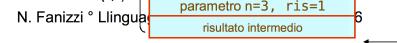
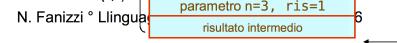
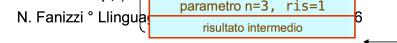
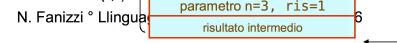
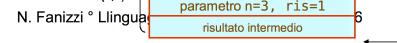
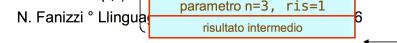
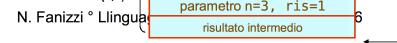
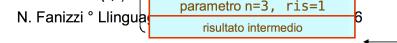
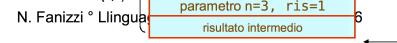
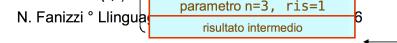
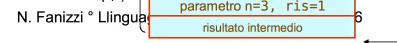
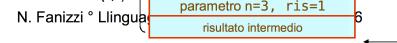
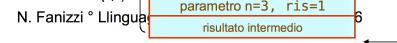
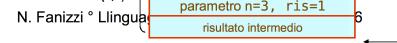
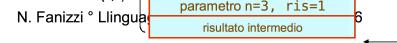
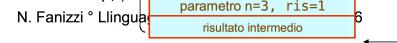
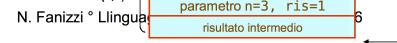
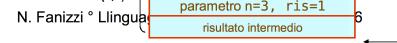
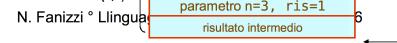
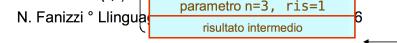
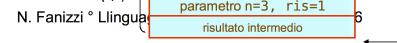
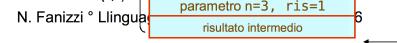
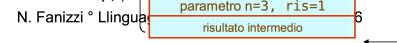
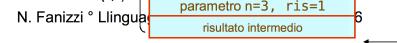
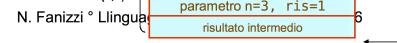
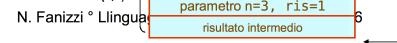
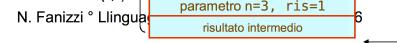
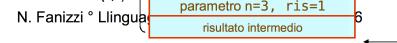
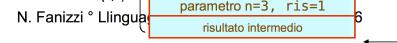
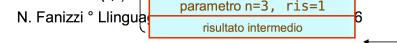
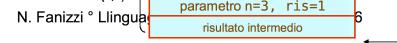
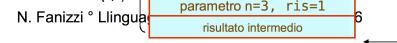
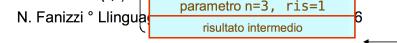
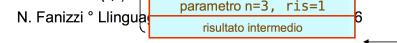
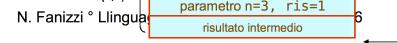
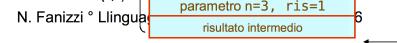
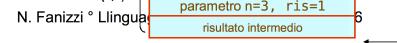
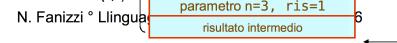
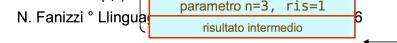
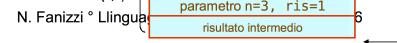
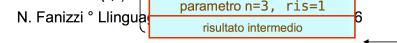
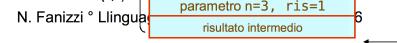
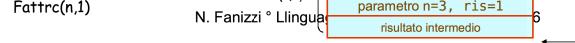
## Ricorsione: esempio2

```
int fattrc (int n, int ris) {
    if (n<=1)
        return ris;
    else
        return fattrc(n-1, n*ris);
}
```



- Una volta chiamto fattrc(n-1, n\*ris) tutte le informazioni necessarie al calcolo del risultato finale gli sono state passate come parametro
  - e così via...
  - quindi non è necessario che i RdA debbano coesistere

Fattrc(n,1)



## ricorsione in coda definizione

- Data una funzione f che nel suo corpo richiami g ( $g \leftrightarrow f$  oppure  $g=f$ ), la chiamata a g si dice chiamata in coda (*tail call*) se f **restituisce il valore restituito da g senza ulteriori computazioni**  
f si dice ricorsiva in coda (*tail recursive*) se tutte le sue chiamate ricorsive sono chiamate in coda
  - int f (int n) {
 if (n==0)
 return 1;
 else if (n==1)
 return f(0);
 else if (n==2)
 return f(n-1);
 else return f(n-1)\*2;
}
- N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

Vantaggio

Risparmio spazio di memoria  
(non vale per funzioni di ordine superiore)

Non in coda

## ricorsione in coda trasformazione

- In generale una funzione ricorsiva si può trasformare in una funzione ricorsiva in coda equivalente con opportune complicazioni
  - anticipare tutta la computazione che avverrebbe dopo la chiamata ricorsiva
  - quello che non può essere anticipato va passato alla chiamata attraverso parametri aggiuntivi
    - ris nell'esempio precedente

N. Fanizzi ° Llinguaggi di Programmazione (c) 2006

## ricorsione in coda esempio2 di trasformazione

```
int fib(int n) {
    if (n==0) return 1;
    else if (n==1) return 1;
    else return fib(n-1) + fib(n-2);
}
```

RICORSIVA

FIB(1)  
FIB(2)  
FIB(3)

FIB(1)  
FIB(2)  
FIB(3)

FIB(1)

FIB(1)