

# Strutturare il Controllo (della valutazione)

Sommario: 17 marzo, 2015

- Controllo della sequenza: Linguaggi Machine-Level, High-level e Valori modificabili
- Espressioni: Valutazione e Controllo della sequenza
- Comandi vs. Espressioni: Assegnamento
- Comandi: Controllo della sequenza
- Comandi Strutturati: goto e comandi per iterazione
- Programmazione Strutturata: Antesignana delle moderne Metodologie
- Ricorsione: Programmazione con induzione, Tail Recursion e Memoization

# Controllo della sequenza: Linguaggi Machine-Level, e Valori Modificabili

- **Linguaggi Machine-Level:**

- La struttura di un programma è una sequenza di statement atomici:

```
Mov L0, R0
```

```
Add R1, R0
```

```
Mov R1, L0
```

- che usano Locazioni come Valori Modificabili
- La sequenza di valutazione è facilmente controllata dal registro PC.
- questa sequenza è determinante quando sono coinvolti Valori Modificabili

- Linguaggi High-Level:

# Controllo della sequenza: Linguaggi High-level e Valori Modificabili

- Linguaggi Machine-Level:
- **Linguaggi High-Level:**
  - La struttura di un programma può non essere una seq.
  - Gli statements possono non essere atomici
    - ma possono usare Variabili come Valori Modif.
  - La sequenza di valutazione dipende dalla semantica.
- Semantica e Controllo di sequenza.

## Osservazione

*La Semantica ci dice:*

*Come stms non atomici sono decomposti in stm più semplici e in quale sequenza questi ultimi devono essere considerati e valutati*

**Espressioni** sono presenti in tutti gli High-Level L.

- Semantica: In tutti i linguaggi, la sem. prevede che calcolino un valore, o risultino in una computazione indefinita (non-terminante).
- Struttura:
  - Un operatore (principale) ed  $n \geq 0$  argomenti (a loro volta espressioni)
  - Un'invocazione di funzione applicata a  $n \geq 0$  argomenti (a loro volta espressioni)
  - Un termine atomico (variabile, valore atomico)
- Controllo di sequenza: segue l'ordine e il metodo di valutazione degli argomenti
  - Ordine: leftmost, rightmost, associatività dx o sin.
  - Metodo:
    - Operatori: stretto, corto-circuito,
    - Invocazioni: trasmissione parametri

**Espressioni** sono presenti in tutti gli High-Level L.

- **Ordine** di valutazione: *leftmost, rightmost, assoc. dx o sin.*
  - critico quando:
    - operatori su aritmetica finita (virgola mobile)
    - effetti laterali (modifica di variabili):  
Es.  $(x=3)+(y=x)$
- **Metodo** di valutazione: *stretto, corto-circuito*
  - critico quando:
    - argomento indefinito.  
Es. Siano:
      - e valuta indefinito,  
 $(\lambda x.\lambda y.\text{if } (x > 0) \text{ then } x \text{ else } x + y)15$  e  
può calcolare 15 se non stretta sul primo a.
      - $e_1$  o  $e_2$  v. indefinito e l'altra valuta **false**.  
 $e_1 \text{ Or } e_2$   
può valutare **false** se **Or** è valutata a corto-circuito

## Assegnamento

- è un'espressione in alcuni linguaggi (C, Java).
- è un comando in altri (Pascal).
- Semantica:
  - in tutti i casi modifica un valore modificabile
  - Linguaggi differenti possono usare differenti ordini di valutazione degli argomenti.
  - conducendo a sequenze di valutazioni compl. diverse e comp. compl. diversi.

```
b = 0;  
a[f(3)] = a[f(3)] +1
```

per f così definito:

```
int f(int 0){  
    if (b==0){b = 1; return 1;}  
    else return 2;  
}
```

- usare ora ordine leftmost ora rightmost.

## Assegnamento:

- Gli argomenti di un assegnamento devono essere un valore modificabile (l-value) e un valore memorizzabile (r-value)
  - valore modificabile = Locazione di memoria
  - valore memorizzabile = valore assegnabile ad una locazione nella memoria
  - valore denotabile = valore associabile ad un identificatore nell'ambiente
  - valore esprimibile = valore che può essere calcolato con un'espressione (non singola variabile)
- Alcune espressioni possono essere valutate per ottenere un l-value oppure un r-value

$$a[f(3)] = a[f(3)] + 1$$

**Comandi** sono presenti in tutti i Linguaggi Prescrittivi (Macchina, Imperativi,...)

- Semantica: In tutti i linguaggi, la sem. prevede che modifichino la memoria (effetti laterali).
- Struttura. Varie classi di comandi:
  - Controllo di sequenza esplicito
  - Condizionali (o di selezione)
  - Iterativi



# Comandi di Sequenza Esplicito

Introducono una sequenza di C, o alterano la sequenza corrente.

- Includono:
  - Comando Sequenza:  $C_1; C_2$
  - Blocco inline
  - goto, break, continue, return.
- Comando Sequenza e Blocco inline sono presenti in tutti i Linguaggi con comandi ed hanno una semantica composizionale ottenuta dalla naturale composizione della semantica dei componenti.
- goto: Iterazione non strutturata
  - oscuro e poco espressivo
  - può sempre essere evitato a favore di iterazione strutturata
  - va evitato
- break, continue, return
  - Utilizzati in C per uscita non strutturata da blocchi.

Discriminano tra due o più sequenze alternative

- Includono:

- if-then-else e if-then

- con struttura e comportamento analogo in tutti i linguaggi

- case e switch

- con struttura simile ma comportamento, a volte, diverso

```
case 2 of 1 : C1; 2 : C2; 3 : C3; 4 : C4; end; C;
```

```
switch (2){case 1 : C1; case 2 : C2; case 3 : C3; case 4 : C4; }C;
```

conducono alla sequenza:

```
C2; C;
```

```
C2; C3; C4; C;
```

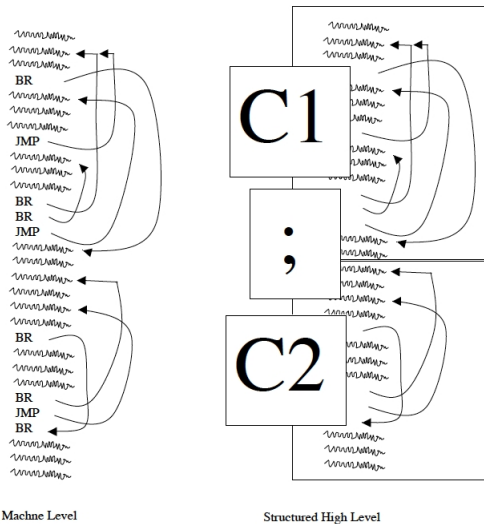
rispettivamente.

# Comandi Iterativi (strutturati)

Incapsulano ed evidenziano la sequenza da iterare

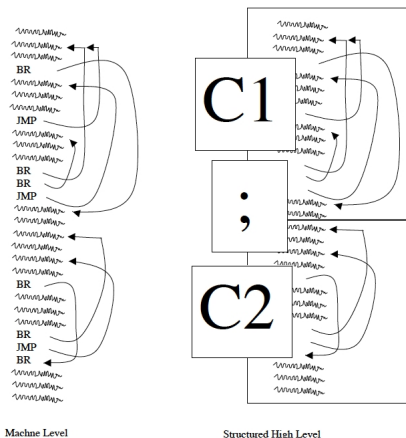
- Si differenziano in questo dall'iterazione non strutturata basata sul goto
- Si suddividono in:
  - iterazione indeterminata: while, repeat, for (del C)
  - iterazione determinata: for del Pascal
- Iterazione indeterminata: **while** E **do** C
  - il numero di iterazioni può non essere determinato utilizzando lo stato iniziale e l'espressione di controllo E
  - l'iterazione può anche non terminare
  - **while** (x>1 and x<10) **do** x = 5;
- Iterazione determinata: **for** i = E<sub>inizio</sub> **to** E<sub>fine</sub> **do** C
  - il numero di iterazioni è:  $n = v_{E_{fine}} - v_{E_{inizio}}$

# Costrutto Strutturato: Sequenze chiuse



- Possiamo sempre "estendere" una sequenza atomica con JMP e BR in una sequenza atomica equivalente formata di sole sezioni "chiusse"

# Costrutto e Programma Strutturato



- **Costrutto Strutturato:** un costrutto non atomico che definisce una sequenza atomica "chiusa", ovvero ha un unico punto di ingresso (il primo atomo) ed un unico punto di uscita (l'ultimo atomo).
- **Linguaggio, Programma Strutturato:** Le funzioni calcolabili possono essere espresse con programmi che utilizzano solo costrutti strutturati (Bohm-Jacopini, 1966)

# Costrutti non strutturati

- **Costrutto Strutturato:** un costrutto non atomico che definisce una sequenza atomica "chiusa", ovvero ha un unico punto di ingresso (il primo atomo) ed un unico punto di uscita (l'ultimo atomo).
- **Costrutto Strutturato:** un costrutto non atomico che definisce una sequenza atomica "chiusa", ovvero ha un unico punto di ingresso (il primo atomo) ed un unico punto di uscita (l'ultimo atomo).
- **Linguaggio, Programma Strutturato:** Le funzioni calcolabili possono essere espresse con programmi che utilizzano solo costrutti strutturati (Bohm-Jacopini, 1966)
- **Rispondiamo:**
  - Quali dei seguenti costrutti sono strutturati e perchè: (a) Goto; (b) break; (c) continue; (d) return
  - Il costrutto Switch del C è un costrutto strutturato?

## Antesignana delle moderne metodologie di sviluppo

- Progettazione top-down del programma:
  - Una Specifica Astratta iniziale che mostra funzionalità attese e correlazioni tra esse
  - raffinamenti successivi che aggiungono dettagli e ripetono il processo su ciascuna funzionalità
- Modularizzazione del codice:
  - Ogni codice che implementa una funzionalità deve essere localizzato in una parte precisa del programma
  - che deve essere acceduta e trattata come un unità di programmazione indivisibile (un modulo)
- Uso di identificatori significativi:
  - Identificatori adatti a indicare ruolo ed agevolare la comprensione del codice dove sono usati
- Uso estensivo di commenti
  - Commenti adatti a fornire indicazioni utili su vincoli ed utili per il testing, la modifica e la manutenzione del codice

## Antesignana delle moderne metodologie di sviluppo

- Progettazione top-down del programma:
- Modularizzazione del codice:
- Uso di identificatori significativi:
- Uso estensivo di commenti
- Uso di Tipi di dato strutturato
  - adatti a rappresentare in modo unitario valori aventi strutture anche complicate
- Uso di costrutti strutturati che devono:
  - mostrare in modo chiaro la trasformazione calcolata.



# Progr. Strutturata: Applichamola al Quicksort /1

La specifica astratta iniziale può coincidere con la descrizione del problema, vincoli e soluzione e/o algoritmo.

Ad esempio:

- Obiettivo Generale: Programma di Ordinamento
- Obiettivo Specifico: Quicksort per sequenze ordinabili
- Vincoli:
  - Una sequenza **c**
  - di valori ordinabili **t**
  - Quindi, **c** è un valore di tipo **Seq(t)**, sequenze di tipo **t**.
- Soluzione/Algoritmo:
  - a Sia **cc** la sequenza corrente da ordinare
  - b Sia **a** di tipo **t**, un elemento di separazione per **cc**
  - c Dividiamo **cc** in due sottosequenze **cLE** e **cGT** separate da **a**
  - d Ripetere [a]-[d] su ogni sottosequenza non singoletta, ottenuta al passo [c] se presente.
  - e Terminiamo, presentando la sequenza ottenuta.

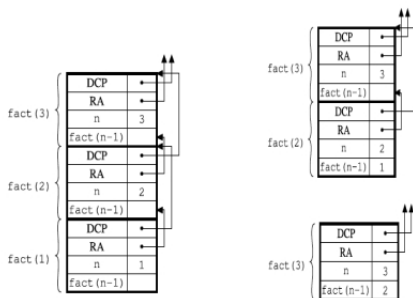
Procediamo con  $N$  raffinamenti successivi, individuando funzionalità e aggiungendo dettagli

- Hints: Chiarire e precisare dettagli rispondendo a domande, quali:
- Cosa si deve intendere per elemento di separazione?
- Cosa si deve intendere per divisione di sequenza rispetto a elemento di separazione, dato un ordinamento?
- ...

# Ricorsione

- Algoritmi definiti in modo induttivo
- Introdotta attraverso Procedure/Funzioni
- Possono richiedere l'allocazione di una grande quantità di AR

```
int fact(int n){  
    if (n<0) return 0;  
    else if (n==0) return 1;  
    else return n * fact(n - 1);  
}
```



- Dire chi sono i campi: DCP, RA presenti negli AR e di quale campo fa parte l'entry "fact(n-1)..."

# Ricorsione: Funzioni Parziali e Gestione delle Eccezioni

- Possiamo ridurre l'allocazione di AR al solo AR dell'invocazione corrente
- Ricorsione di coda: È una tecnica che lo permette
- Prima però commentiamo la definizione data per fact.

```
int fact(int n){ /* fact calcola 0 se n e' negativo, altrimenti n! */
    if (n<0) return 0;
    else if (n==0) return 1;
    else return n * fact(n - 1);
}
```

- C non ha un meccanismo di eccezioni che sarebbe stato preferibile a questo uso arbitrario (e oscuro) del valore 0
- Il commento inserito è d'obbligo ma non risolve il problema
- la definizione sotto è una corretta definizione della funzione **parziale** n!.

```
int fact(int n){ /* fact diverge se n e' negativo, altrimenti n! */
    if (n==0) return 1;
    else return n * fact(n - 1);
}
```

- Noi vogliamo usare proprio quella parziale nei nostri programmi?
- Nella quasi totalità dei casi NO
- Vorremo usare una definizione:  
che ci dica quando le cose non vanno come atteso,  
ovvero, faccia uso di un *meccanismo di eccezioni*.

# Ricorsione: Ricorsione di Coda

- Possiamo ridurre l'allocazione di AR al solo AR dell'invocazione corrente
- Ricorsione di coda: È una tecnica che lo permette

```
int fact(int n){ /* fact calcola 0 se n e' negativo, altrimenti n! */
    if (n<0) return 0;
    else if (n==0) return 1;
    else return n * fact(n - 1);
}
```

## Definition (Invocazione di Coda e Ricorsione di coda)

Sia  $F$  una definizione di funzione che contiene un'invocazione di una funzione  $g$ . Tale invocazione è detta invocazione di coda se, quando applicata,  $F$  restituisce il valore calcolato da tale invocazione senza ulteriore calcolo. Una definizione ricorsiva  $F$  è con ricorsione di coda se ogni sua invocazione contenuta in  $F$  è una invocazione di coda.

- la definizione di `fact` non è con ricorsione di coda
- questa sotto lo è

```
int factT(int n, int acc){ /* acc produttoria argomenti n di invocazioni precedenti */
    if (n<0) return 0;
    else if (n==0) return acc;
    else return fact(n - 1, n * acc);
}
```

- Per ogni intero  $n$ , `factT(n, 1)` calcola  $n!$ .

## Invocazione di funzione con Ricorsione di Coda.

- Sia  $g(e_0)$  l'invocazione di una tale funzione, con  $e_0$  (lista degli) argomenti dell'applicazione.
- L'invocazione può condurre a due casi possibili:
  - Calcola un valore senza richiedere ulteriori invocazioni di  $g$ . Caso finale
  - Conduce ad un'invocazione di coda  $g(e_1)$ . Allora  $g(e_1)$  è tutto ciò che serve per calcolare  $e_0$ 
    - Possiamo rimpiazzare l'invocazione  $g(e_0)$  con  $g(e_1)$ , ovvero
    - Possiamo, rimpiazzare il contenuto dell'AR prodotto per calcolare  $g(e_0)$  con quello per  $g(e_1)$

# Ricorsione di Coda: Activation Record di factT

- Sia  $g(e_0)$  l'invocazione di una funzione con ricorsione di coda, con  $e_0$  (lista degli) argomenti dell'applicazione.
- L'invocazione può condurre a due casi possibili:
  - Calcola un valore senza richiedere ulteriori invocazioni di  $g$ . Caso finale
  - Conduce ad un'invocazione di coda  $g(e_1)$ . Allora  $g(e_1)$  è tutto ciò che serve per calcolare  $e_0$ 
    - Possiamo rimpiazzare l'invocazione  $g(e_0)$  con  $g(e_1)$ , ovvero
    - Possiamo, rimpiazzare il contenuto dell'AR prodotto per calcolare  $g(e_0)$  con quello per  $g(e_1)$

Applichiamolo a  $\text{factT}(3,1)$ , definita sotto

```
int factT(int n, int acc){ /* acc produttoria argomenti n di invocazioni precedenti */
    if (n<0) return 0;
    else if (n==0) return acc;
    else return factT(n - 1, n * acc);
}
```

|                 |     |
|-----------------|-----|
| ...             | ... |
| CD              | ... |
| RA              | ... |
| n               | 3   |
| acc             | 1   |
| fact(n-1,n*acc) |     |

|                 |     |
|-----------------|-----|
| ...             | ... |
| CD              | ... |
| RA              | ... |
| n               | 2   |
| acc             | 3   |
| fact(n-1,n*acc) |     |

|                 |     |
|-----------------|-----|
| ...             | ... |
| CD              | ... |
| RA              | ... |
| n               | 1   |
| acc             | 6   |
| fact(n-1,n*acc) |     |