Corso di Programmazione

Sottoprogrammi

Procedure e funzioni

Prof.ssa Teresa Roselli

roselli@di.uniba.it

Funzione

Sottoprogramma che ha come risultato il calcolo di un valore:

- è dotata di nome (*designatore* di funzione) a cui viene associato un “valore di ritorno”
- per il motivo precedente deve essere dotata di *tipo* che va dichiarato nell’intestazione della funzione
- per lo stesso motivo, nel corpo della funzione, il nome deve comparire come parte sinistra di un assegnazione per attribuire ad esso il valore da trasmettere al programma chiamante
- La chiamata di una funzione avviene inserendo il suo nome in una espressione (non può mai trovarsi a sinistra di una assegnazione nel programma chiamante)
- Una funzione ha *parametri* come una procedura

Funzione

Il nome di una funzione

- a destra di una assegnazione rappresenta il *valore* della funzione
- a sinistra di una assegnazione individua la *locazione di memoria* (corrisponde al concetto di variabile e può apparire solo all'interno della funzione stessa

Funzioni

- Indipendentemente dal linguaggio di programmazione una dichiarazione di funzione prevede un costrutto linguistico del tipo

funzione <identificatore> <lista di argomenti> **:** <tipo risultato>

  - Gli argomenti rappresentano i parametri di entrata
    - Andrebbero sempre passati per valore al fine di evitare effetti collaterali
  - Come nel caso delle procedure, è possibile definire all'interno della funzione una sezione dichiarativa con variabili locali

Funzioni
esempio

- Dichiarazione della funzione *resto*

```
FUNZIONE resto(a:integer;b:integer):integer
        BEGIN
                resto ⟵ a - (a DIV b) * b
        END
```

- Chiamata della funzione *resto* nel programma chiamante

```
.
.
residuo ⟵ resto(x,y)
.
.
```

Procedure vs. Funzioni

	Procedure	**Funzioni**
Tipo associato		X
Parametri	X	X
Possibilità di modificare il valore dei parametri	X	
Valore di uscita		X
Chiamata	autonoma	in un’espressione

SIDE EFFECT
effetti collaterali in procedure e funzioni

- Il side effect si verifica quando a seguito dell'attivazione di un sottoprogramma si ha una modifica delle variabili non locali al sottoprogramma (i parametri passati per riferimento e le variabili non locali possono esportare valori al di fuori del sottoprogramma)
- Il side effect è accettabile nelle procedure ma deve essere evitato nelle funzioni poiché produrrebbe più di un valore di ritorno.
- L'utilizzo di variabili globali impedisce di considerare il sottoprogramma come una entità completa e autoconsistente poiché non fa riferimento esclusivamente alle sue variabili, tipi e costanti.

Sottoprogrammi come Parametri

- Nella classe dei parametri di sottoprogrammi rientrano anche procedure e funzioni
  - Un sottoprogramma *F* può essere usato come parametro di un altro sottoprogramma *G*, quando *F* deve essere eseguito durante l'esecuzione di *G*
- Il parametro formale corrispondente ad un sottoprogramma
  - Riporta l'intestazione
    - I nomi del sottoprogramma e dei parametri possono cambiare
  - Deve avere parametri passati esclusivamente per valore

Sottoprogrammi come Parametri

- All'invocazione di un sottoprogramma avente come parametri altri sottoprogrammi
  - Il corrispondente parametro effettivo deve essere l'identificatore di una procedura o di una funzione con i medesimi requisiti riguardo a parametri o tipo del risultato
  - Durante l'esecuzione del corpo del sottoprogramma invocato, ogni occorrenza del parametro formale implica l'uso corrispondente del sottoprogramma fornito come parametro effettivo

Sottoprogrammi come Parametri esempio

Il sottoprogramma

```
sottoprogramma s(sottoprogramma p(x,a))
    begin
        p(x,a)
    end
```

può essere invocato come segue

```
s(pinco(m,n))
```

Attivazione di Sottoprogrammi

- Nei linguaggi tipo Pascal l'attivazione di sottoprogrammi è gestita tramite *pila*
  - Ad ogni attivazione di un'unità di programma
    - Viene creato un *record di attivazione*
    - Il record viene messo in cima alla pila
  - Al termine dell'attivazione
    - Il record è tolto dalla pila
    - La memoria viene rilasciata
    - Si perdono i legami tra parametri

Record di Attivazione
Informazioni contenute e dimensioni

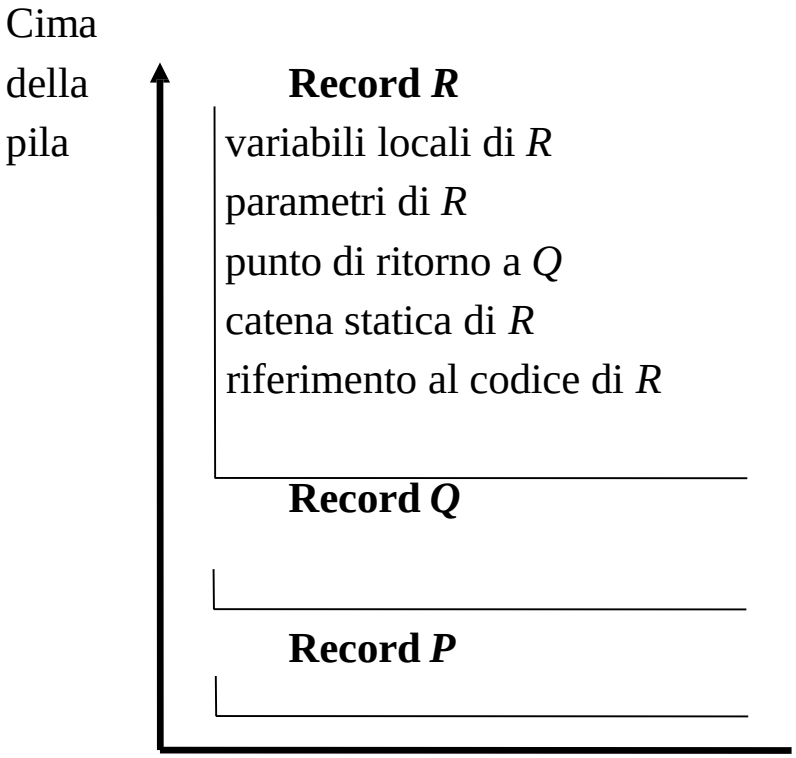
- Nome dell'unità di programma
- Riferimento all'area di memoria in cui è memorizzato il corpo di istruzioni da eseguire
- Punto di ritorno
  - Riferimento all'istruzione a cui tornare al termine dell'attivazione
- Gerarchia creata al momento della dichiarazione (*catena statica*)
- Parametri formali e loro legame con i corrispondenti parametri effettivi (dipende dal tipo di passaggio: il passaggio per referenza di un array fa occupare meno memoria rispetto al passaggio per valore)
- Variabili locali con riferimento alle aree di memoria allocate (dipende dal numero e dal tipo di variabili)

Concatenazione

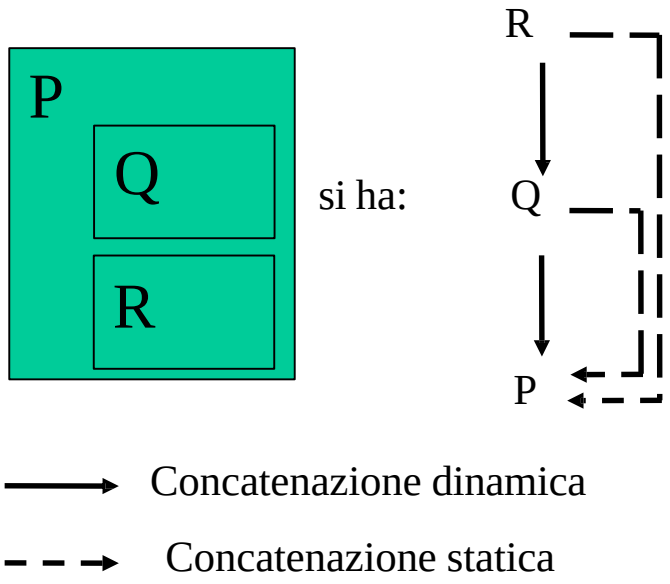
- Statica (può essere determinata guardando il testo del programma)
  - Definita dalla nidificazione nella dichiarazione delle procedure
    - non può variare
    - fornisce i riferimenti nella pila alle variabili non locali
- Dinamica (dipende dall'esecuzione)
  - Realizzata attraverso la sequenzializzazione delle attivazioni nella pila
    - Cambia a seconda dell'evolversi dell'esecuzione

Concatenazione
Esempio

- *P* ha attivato *Q* che ha attivato *R*



- Nel caso di dicharazioni nidificate



Gestione dell'Esecuzione

- Instruction pointer *IP*
  - Individua l'istruzione in corso di esecuzione nel programma
    - Riferita all'area di memoria associata al programma
- Environment pointer *EP*
  - Individua l'ambiente di lavoro
    - Risorse della porzione di programma attualmente in esecuzione

Gestione dell'Esecuzione

- Inizialmente:
  - *IP* punta al **begin**
  - *EP* non ha un ambiente cui puntare
- Nel momento in cui inizia l'esecuzione
  - Vengono allocate le variabili richieste dal programma principale
  - Le variabili vengono reperite ed usate nell'ambiente puntato da *EP*

Gestione dell'Esecuzione

- Al momento dell'attivazione di un sottoprogramma
  - Vengono allocate nuove variabili
    - Fanno parte di un nuovo ambiente
  - *EP* punta a questo ambiente
    - Il sottoprogramma userà le variabili puntate dall'*EP* attuale
    - Se non le trova, risalirà negli ambienti attivati precedentemente fino a trovare la prima occorrenza delle variabili

Gestione dell'Esecuzione

- Al termine dell'esecuzione di un sottoprogramma
  - *EP* punta all'ambiente precedente nella catena di attivazioni
  - Le variabili locali relative al sottoprogramma vengono distrutte

Ricorsione

- Si dimostra che
  - Ogni problema ricorsivo è computabile per mezzo di un programma

  e, viceversa,
  - Ogni problema computabile per mezzo di un programma è esprimibile in forma ricorsiva
- Le scomposizioni ricorsive implicano, a livello di codice, programmi in grado di invocare se stessi
  - procedure o funzioni ricorsive

Ricorsione

- Gestita come una normale attivazione di procedura o funzione
  - Disciplina a stack
  - Per ogni variabile *v* locale alla procedura *R*
    - Una chiamata di *R* che dia vita alla generazione di *k* chiamate ricorsive produrrà *k*+1 distinte istanze della variabile *v*
    - Il tempo di vita di ciascuna di esse è contenuto (innestato) in quello delle altre che la precedono

Ricorsione
Esempio

- Calcolo del fattoriale

$$n! = n * (n - 1) * (n - 2) * \ldots * 2 * 1$$

– esprimibile ricorsivamente come

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n * (n - 1)! & \text{se } n > 0 \end{cases}$$

Ricorsione
Esempio

- Sia nfatt(n) la funzione che calcola il fattoriale di *n* per ogni intero $n \geq 0$

  Se n = 0

  allora nfatt(n) = 1

  altrimenti nfatt(n) = n * nfatt(n – 1)

- L'esecuzione della funzione *nfatt* causa chiamate ricorsive della stessa funzione
  - Sovrapposizione di ambienti di programmazione nello stack in fase di esecuzione

Ricorsione
Esempio

- 4!: poiché 4 > 0, 4! = 4 * 3! = 4 * ?…
  - 3!: poiché 3 > 0, 3! = 3 * 2! = 3 * ?…
    - 2!: poiché 2 > 0, 2! = 2 * 1! = 2 * ?…
      - 1!: poiché 1 > 0, 1! = 1 * 0! = 1 * ?…
        - » 0!: è noto che 0! = 1
      - … = 1 * 1 = 1
    - … = 2 * 1 = 2
  - … = 3 * 2 = 6
- … = 4 * 6 = 24

Ricorsione
Esempio

memoria istruzioni	
FUNZIONE	p=6
nfatt(n:integer):integer	
BEGIN	p=5
if n=0 then nfatt←1	
else nfatt← n*nfatt(n-1)	p=4
END	
BEGIN (del main)	p=3
leggi n	
fatt← nfatt(n)	p=2
scrivi fatt	
END	p=1
	p=passo