Corso di Programmazione

Sottoprogrammi

Procedure e funzioni

Prof.ssa Teresa Roselli

teresa.roselli@uniba.it

Programmazione Modulare

- Tecnica basata sul metodo di scomposizione di un problema in sottoproblemi logicamente indipendenti tra loro
  - Ad ogni sottoproblema corrisponde un modulo
    - Codificati separatamente
    - Compilati separatamente (talvolta)
    - Integrati solo alla fine per formare il programma complessivo

Programmazione Modulare

- Un problema caratterizzato da
  - un algoritmo *A*
  - che opera sull'insieme dei dati di partenza *D*
  - per produrre l'insieme dei risultati *R*

  viene suddiviso in un insieme finito di *n* sottoproblemi a differenti livelli caratterizzati dalla tripla

$$(D_i, A_i, R_i)$$

  - Interazione e ordine di esecuzione degli algoritmi secondari per ottenere la soluzione del problema originario gestita da un algoritmo coordinatore

Programmazione Modulare

- Algoritmo coordinatore → programma principale o main
- Algoritmi secondari → sottoprogrammi
  - Diversi livelli
  - Si costruisce una gerarchia di macchine astratte, ciascuna delle quali
    - Realizza un particolare compito in modo completamente autonomo
      - Proprie definizioni di tipi, dichiarazioni di variabili e istruzioni
    - Fornisce la base per il livello superiore
    - Si appoggia su un livello di macchina inferiore (se esiste)

  Il programma è visto come un *nuovo operatore* disponibile sui dati

  L'*astrazione funzionale* è la tecnica che permette di ampliare il repertorio di operatori disponibili

Programmazione Modulare
Tecniche per individuare i sottoproblemi

- Basate sul metodo di soluzione di problemi consistente nello scomporre un problema in sottoproblemi più semplici
  - Sviluppo top-down
    - Approccio step-wise refinement
  - Sviluppo bottom-up
  - Sviluppo “a sandwich”

Raffinamento per passi successivi

- Basato su
  - Raffinamento di un passo della procedura di soluzione
    - Legato alle modalità di esecuzione conseguenti una certa suddivisione in sottoproblemi
      - Necessario concentrarsi sul “cosa” piuttosto che sul “come”
  - Raffinamento della descrizione dei dati
    - Definizione della struttura e tipo
    - Definizione delle modalità di comunicazione
      - Come renderli comuni a più sottoproblemi

Sviluppo Top-Down

- Costruzione del programma per livelli successivi
  - Corrispondenza con la scomposizione del problema cui è relativo
    - Strumento concettuale per la costruzione di algoritmi
  - Dettagliamento successivo delle parti in cui viene scomposto (sottoprogrammi) fino al codice finale
    - Strumento operativo per l'organizzazione e lo sviluppo di programmi complessi
- Metodo *trial and error*
  - Prova e riprova alla ricerca della scomposizione ottimale

Sviluppo Bottom-Up

- Partendo dalle istruzioni del linguaggio
  - Costruzione di programmi molto semplici
  - Collegamento successivo in programmi più complessi

  fino ad ottenere il programma finale
- Usato soprattutto nell'adattamento di algoritmi codificati già esistenti a nuove situazioni

Metodo a Sandwich

- Basato su una cooperazione fra le tecniche top-down e bottom-up
  - Necessità di raffinare via via la soluzione del problema principale
    - Scomposizione in algoritmi che ne risolvono delle sottoparti
  - Disponibilità di algoritmi di base per problemi semplici
    - Raggruppamento in algoritmi via via più complessi

Sottoprogramma

- Corrisponde all'algoritmo secondario che risolve un sottoproblema
- Insieme di istruzioni
  - Individuate da un nome
  - Che concorrono a risolvere un problema
    - Ben definito
      - Sensato
    - Non necessariamente fine a se stesso
      - è di supporto per la risoluzione di problemi più complessi
      - rappresenta una funzionalità a se stante, una unità concettuale con un significato più ampio (prescinde dal problema presente)
- Esempi:
  - Scambio, Ricerca del Minimo, Ordinamento, …

Sottoprogrammi
Utilità

- Un programma viene strutturato in sottoprogrammi:
  - Per rispettare la decomposizione ottenuta con il metodo di progettazione dell'algoritmo
  - Per strutturare in maniera chiara l'architettura del programma
  - Perché lo stesso gruppo di istruzioni deve essere ripetuto più volte in diversi punti del programma (blocchi ripetibili)

Sottoprogrammi
Utilità

- Risponde alla necessità di risolvere uno stesso problema
  - Più volte
    - All'interno dello stesso programma
    - In programmi diversi
  - Su dati eventualmente diversi
- Unicità dello sforzo creativo

Sottoprogrammi
Utilità

- Stile e qualità del software
  - Leggibilità
  - Manutenibilità
  - Trasportabilità
  - Modularità
  - Reuso

  I sottoprogrammi giocano un ruolo fondamentale nella tecnica della programmazione

Sottoprogrammi

- SOTTOPROGRAMMA è una astrazione funzionale che consente di individuare gruppi di istruzioni che possono essere *invocate* esplicitamente e la cui chiamata garantisce che il *flusso di controllo* ritorni al punto successivo all'invocazione

Astrazioni Funzionali

- Fornite dai linguaggi di programmazione ad alto livello
  - Consentono di creare unità di programma (macchine astratte)
    - Dando un nome ad un gruppo di istruzioni
    - Stabilendo le modalità di comunicazione tra l'unità di programma creata ed il resto del programma in cui essa si inserisce
  - Assumono nomi diversi a seconda del linguaggio di programmazione
    - Subroutine
    - Procedure
    - Sub program
    - …

Astrazioni Funzionali

- Paragonabili a nuove istruzioni che si aggiungono al linguaggio
  - Definite dall'utente
    - Specifiche per determinate applicazioni o esigenze
  - Più complesse delle istruzioni base del linguaggio
    - Analogia con il rapporto fra linguaggi ad alto livello e linguaggio macchina
  - Ciascuna risolve un ben preciso problema o compito
    - Analogia con un programma

Astrazioni Funzionali

- Struttura risultante di un programma:

  Intestazione di programma
    Definizione di tipi
    Dichiarazioni di variabili
    Dichiarazione di macchine astratte (sottoprogrammi)
  Corpo di istruzioni operative del programma principale

  – La dichiarazione di una macchina astratta rispecchia le regole di struttura di un programma

Sottoprogramma

- Indipendentemente dalle regole sintattiche del particolare linguaggio di programmazione
  - Individuabile con un nome
    - Identificatore
  - Prevede l'uso di un certo insieme di risorse
    - Variabili, costanti, …
  - Costituito da istruzioni
    - Semplici o, a loro volta, composte (altre macchine astratte)
  - Differisce da un programma nelle istruzioni di inizio
    - Specificano che (e come) altri pezzi di programma possono utilizzarlo

Chiamata di Sottoprogrammi

- Provoca l'esecuzione delle istruzioni del sottoprogramma
  - Modalità: deve essere comandata dal programma chiamante
    - Specifica del nome associato
  - Effetto: si comporta come se il sottoprogramma fosse copiato nel punto in cui è stato chiamato
    - Eliminazione di ridondanza

Chiamata di Sottoprogrammi

- All’atto dell’attivazione (su chiamata) dell’unità di programma
  - Viene sospesa l’esecuzione del programma (o unità) chiamante
  - Il controllo passa all’unità attivata
- All’atto del completamento della sua esecuzione
  - L’attivazione termina
  - Il controllo torna al programma chiamante

Sottoprogrammi
Nidificazione

- Le risorse di cui fa uso un sottoprogramma possono includere altri sottoprogrammi
  - Completa analogia con i programmi
- Si viene a creare una gerarchia di sottoprogrammi
  - Struttura risultante ad albero
    - Relazione padre-figlio riferita alla dichiarazione

Sottoprogrammi
Comunicazione

- Definizione
  - Titolo o intestazione
    - Identificatore
    - Specificazione delle risorse usate (talvolta)
  - Corpo
    - Sequenza di istruzioni denotata dal nome del sottoprogramma
- Comunicazione
  - Come si connettono i sottoprogrammi tra di loro?
  - Come si scambiano dati?
  - Come comunicano col programma principale?

Sottoprogrammi
Comunicazione

- Un sottoprogramma può comunicare
  - Con l'ambiente esterno
    - Istruzioni di lettura e/o scrittura
  - Con l'ambiente chiamante
    - Implicitamente
      - Tramite le variabili non locali (secondo le regole di visibilità del linguaggio)
    - Esplicitamente
      - Attraverso l'uso di parametri
        - » rappresentano le variabili che il sottoprogramma ha in input dal programma chiamante e che, opportunamente elaborate, vengono tramutate in output del sottoprogramma

Vista di un Sottoprogramma

- Rappresenta l'insieme delle risorse a cui il sottoprogramma ha accesso
  - Dati
  - Altri sottoprogrammi
- E' definita da
  - Nidificazione nella dichiarazione dei sottoprogrammi
    - *Vista Statica*
  - Sequenza di chiamata dei sottoprogrammi
    - *Vista Dinamica*
- Utile per limitare l'accesso alle risorse soltanto ai sottoprogrammi interessati

Vista di un Sottoprogramma
Sottoprogrammi

- Un sottoprogramma può richiamare soltanto i sottoprogrammi
  - che esso dichiara direttamente
  - che sono stati dichiarati dallo stesso sottoprogramma che lo dichiara
    - Incluso se stesso
      - Ricorsione
- Visibilità definita esclusivamente in base alla nidificazione
  - Figli e fratelli nella struttura ad albero

Vista di un Sottoprogramma
Variabili

- Ciascun sottoprogramma può usare esclusivamente
  - Le proprie variabili
  - Le variabili dichiarate dai sottoprogrammi attualmente in esecuzione
    - Visibilità dipendente
      - Dalla struttura della gerarchia di dichiarazione dei sottoprogrammi (statica)
      - Dall'ordine di chiamata dei sottoprogrammi precedenti (dinamico)

Vista di un Sottoprogramma
Shadowing (oscuramento)

- Sottoprogrammi diversi possono dichiarare risorse con lo stesso nome
  - Oggetti diversi, totalmente scorrelati
    - Possono essere di tipi differenti
- Sottoprogrammi attivi in un certo istante possono aver dichiarato risorse con lo stesso nome
  - Ciascun sottoprogramma attivo ha accesso solo al sinonimo “più vicino”
    - Visibilità dipendente esclusivamente dall’ordine di chiamata dei sottoprogrammi precedenti

Sottoprogrammi
Tipi di Variabili

- Variabili locali al sottoprogramma
  - Interne al sottoprogramma
  - Temporanee
    - Create quando il sottoprogramma entra in azione
    - Distrutte quando il sottoprogramma è stato eseguito
      - Liberazione del relativo spazio di memoria
- Variabili non locali al sottoprogramma
  - Definite nel resto del programma, al di fuori del sottoprogramma
    - Dette *globali* se definite nel programma principale

Sottoprogrammi
Tipi di Variabili

MAIN

Risorse globali

SOTTOPROGRAMMA P1

Risorse locali a P1 e non locali a P1.1

SOTTOPROGRAMMA P1.1

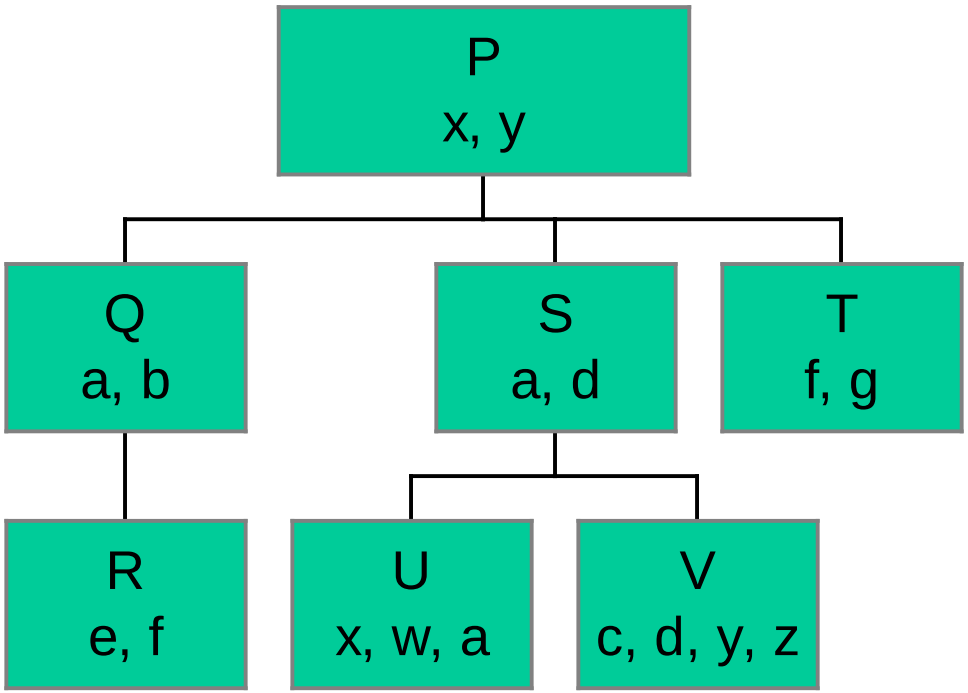
Risorse locali a P1.1

Sottoprogrammi
Regole di Visibilità

- Un identificatore è visibile nel programma o sottoprogramma in cui è dichiarato e in tutti i sottoprogrammi locali ad esso nei quali non è stato ridichiarato.
- Tutte le risorse di un programma o sottoprogramma devono essere dichiarate prima di essere usate
- Una *risorsa globale* è visibile ovvero accessibile ovvero usabile sempre e da tutti (main e sottoprogrammi) a meno che non venga oscurata (shadowing)
- Una *risorsa locale* ad un sottoprogramma P è visibile solo dalle istruzioni di P e dagli eventuali sottoprogrammi definiti in P

Vista di un Sottoprogramma
Esempio

Programma P



Vista di un Sottoprogramma

- Delimitazione spaziale di una risorsa
  - Le regole di visibilità degli identificatori stabiliscono l'*ambito* o *campo di visibilità* o *scopo degli identificatori* ovvero la zona di programma in cui è possibile fare riferimento a quell'identificatore.
- Delimitazione temporale di una risorsa
  - Le regole di visibilità definiscono anche la durata o il tempo di vita di una variabile ovvero l'intervallo di tempo in cui una variabile esiste (è allocata una area della RAM per essa)

Attributi delle variabili

- VARIABILE
  - NOME
  - VALORE
  - INDIRIZZO
  - TIPO
  - AMBITO
  - DURATA

Durata e ambito di una variabile

- In alcuni linguaggi il tempo di vita di una variabile coincide con il *tempo di esecuzione* del programma o del sottoprogramma in cui la variabile è dichiarata
- La fase in cui è definita un'area di memoria per una variabile è detta *fase di allocazione.*
  - L'allocazione può essere eseguita:
    - dal compilatore (allocazione statica) in base alla struttura lessicale del programma
    - in esecuzione (allocazione dinamica)

Effetti Collaterali

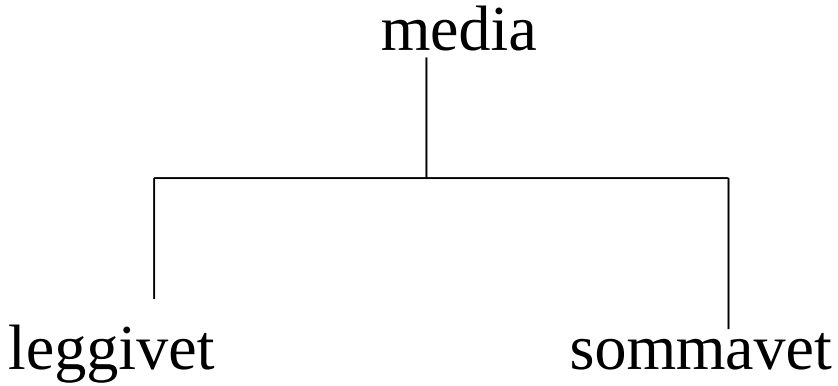
- Effetti di un sottoprogramma che altera il valore di una variabile non locale
  - La presenza di tali variabili impedisce che il sottoprogramma possa essere considerato come un'entità completa e autoconsistente
    - Non si riferisce esclusivamente alle sue costanti, variabili e parametri
- Attenzione: occorre valutare attentamente l'uso, all'interno di un sottoprogramma, di variabili non locali
    - Chiarezza
    - Sicurezza

Sottoprogramma
il countour model

```
PROGRAMMA media
    TIPO vettore=ARRAY(1,100) OF real
    vet:vettore
    n:integer
    m:real
    s:real
    SOTTOPROGRAMMA leggivet
            x:real
            i:integer
            BEGIN
                DO VARYING i FROM 1 TO n
                        leggi x
                        vet(i)=x
                REPEAT
            END
     SOTTOPROGRAMMA sommavet
            i:integer
            BEGIN
                s=0
                DO VARYING i FROM 1 TO n
                        s=s+vet(i)
                REPEAT
            END
    BEGIN
    leggi n
    leggivet
    sommavet
    m=s/n
    stampa m
    END
```

Sottoprogramma
il countour model

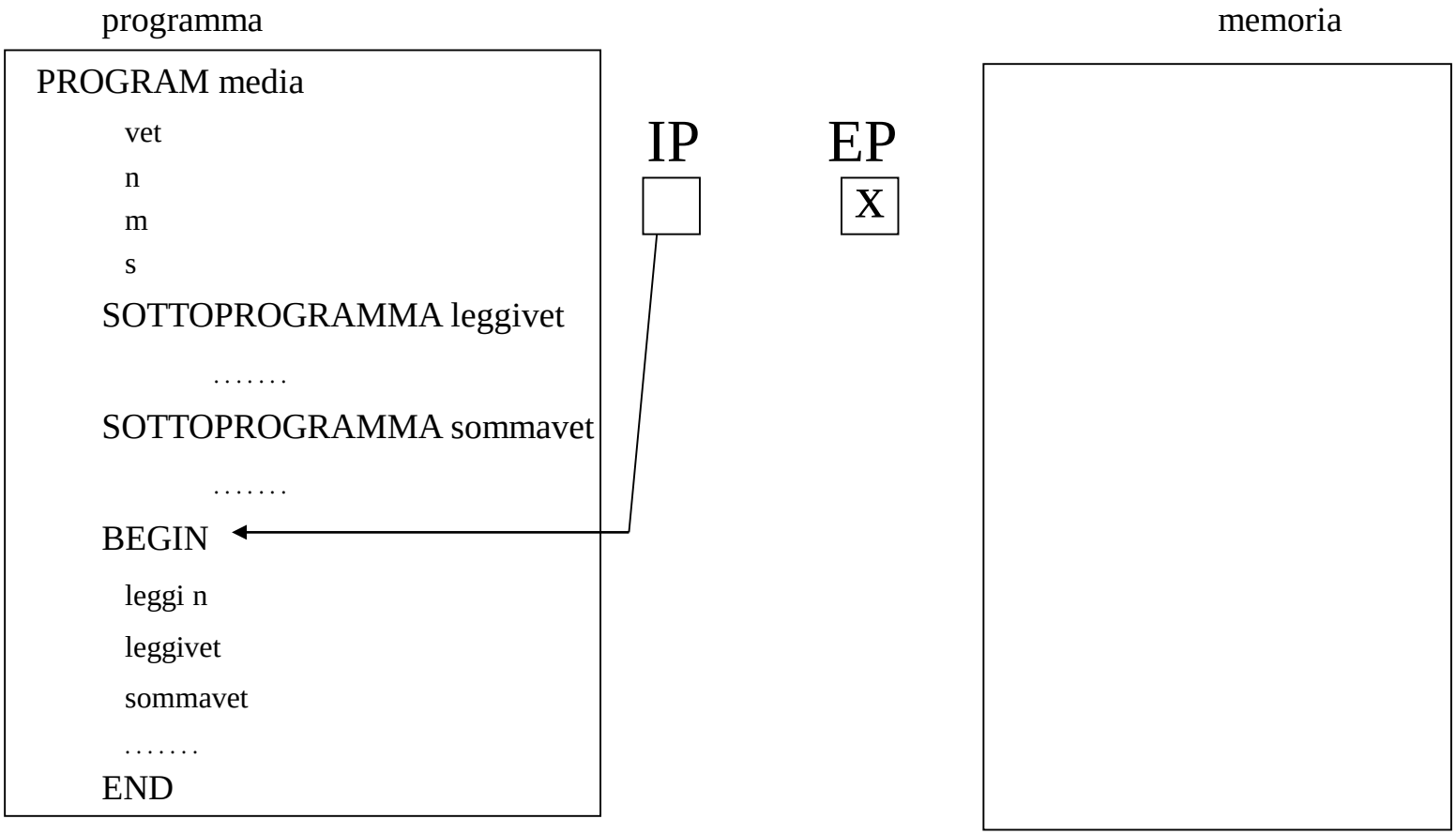
- La gerarchia di macchine è



  - media cede ai sottoprogrammi il diritto di accesso a tutte le sue variabili
  - ogni sottoprogramma ha le proprie variabili

Sottoprogramma
il countour model

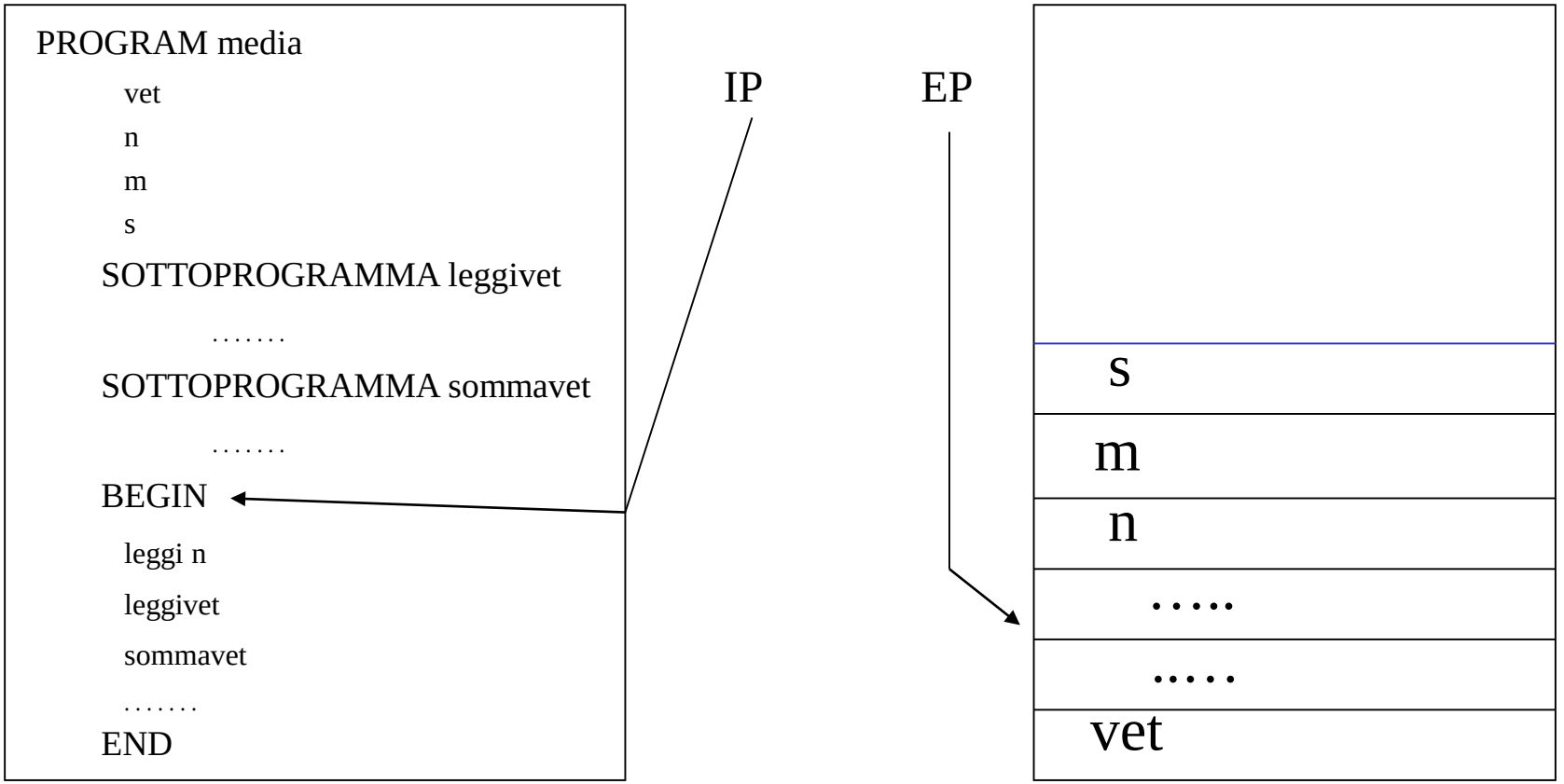
Il modello associato all'esecuzione del programma è detto *countour model*



IP instruction pointer (puntatore all'istruzione) EP environment pointer (zona della memoria per le variabili)

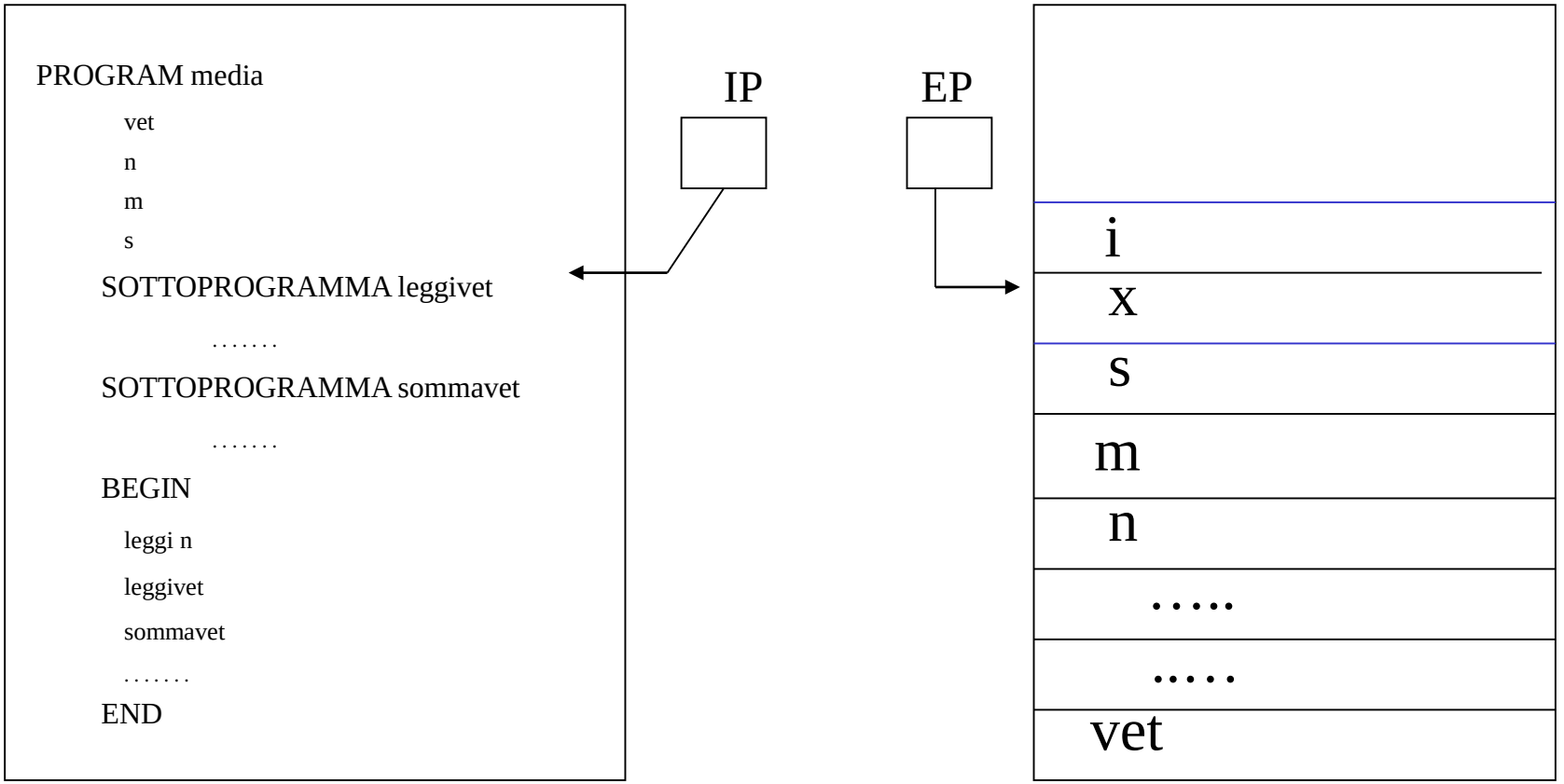
Sottoprogramma
il countour model

All’inizio dell’esecuzione vengono allocate le variabili del programma principale



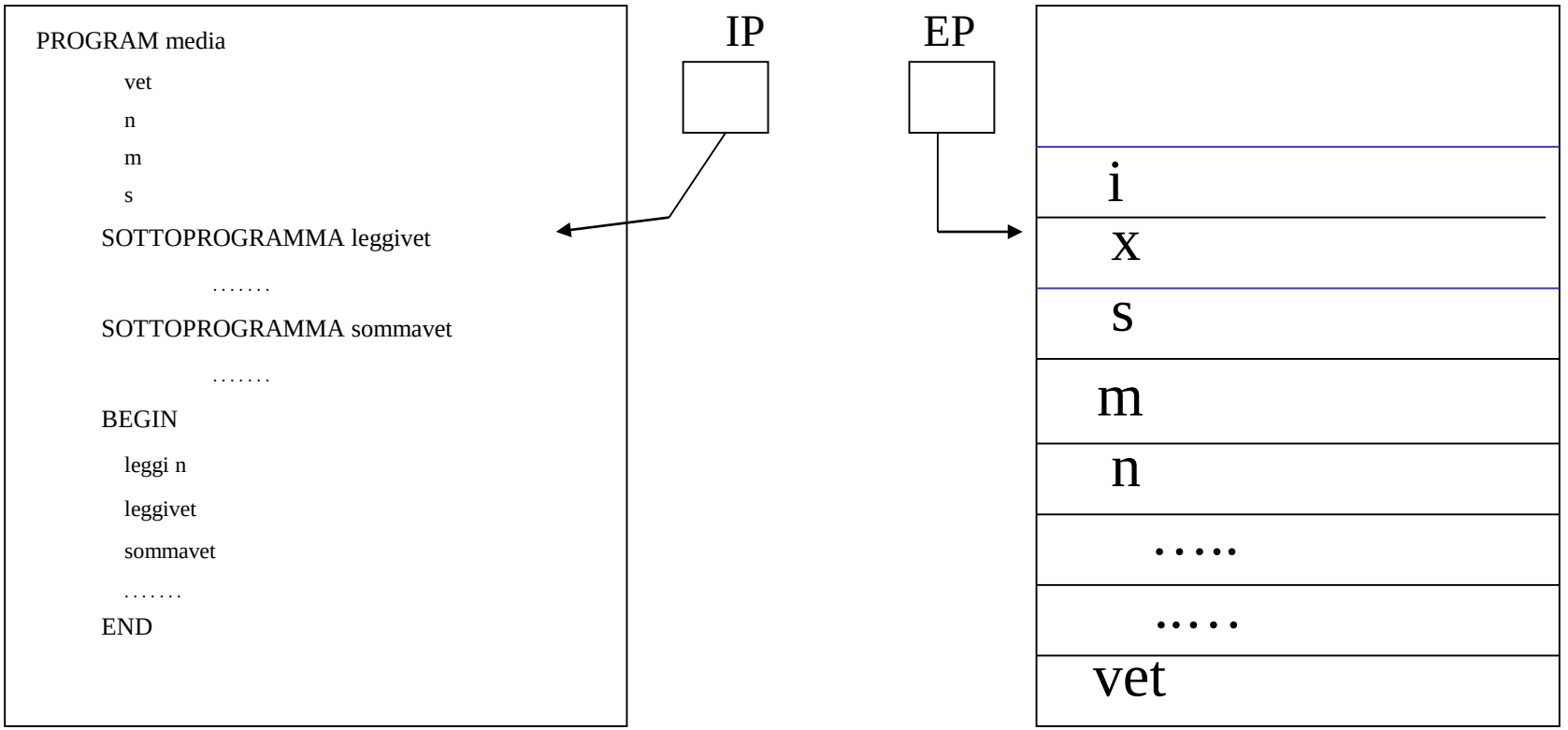
Sottoprogramma
il countour model

Quando viene richiamato il sottoprogramma leggivet vengono allocate nuove variabili in un nuovo ambiente a cui punterà EP



Sottoprogramma
il countour model

Leggivet userà le variabili i e x puntate da EP attuale mentre, non trovando n e vet nell'ambiente puntato da EP, risalirà negli ambienti precedenti sino a trovare la prima loro occorrenza. Questo è il meccanismo che consente ad un sottoprogramma di ereditare il diritto di accesso al programma che lo richiama



Sottoprogramma
il countour model

Terminata l’esecuzione di leggivet, l’ambiente puntato da EP non serve più quindi viene rilasciata l’area di memoria occupata e EP torna a puntare all’ambiente che aveva richiamato leggivet.

Sottoprogramma esempio

```
sottoprogramma quadrato
    i: integer
    quad: integer
    begin
        do varying i from 1 to 10
            quad ← i*i
            stampa quad
        repeat
    end
```

dichiarazione variabili

Per stampare i primi 15 quadrati?
Per stampare i primi 35 quadrati?

Sottoprogramma esempio

```
sottoprogramma quadrato (n: integer)
    i: integer
    quad: integer
    begin
        do varying i from 1 to n
            quad        i*i
            stampa quad
        repeat
    end
```

dichiarazione variabili

```
quadrato(10)
quadrato(m)
```

chiamate

Sottoprogramma

Nella dichiarazione del sottoprogramma:

- l'*intestazione* introduce il *nome* e gli *argomenti* del sottoprogramma
- Il *corpo* descrive le regole di comportamento del sottoprogramma

Nell'intestazione del sottoprogramma appaiono i *parametri formali* che all'atto della chiamata vengono sostituiti dai *parametri effettivi* (*attuali*) ovvero dai dati su cui il sottoprogramma deve operare

Parametri Formali

- Segnaposto per indicare simbolicamente
  - Gli oggetti su cui il sottoprogramma lavora
  - Il loro tipo e struttura
- I loro nomi appaiono nell'intestazione del sottoprogramma
  - Non hanno alcuna connessione con nomi usati altrove
- All'atto della chiamata vengono sostituiti dai parametri *effettivi* (o *reali*)
  - Dati su cui effettivamente il sottoprogramma deve operare

Parametri Formali

- Specificati all'atto della definizione del sottoprogramma
  - Legati al sottoprogramma
  - Simbolici
  - Consentono di definire
    - Quale tipo di dato deve essere passato alla procedura
    - Quale argomento deve essere trasmesso alla funzione

  quando queste sono invocate

Parametri Effettivi

- Alla chiamata di un sottoprogramma, vanno specificati i dati effettivi su cui esso dovrà operare
  - Valori, Espressioni, Variabili, …
  - Coincidenza con i parametri formali
    - Numero, tipo e ordine
- L'esecuzione dell'istruzione di chiamata comporta la sostituzione dei parametri formali con quelli reali

Parametri
Tipi di Passaggio

- La sostituzione può essere:
  - Per valore
    - Si calcola il valore del parametro reale e lo si sostituisce al corrispondente parametro formale (assegnazione)
  - Per referenza
    - Il parametro effettivo è una variabile ed ha a disposizione una locazione di memoria il cui indirizzo viene “passato” al parametro formale
  - Per nome (o *valore-risultato*)
    - Il nome del parametro formale, all’occorrenza, viene sostituito col nome del parametro reale

Parametri
Tipi di Passaggio

- *P*(*pf*) attivata con parametro reale *pe*
  - Per valore:
    - Il parametro formale si comporta come una variabile locale a *P*
  - Per riferimento:
    - Al momento dell'attivazione di *P* viene calcolato l'indirizzo di *pe* e *pf* viene creato con riferimento alla stessa locazione di memoria
  - Per nome:
    - Al momento dell'attivazione di *P* il valore di *pe* viene calcolato e memorizzato in una nuova locazione di indirizzo *pf*
    - Al termine dell'esecuzione di *P* il contenuto di *pf* viene trasferito in *pe* e la memoria riservata per *pf* viene rilasciata

Passaggio di Parametri
Esempio

```
program legaparametri (…,…);
    var n: integer;
    sottoprogramma P (? x : integer)
      begin
      x := x + 1;
      writeln(n);    {1}
      writeln(x)     {2}
      end;
    begin
    n := 3;
    P(n);
    writeln(n)       {3}
    end.
```

- Per valore
  1. 3
  2. 4
  3. 3
- Per referenza
  1. 4
  2. 4
  3. 4
- Per nome
  1. 3
  2. 4
  3. 4

Passaggio di Parametri
Pro (+) e Contro (-)

- Copia di valori
  - \+ permette la trasmissione del valore di un parametro dal chiamante
  - \+ permette la separazione tra programma chiamante e programma chiamato
  - – aumenta l'occupazione di memoria ed il tempo
  - – rende difficile la gestione di parametri di dimensione variabile
- Trasmissione per riferimento
  - \+ evita problemi di passaggio perché il trattamento degli indirizzi è gestito direttamente dal compilatore
  - \+ non occupa memoria aggiuntiva
  - – causa effetti collaterali spesso imprevedibili

Passaggio di Parametri

per Valore

- Generalmente usato per parametri che
  - Rappresentano un argomento e non il risultato di un sottoprogramma
    - Inutile consentirne la modifica

Passaggio di Parametri
per Referenza

- Usato più spesso quando il parametro
  - Rappresenta un risultato
    - Necessità di conoscere la modifica
  - Ha dimensioni notevoli
    - Pesante ricopiarlo interamente
- Potenziale fonte di errori
  - Stessa variabile usata sotto diverse denominazioni
  - Errori nel sottoprogramma irrecuperabili

Procedure

- Sottoprogrammi il cui compito è quello di produrre un effetto
  - Modifica del valore di variabili
  - Comunicazione di informazioni all'utente
- L'intestazione di procedura (e la sua chiamata) includono
  - Nome della procedura
    - Paragonabile ad una nuova istruzione del linguaggio
  - Lista di parametri

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

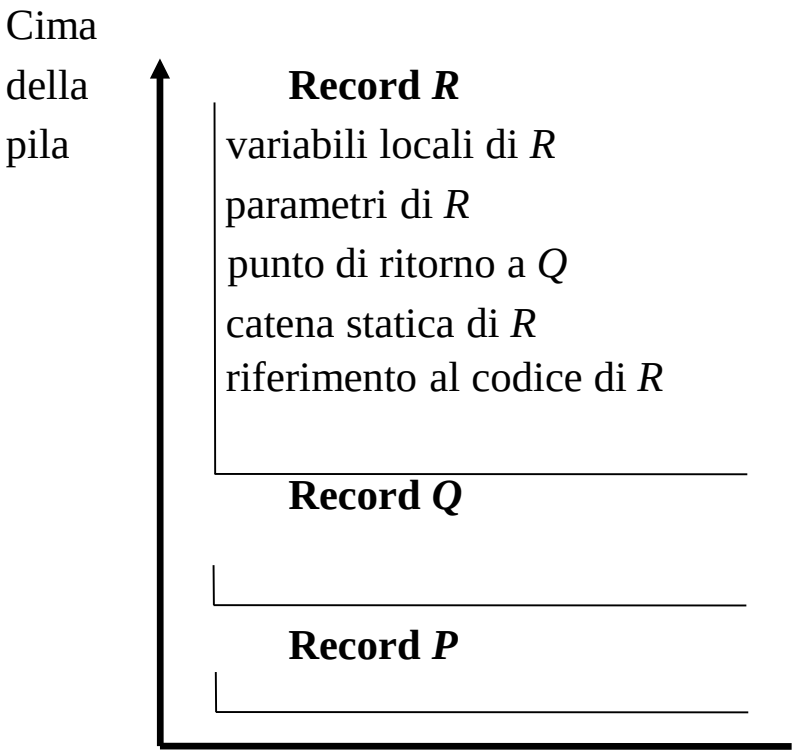
- Nome dell'unità di programma
- Riferimento all'area di memoria in cui è memorizzato il corpo di istruzioni da eseguire
- Punto di ritorno
  - Riferimento all'istruzione a cui tornare al termine dell'attivazione
- Gerarchia creata al momento della dichiarazione (*catena statica*)
- Parametri formali 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

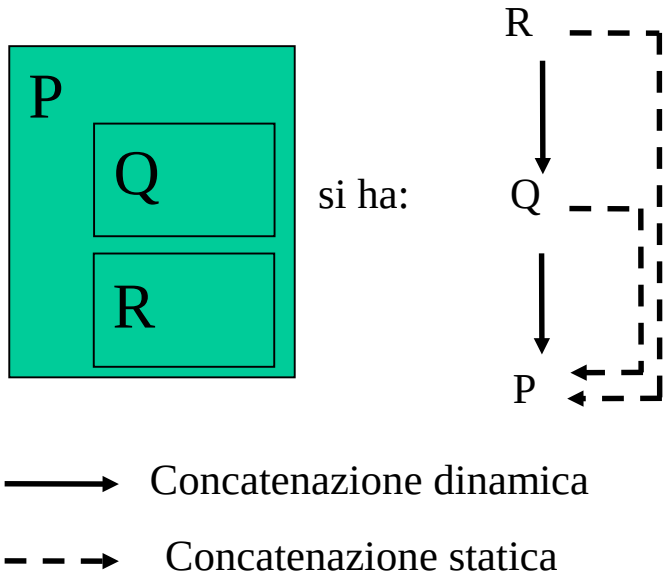
- Statica (può essere determinata guardando il testo del programma)
  - Definita dalla nidificazione nella dichiarazione delle procedure
    - non può variare
    - fornisce i riferimenti nella pila alle variabili non locali
- Dinamica (dipende dall'esecuzione)
  - Realizzata attraverso la sequenzializzazione 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
    nfatt(n:integer):integer
    BEGIN
    if n=0 then nfatt←1
     else nfatt←  n*nfatt(n-1)
    END
    BEGIN (del main)
    leggi n
    fatt← nfatt(n)
    scrivi fatt
    END
```

p=6
p=5
p=4
p=3
p=2
p=1
p=passo