

Procedura → Astrazione di parti di programma in vista di esecuzione più piccole, nascondendo i dettagli sul loro uso integrazione

Vantaggi :

- Programmi facili da manipolare e sviluppare top-down;
- Unità di programma indipendenti o con dipendenza ad alto livello;
- Riusabilità;

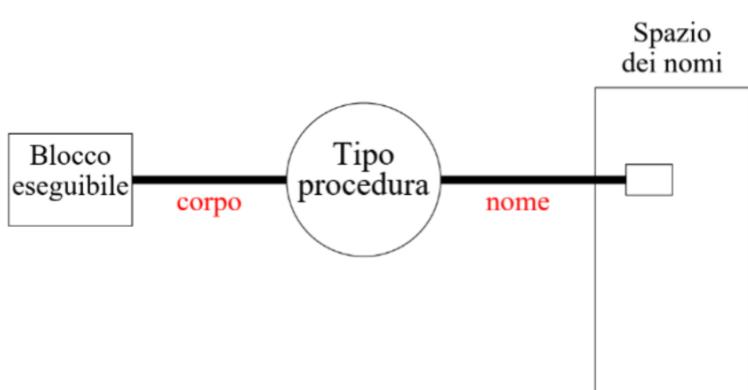
Astrazione procedurale :

Con una suddivisione in unità di esecuzione con completezza, espressione, statement, blocchi e programmi diversi, allora, l'astrazione procedurale è la rappresentazione di un'unità di esecuzione usando un'unità più semplice (integrazione).

Rapp. di un blocco molto un'espressione

Dichiarazione di procedura → Creazione legame di nome con il corpo della procedura.

Questo legame prende il nome di Tipo procedura
↳ stabilito durante la compilazione



Invocazione di una procedura → Generazione legame tra **tip procedura** e un RDA.

Questo legame prende il nome di **Oggetto procedura**

↳ nasce durante l'esecuzione, quando la procedura viene eseguita.

Ogni volta invocazione della stessa procedura genera un nuovo **Oggetto procedura**, con medesimo legame di tipo, ma diverso **RDA**



Ambiente d'esecuzione → come avvenire per un blocco in-line, il RDA è l'intero ambiente di esecuzione di una procedura, con delle differenze:

- ha un nome per l'invocazione;
- Può avere parametri;
- Può restituire un valore;

l'RDA di una funzione contiene:

- ① Puntatore di catena dinamica (link al precedente RDA);
- ② Puntatore di catena statica (new!) (in caso di **scoping statico**);
- ③ Indirizzo di ritorno (new!)
- ④ Indirizzo del risultato (new!)
- ⑤ Parametri (new!)
- ⑥ Ambiente locale

L'ambiente **non** locale di una funzione viene recuperato con un puntatore di catena dinamica (**scoping dinamico**) o statico (**scoping statico**)

Propagazione dei data object

Ambiente non locale \rightarrow Insieme di nomi esterni di una funzione, ma a cui può riferirsi (È la propensione a dipendere dal linguaggio)

3 tipi di propagazione (**Scoping**):

- 1 Statica, l'ambiente non locale viene propagato programma/procedura **intestatamente** (**posizionale**);
- 2 Dinamica, l'ambiente viene propagato dal programma/procedura chiamante;
- 3 Nessuna propagazione (o l'uso può causare danni)

Implementazione \rightarrow Lo scoping viene realizzato inserendo nell'RDA un puntatore delle

funzione che ammette la propagazione dei dati object

Scoping statico → uso puntatore di catena statica

Scoping dinamico → uso puntatore di catena dinamica

Se si desidera un dato non locale, esso viene ricercato seguendo la catena

Nel caso di blocchi in-line, l'RSA non ha il puntatore di catena statica perché viene eseguito sempre nel contesto del blocco.

Un for dentro un for verrà sempre eseguito

Prendendo come esempio C :

- ha Scoping statico;
- non supporta funzioni annidate;
- Solo blocchi in-line annidati;
- Non ha il puntatore di catena statica;
- Lo scope non locale contiene → blocchi esterni a quelli correnti, poi le funzioni correnti e lo scope globale

Pseudo-codice:

```
program p;
  var a, b, c: integer;

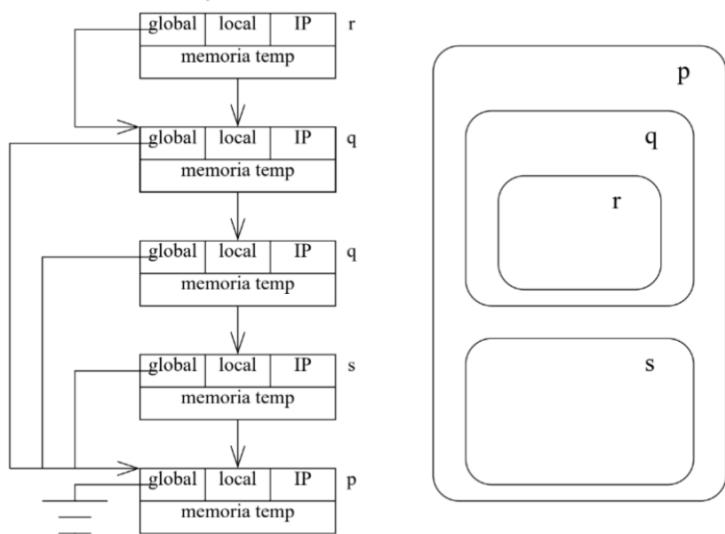
  procedure q;
    var a, c: integer;
  procedure r;
    var a: integer;
    begin r           {variabili: a da r; b da p; c da q;
      ...             procedure: q ed s da p; r da q}
      ...
    end r;
    begin q           {variabili: a da q; b da p; c da q;
      ...             procedure: q ed s da p; r da q}
      ...
    end q;

  procedure s;
    var b: integer;
    begin s           {variabili: a da p; b da s; c da p;
      ...             procedure: q ed s da p}
      ...
    end s;

begin p            {variabili: a, b, c da p;
  ...             procedure: q, s da p}
end p.
```

Scoping statico (2)

Supponendo una sequenza di attivazione (p, s, q, q, r) , lo stack di esecuzione ha questa forma:



Osservazioni sullo scoping dinamico

- Il puntatore di cattedo statico non è necessario ;
- Basta quelli di cattedo dinamico ;

Vantaggio : propagazione di informazioni soltanto tra procedure ;

Svantaggio : impossibile determinare l'ambiente di esecuzione di una procedura

```
program p;
  var a: integer;
procedure q;
  begin q           {vars: a da p o da r; procs: q, r da p}
    ...
  end q;
procedure r;
  var a: integer;
  begin r          {vars: a da r; procs: q, r da p}
    ...
  end r;
begin p            {vars: a da p; procs: q, r da p}
  ...
end p.
```

può essere anche "a" di "p" oppure di "r"

