Capitolo IV

La struttura dei Programmi

Frasi del linguaggio

- Tutti i linguaggi di alto livello prevedono quattro tipologie di frasi diverse:
  - le **istruzioni** che tradotte in linguaggio macchina indicano al processore le operazioni da svolgere;
  - le **strutture di controllo** che definiscono l'ordine di esecuzione delle istruzioni;
  - le frasi di **commento** che permettono l'introduzione di frasi in linguaggio naturale utili a rendere più comprensibili i programmi ad un lettore umano
  - le **dichiarazioni** con le quali il programmatore da ordini al traduttore del linguaggio di programmazione;

Variabili e Tipi

- Variabili
  - Ad una variabile corrisponde una porzione di memoria la cui dimensione e le cui regole di uso dipendono dal suo tipo.
- Per tipo di dato si intende un insieme di valori associati a delle operazioni definite su di essi
  - Consente di introdurre dei modelli degli oggetti della realtà in quanto ne fornisce le caratteristiche *strutturali* (cosa sono) *e funzionali* (a cosa servono).
  - È possibile trattare l'informazione in maniera astratta, cioè prescindendo dal modo concreto con cui essa è rappresentata all'interno di una macchina.
  - In molti linguaggi esistono dichiarazioni per la definizione di tipo: **type t = T;** dove t è il nome del nuovo tipo e T è un descrittore di tipo, cioè un costrutto sintattico del linguaggio atto a definire il tipo t.

  Il descrittore T può essere anche un tipo già definito o predefinito nel linguaggio, in modo da introdurre dei tipi sinonimi.
- L'istanziazione di variabile avviene poi semplicemente con frasi dichiarative come quella seguente: **var v : T;**

Commenti

- Le frasi di commento sono frasi in linguaggio naturale prive di ogni valore esecutivo o dichiarativo che consentono di migliorare la leggibilità e la chiarezza del programma.
  - **Asserzioni:** sono commenti destinati a fissare in un punto del programma lo stato di una o più variabili.
    - Per tale motivo permettono di chiarire le condizioni nelle quali vengono ad operare le istruzioni successive. Le asserzioni sono molto importanti per controllare la correttezza dei programmi.
  - **Motivazioni:** sono commenti destinati a chiarire le ragioni e/o gli obiettivi per i quali il blocco di programma, successivo al commento, viene scritto. Le motivazioni sono essenziali perché un programma possa essere compreso da altre persone.

I commenti nei linguaggi

- Per indicare le frasi di commento si usano delle sequenze di caratteri (marcatori) che indicano l'inizio e la fine
  - ad esempio /* */ in C
  - In alcuni casi il commento occupa un solo rigo ed allora lo si indica con un solo marcatore iniziale (ad esempio // del C). Le due modalità possono coesistere nello stesso linguaggio.
- I linguaggi non forniscono strumenti per specificare il tipo di commento: per farlo il programmatore potrà anteporre alla frase la notazione "A:" per le asserzioni e "M:" per le motivazioni come gli esempi seguenti illustrano.
- (*A: alfa deve avere valore negativo *)
- /*A: i>=10 implica condizione di errore */
- x= totale / numero_elementi //M: Calcolo della media

Istruzioni di Assegnazione

- Con le istruzioni elementari di calcolo ed assegnazione si prescrive il calcolo di una qualche espressione con memorizzazione del risultato.
- In tutti i linguaggi di programmazione essa è presente in una forma del tipo:

  v ← *espressione;*

  che indica che l'esecutore deve prima risolvere l'espressione presente nel secondo membro e, solo quando ne ha prodotto il risultato, assegnare quest'ultimo alla variabile v posta a primo membro.

- L'istruzione viene anche scritta in una delle forme seguenti:

  v := *espressione; (Pascal)*

  v = *espressione; (C)*

Costrutti di controllo

- I costrutti di controllo indicano all'esecutore l'ordine in cui le operazioni devono essere eseguite.
  - Essi devono essere scelti in modo da rispecchiare quanto più possibile i meccanismi che naturalmente vengono usati quando si descrive (ad esempio in italiano) una qualsiasi successione di operazioni.
- Costrutti di sequenza, selezione ed iterazione

Sequenza

- Specifica che due istruzioni devono essere eseguite l'una dopo l'altra

```
PRIMA     stacca il contatore
POI       sostituisci la presa
QUINDI    riattacca il contatore
```

```
BEGIN                          {
   stacca il contatore;           stacca il contatore;
   sostituisci la presa;          sostituisci la presa;
   riattacca il contatore         riattacca il contatore
END                            }
```

Costrutti Condizionali (1)

- consentono di subordinare l'esecuzione di una certa operazione al verificarsi o meno di una specificata condizione

SE (la carta scoperta è quadri)
ALLORA vinci
ALTRIMENTI perdi

IF (la carta scoperta è quadri)
then vinci
else perdi

Costrutti Condizionali (2)

- Selezione tra più alternative

NEL CASO (in cui il colore del semaforo) **è** ROSSO: fermati **è** VERDE: passa l'incrocio **è** GIALLO: passa con prudenza o fermati	**CASE** (colore del semaforo) **OF** ROSSO: fermati; VERDE: passa l'incrocio; GIALLO: passa con prudenza o fermati. **END**

Costrutti Iterativi

- prescrivono di ripetere l'esecuzione di una o più operazioni; tale ripetizione viene sospesa al verificarsi di un evento

RIPETI i compiti **FINCHE'** (suona la sveglia)	**REPEAT** i compiti **UNTIL** (suona sveglia)
MENTRE (piove) **DEVI** usare l'ombrello	**WHILE** (piove) **DO** usa l'ombrello
PER (10 giorni) **DEVI** non venire all'università	**FOR** giorni:=1 **TO** 10 **DO** non venire all'università

Equivalenza ed Equivalenza Funzionale

- Si dicono **equivalenti** due programmi che evocano le stesse sequenze di esecuzione.
- Sono invece **funzionalmente equivalenti** due programmi che, sollecitati nello stesso modo, producono lo stesso risultato.
  - Si noti che due programmi equivalenti sono anche funzionalmente equivalenti, mentre non è vero, in generale, che due programmi funzionalmente equivalenti sono anche equivalenti
- L'equivalenza funzionale è utile per comprendere le differenze tra le varie strutture di controllo e per giustificare da un punto di vista pratico la necessità di costrutti simili.

Esempio di Equivalenza Funzionale

assegna valore ad x e y; **if** (x≤0) **then** stampa il valore di x+y **else** stampa il valore di x*y;	assegna valore ad x e y; **if** (y>0) **then** stampa il valore di x*y **else** stampa il valore di x/y;

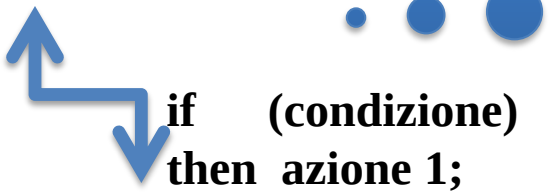
- I due programmi producono lo stesso risultato
  - il prodotto di x per y

  se le due variabili sono positive, e risultati diversi negli altri casi
  - la somma e la divisione se sono entrambe negative, la somma o il prodotto e il prodotto e il rapporto se il loro segno è discorde.

Costrutti di selezione Funzionalmente Equivalenti

- è sempre possibile ricondurre un costrutto **if-then-else** ad una sequenza di soli **if-then**

```
if      (condizione)
then  azione 1
else   azione 2;
```



```
if      (condizione)
then  azione 1;
if      (not condizione)
then   azione 2
```

Sono anche equivalenti? Perché no?

Strutture funzionalmente equivalenti al CASE

- La struttura case, che prescrive la valutazione di una espressione (anche detta selettore) e la scelta dell'azione a cui è stato associato il valore ottenuto da tale valutazione, può essere ricondotta ad un insieme di if disposti l'uno dentro l'altro

```
case (espressione) of
        a: azione 1;
        b: azione 2;
        c: azione 3
end;
```

```
if (espressione=a)
then azione 1
else
    if (espressione=b)
        then azione 2
        else
          if (espressione=c)
          then azione 3;
```

CASE (2)

- non vale il passaggio inverso: ossia non è detto che una struttura di if-then-else nidificati (inseriti uno dentro l'altro) è esprimibile con un case

```
if (espressione 1)
then azione 1
else
        if (espressione 2)
                then azione 2
                else
                  if (espressione 3)
                  then azione 3;
```

While e Repeat

- E’ sempre possibile ricondurre l'una all'altra.
  - While:
    - prima viene valutata la condizione e dopo c’è l'esecuzione delle azioni qualora il valore ottenuto sia vero. Il ciclo termina quando la condizione diventa falsa.
    - il ciclo non avviene se la condizione è falsa la prima volta che è calcolata.
  - Repeat
    - Prima c’è l'esecuzione delle azioni e, dopo, la valutazione della condizione
    - Le azioni vengono eseguite almeno una volta.
    - La ripetizione delle azioni termina quando la condizione diventa vera.
- Si noti che, in entrambi casi, quando il ciclo ha avuto inizio, si deve far in modo che le azioni eseguite ad ogni ripetizione alterino le variabili presenti nella condizione se si vuole che il ciclo termini.

While e repeat (2)

repeat azione **until** (condizione);	azione; **while** (**not** condizione) **do** azione;

while (condizione) **do** azione;	**repeat** **if** (condizione) **then** azione **until** (**not** condizione);

For

- la struttura ciclica è riconducibile ad una iterativa, e di conseguenza anche all'altra.
  - un ciclo iterativo prescrive la ripetizione di azioni un numero di volte fissato a priori e determinato dal fatto che una variabile detta contatore di ciclo, a partire da un valore iniziale raggiunge un valore finale o per valori crescenti (indicato dal ***to***) o decrescenti (indicato dal ***downto***).

for i:=vp **to** vg **do** azione;	i:= vp; **while** i≤vg **do begin** azione; i:= i +1 **end**;

for i:=vg **downto** vp
do azione;
	i:= vg;
	while i≥vp
	do begin
	azione;
	i:=i-1
	end;

Cicli a conteggio vs iterativi

- i costrutti ciclici sono:
  - più **espressivi**, perché mostrano su una sola linea tutti i parametri che governano l'iterazione;
  - più **sintetici**, perché risparmiano la scrittura delle frasi di inizializzazione, controllo e modifica della variabile;
  - più **sicuri**, in quanto garantiscono che non si verifichino cicli senza fine (dovuti ad errori di gestione della variabile contatore di ciclo).
- In generale i costrutti ciclici consentono anche di specificare il passo nel caso sia diverso da 1 :

for i:=vp **to** vg **STEP** v
do azione;

for i:=vg **downto** vp **STEP** v
do azione;

Il Teorema di Böhm-Jacopini

- Böhm e Jacopini nel 1966, hanno dimostrato che **tutti gli algoritmi possono essere espressi utilizzando**:
  - **una sola struttura di sequenza**
  - **una sola struttura selettiva**
  - **un sola struttura iterativa**.

Il linguaggio “a salti”

- alcuni costrutti diventano non necessari se si introduce nel linguaggio un meccanismo di riferimento delle istruzioni.
- In alcuni linguaggi è difatti possibile etichettare con un **numero o un nome le istruzioni**.
  - L'uso di tali etichette permette di prescrivere che l'istruzione successiva da eseguire non sia quella scritta nel rigo seguente ma quella identificata dalla etichetta specificata.

100 azione 1;
azione 2;
if (condizione)
then goto ***100***;
azione 3;

Il “goto”

IF (condizione) **then** azione 1 **else** azione 2; azione 3;	**if** (condizione) **then begin** azione 1; **goto** 100 **end**; azione 2; 100 azione 3;
while (condizione) **do** azione**;**	100 **if** (condizione) **then begin** azione; **goto** 100 **end**;
repeat azione **until (**condizione);	100 azione; **if** (**not** condizione) **then goto** 100;
case (espressione) **of** a : azione 1; b : azione 2; c : azione 3 **end;** azione 4;	if (espressione=a) **then begin** azione 1; **goto** 100 **end**; **if** (espressione=b) **then begin** azione 2; **goto** 100 **end**; **if** (espressione=c) **then begin** azione 3; **goto** 100 **end**; 100 azione 4;

L’articolo di Dijkstra (1968)

- Per lunghi anni il goto è stato largamente usato dai programmatori.
- Dijkstra:
  - “la capacità dei programmatori è inversamente proporzionale all'uso del goto”
  - propose con molto scalpore l’abolizione del goto dai linguaggi di programmazione.
    - l'uso indiscriminato del goto porta a programmi con molti intrecci, senza un filo logico, e quindi poco chiari.
      - si possono creare blocchi che hanno più ingressi e più uscite
      - è possibile entrare nel mezzo di funzionalità ben evidenziate nel testo, inficiando la costruzione logica di un programma.

Esempio

```
100  azione 1;
     azione 2;
200  azione 3;
     if (condizione 1)
     then goto 200
     else goto 400;
300  azione 4;
     for i:=e1 to es
     do begin
          azione 5;
          if (condizione 2)
          then goto 300;
          azione 6
        end;
```

si nota come ci siano tre punti di ingresso (istruzioni etichettate con 100, 200 e 300), due punti di uscita (goto 400 e la fine della sequenza), l'interruzione del for (goto 300) nonostante tale costrutto debba essere usato quando il numero di volte sia fissato a priori.

Il goto non va mai usato?

- In alcuni casi non si può fare a meno di usare il goto perché:
  - non sono presenti altri costrutti di controllo
  - a volte risulta necessario l'abbandono completo di porzioni di programma
    - per esempio per segnalare subito errori catastrofici
- In questi casi l'uso dell'istruzione di salto deve essere valutato con molta attenzione e segnalato con appositi commenti.

Il Nesting

- le strutture di controllo possono essere disposte
  - in sequenza tra di loro
  - inserite le une dentro le altre con una modalità detta di innestamento o nidificazione.
- Se sono disposte l'una dopo l'altra, allora dopo aver espletato le azioni indicate da una struttura di controllo, si procede prendendo in considerazione quella ad essa successiva nella sequenza statica

I_0;
case (espressione) of;
1 : **if (condizione1)**
then begin
I_1;
I_2
end
else I_3;
5 : **for i:=1 to 3**
do I_4;
8 : I_5;
end;
if (condizione2)
then I_6;
I_f;

… sequenze possibili

s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
I_0	I_0	I_0	I_0	I_0	I_0	I_0	I_0	I_0	I_0
I_1	I_3	I_4	I_5	I_6	I_1	I_3	I_4	I_5	I_f
I_2	I_6	I_4	I_6	I_f	I_2	I_f	I_4	I_f	
I_6	I_f	I_4	I_f		I_f		I_4		
I_f		I_6					I_f		
		I_f							

```
I0;
case (espressione) of;
    1 :  if (condizione1)
           then begin
                  I1;
                  I2
                end
          else I3;
    5 : for i:=1 to 3
        do I4;
    8 :  I5;
end;
if (condizione2)
then I6;
If;
```

One in one out

- Poiché le istruzioni semplici
  - l'assegnazione di valore, le procedure di input e di output e le operazioni definite sulle strutture dati

  presentano un solo punto di ingresso e di uscita, la loro composizione attraverso le strutture di controllo
  - IF-THEN-ELSE
  - IF-THEN
  - CASE
  - REPEAT-UNTIL
  - WHILE-DO
  - FOR-DO

  porta ad **una istruzione composta che mostra ancora un unico punto di ingresso e di uscita**.
- Tale caratteristica viene anche indicata dicendo che la sequenza è di tipo ***one-in*** **e** ***one-out***.
  - L'uso di istruzioni di salto impedisce la costruzione di sequenze di tipo one-in e one-out.

Quante sequenze di esecuzione?

- Il numero di sequenze di esecuzione evocate da una istruzione composta dipende dal **tipo di strutture** di controllo utilizzate e dal **modo** in cui queste sono composte tra loro.
- Se le strutture sono considerate singolarmente, allora:
  - entrambe le strutture di controllo IF-THEN-ELSE e IF-THEN generano **due** sequenze di esecuzione
  - il CASE tante sequenze di esecuzione **quanti sono i valori previsti per il selettore**
  - non è possibile fare previsioni per le strutture iterative in quanto il numero di sequenze di esecuzione dipende dai valori che assumono le variabili di controllo dei cicli durante l'esecuzione

Esempi - sequenza

	s_1	s_2	s_3	s_4
if c_a then I_1	I_1	I_1	I_2	I_2
else I_2;	I_3	I_4	I_3	I_4
if c_b then I_3	I_f	I_f	I_f	I_f
else I_4;				
I_f;				

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
if c_a then I_1	I_1	I_1	I_1	I_1	I_2	I_2	I_2	I_2
else I_2;	I_3	I_3	I_4	I_4	I_3	I_3	I_4	I_4
if c_b then I_3	I_5	I_6	I_5	I_6	I_5	I_6	I_5	I_6
else I_4;	I_f	I_f	I_f	I_f	I_f	I_f	I_f	I_f
if c_d then I_5								
else I_6;								
I_f;								

una sequenza di n strutture selettive evoca 2^n sequenze di esecuzione.

Esempi: innesto

	s_1	s_2	s_3

if c_a then
  if c_b then I_1
  else I_2;
 else I_3
I_f;
	I_1
	I_f
		I_2
		I_f
			I_3
			I_f

	s_1	s_2	s_3	s_4

if c_a then
  if c_b then
  if c_d then I_1
  else I_2;
  else I_3
 else I_4;
I_f;
	I_1
	I_f
		I_2
		I_f
			I_3
			I_f
				I_4
				I_f

- un tale innesto di n strutture selettive evoca n+1 sequenze di esecuzione.

Esempi: iterazione

	s_1	s_2	s_3
I_0; for i:=ei to es do I_1 I_f;	(ei=1 es=3) I_0 I_1 I_1 I_1 I_f	(ei=8 es=0) I_0 I_f	(ei=50 es=50) I_0 I_1 I_f

- nel caso di strutture iterative, il numero di sequenze di esecuzione varia in funzione dei valori assunti, durante l'esecuzione, dalle variabili che governano le condizioni della iterazione.

Modularità

- La sola applicazione della tecnica di progettazione top-down non permette di:
  - *evitare riscritture inutili* di sottoproblemi dello stesso tipo in più punti del programma;
  - *integrare soluzioni già disponibili* di alcuni sottoproblemi;
  - non perdere la *sinteticità d*elle frasi in linguaggio naturale introdotte per descrivere i vari sottoproblemi;
  - *verificare la correttezza della soluzione* del problema per passi
    - ossia verificando dapprima la correttezza dei singoli sottoproblemi e successivamente dell'insieme da essi costituito.

l sottoprogrammi

- i linguaggi di programmazione prevedono l'uso dei sottoprogrammi:
  - **assegnare ad una sequenza di istruzioni un nome** che può essere utilizzato come sua abbreviazione e che può essere inserito al suo posto nel programma.
    - se il nome rappresenta anche un risultato, che può essere inserito in una espressione o in una istruzione, il sottoprogramma viene chiamato ***funzione***: in tutti gli altri casi viene anche chiamato ***procedura***.
- L'indicazione del nome di un sottoprogramma viene chiamata "indicazione della procedura" (o della funzione) corrispondente;
- l'impiego di una procedura in un programma viene detto "chiamata della procedura"
  - quello di una funzione, viene detto "chiamata della funzione”.

Struttura di un sottoprogramma

- L'indicazione del sottoprogramma si compone di due parti: il titolo e il corpo.
  - Nel titolo della procedura vengono indicati il suo identificatore (il nome) e altre informazioni;
  - il corpo si compone della sequenza di dichiarazioni e istruzioni denotata dal nome.

sottoprogramma prepara_la_crema;
{
SP_1: sbatti le uova con lo zucchero;
SP_2: stempera la farina nel latte;
SP_3: versa il latte nelle uova e nello zucchero;
SP_4: metti sul fuoco e porta ad ebollizione
}

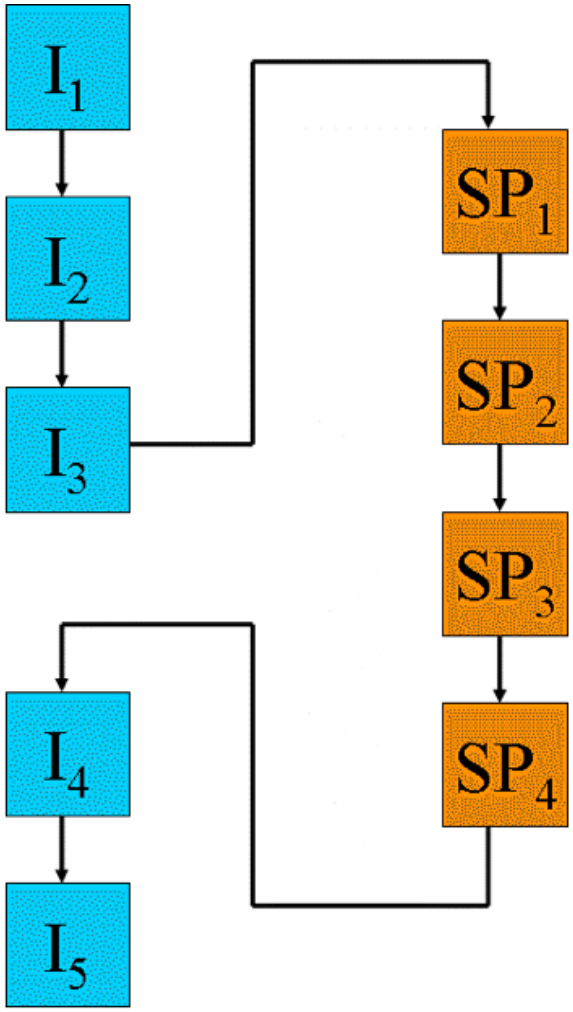
Programma Chiamante

- Si consideri per esempio il testo della ricetta di una torta alla frutta ricopiato da un libro di cucina:

programma torta;
{
l_1: impasta farina, burro, uova, sale e zucchero fino ad ottenere una pasta soffice;
l_2: inforna la pasta per una decina di minuti;
l_3: ***prepara_la_crema***;
l_4: distribuisci la crema sulla sfoglia;
l_5: ricopri con frutta di stagione tagliata a dadini e gelatina.
}

Chiamata a sottoprogramma

- L'esecutore della torta dovrà:
  - spostarsi sulla ricetta della crema quando trova il suo riferimento;
  - riportarsi sulla ricetta della torta nel momento in cui ha terminato la crema.
- In altre parole la chiamata di procedura può essere vista come una particolare istruzione di salto alla prima istruzione della procedura.
- La terminazione della procedura genera poi un ulteriore salto alla istruzione successiva a quella di chiamata.



Vantaggi nell'uso dei sottoprogrammi

- abbreviare il lavoro di scrittura
- articolare, suddividere e strutturare un programma in componenti fra loro coerenti, favorisce:
  - la comprensibilità di un programma
    - soprattutto quando esso è complicato e il testo ha dimensioni che non permettono di scorrerlo con un unico sguardo.
  - la scrittura della documentazione e la verifica.
    - Il sottoprogramma rappresenta una precisa e definita funzionalità ed è dotato di una altrettanto ben definita interfaccia con l'esterno.
      - Così un programma organizzato in sottoprogrammi può essere facilmente controllato.
- Per questo è utile indicare una sequenza di istruzioni con un sottoprogramma anche se essa
  - compare in un sol punto del programma
  - l'introduzione del sottoprogramma non porta ad un testo più breve.

Sottoprogrammi e Metodo Top-Down

- L'adozione del metodo top down comporta un uso naturale dei sottoprogrammi.
- nel passaggio da una frase del linguaggio naturale al suo raffinamento si può far ricorso all'uso di procedure o funzioni ogni qualvolta la funzionalità individuata:
  - viene utilizzata in varie sezioni del programma;
  - è già disponibile in una libreria di sottoprogrammi;
  - può essere utile inserirla in una libreria per poterla riutilizzare in altri progetti;
  - dipende fortemente dal particolare elaboratore che si sta usando;
  - deve nascondere le modalità con cui vengono effettuate le elaborazioni al suo interno;
  - deve contenere astrazioni sui dati in modo che si possa interagire con un particolare tipo di dato soltanto attraverso le operazioni di accesso al tipo senza interessarsi della reale rappresentazione del dato stesso.

La parametrizzazione

- Spesso una sequenza di istruzioni compare in punti diversi non esattamente nella stessa forma.
- Particolare attenzione merita il caso in cui la differenza consiste nell'uso di operandi diversi.

```
{
   stampa('Dammi riempimento:');
   leggi(riemp)
}
```

```
{
   stampa('Numero fogli :');
   leggi(num_fogli)
}
```

Schema comune

```
{
   stampa(messaggio);
   leggi(x)
}
```

Parametri Formali ed Effettivi

- Si definiscono "***parametri formali***" gli oggetti utilizzati nel corpo della procedura.
  - Essi devono essere indicati nel titolo della procedura.
- Gli oggetti da sostituire al posto dei parametri formali, prima di ogni esecuzione, sono detti "***parametri attuali***" (o ***effettivi***) e devono essere specificati in ogni chiamata della procedura o della funzione.
- In altre parole, i parametri formali indicano genericamente su quali oggetti il sottoprogramma deve agire (servono quindi alla sua formulazione), mentre quelli effettivi precisano gli oggetti sui quali il sottoprogramma deve effettivamente operare

Definizione del Titolo del sottoprogramma

- Nel titolo della procedura si specificano
  - il tipo dei parametri
  - [meccanismo di sostituzione]
- Tra il titolo della procedura e la sua chiamata si stabilisce una corrispondenza tra la lista dei parametri effettivi e formali di tipo posizionale:
  - al primo parametro effettivo viene fatto corrispondere il primo parametro formale
  - e così via per i successivi.
- I parametri attuali devono quindi essere forniti con rispetto di numero, tipo e ordine rispetto a quelli formali

Meccanismi di sostituzione: sostituzione per valore

- Assegna al parametro formale del sottoprogramma il valore del corrispondente parametro effettivo.
  - Il parametro effettivo può essere un'espressione e come casi particolari di espressione una costante o una variabile.
    - Se è un'espressione, se ne calcola il valore e lo si assegna al corrispondente parametro formale.
- Garantisce che la variabile passata come parametro effettivo non venga alterata dal sottoprogramma.
  - Difatti il sottoprogramma dopo averne *copiato il valore nella corrispondente variabile formale al momento della chiamata*, lavora in sostanza su una sua **copia** (il parametro formale) e tutte le modifiche effettuate su tale copia non riguardano in alcun modo il corrispondente parametro effettivo.

Meccanismo di sostituzione: sostituzione per riferimento

- fornisce al parametro formale del sottoprogramma **l'indirizzo di memoria del corrispondente parametro effettivo**.
  - Per tale motivo il parametro effettivo può essere soltanto una variabile.
  - A differenza della sostituzione per valore, il parametro formale **non è una copia** ma un'informazione attraverso cui **si lavora direttamente sul parametro effettivo**
    - la sua occupazione di memoria è esclusivamente quella necessaria per contenere l'indirizzo del parametro attuale.

Quale meccanismo scegliere?

- Parametri Formali di Ingresso
  - quando un parametro rappresenta un argomento di una procedura, si sceglie la sostituzione per valore;
- Parametri Formali di Uscita o di Ingresso/Uscita
  - quando un parametro rappresenta invece un risultato, occorre usare la sostituzione per riferimento

Le Funzioni

- Quando una procedura fornisce un unico risultato, può essere organizzata in funzione, in modo che il suo nome corrisponda proprio a tale risultato.
  - Per esempio, si organizzano in funzione le funzioni matematiche che devono essere utilizzate all'interno del programma, o le espressioni o porzioni di esse che devono essere usate più volte all'interno di uno stesso programma.
  - In tali casi si ha che:
    - il nome della funzione è l'unico risultato della procedura;
    - poiché il nome della procedura è una variabile a tutti gli effetti, deve essere dotata di un tipo che deve essere indicato nell'intestazione;
    - si può assegnare valore a tale variabile soltanto nel corpo della funzione stessa;
    - si può usare il valore di tale variabile inserendone il nome in espressioni che si trovano all'esterno del corpo della funzione.

Esempio di Funzione

- una funzione che calcola la somma di due numeri potrebbe essere definita da:

```
funzione somma(a,b:interi):intera
{
  somma:=a+b
}
```

ed essere utilizzata in uno dei seguenti modi:

```
x:=a+somma(3,y)/3;
stampa(somma(alfa,beta));
```

Programma Principale, procedure e funzioni

- L'uso dei sottoprogrammi permette di costruire programmi non costituiti da una unica sequenza di istruzioni ma da una gerarchia di unità di programma
  - o anche macchine astratte se si vuole dare risalto al cosa fanno e non al modo in cui sono realizzate
- ciascuna delle quali realizza una particolare istruzione in modo autonomo, con proprie definizioni di tipi, dichiarazioni di variabili e istruzioni.
- Ogni unità forma la base per il livello superiore (le unità che la chiamano) e si appoggia eventualmente su un livello di macchina inferiore (le procedure che sono chiamate al suo interno).
- Si chiamerà programma principale quella unità di programma che si interfaccia direttamente con il sistema operativo.

Regole di Visibilità

- Con il termine regole di scope (visibilità) si fa riferimento a quelle regole che consentono di determinare i campi di influenza di una variabile (e più in generale di un qualunque oggetto) in tutte le varie parti costituenti un programma.
- In altre parole si dice *scope di un oggetto* la porzione di programma che è in grado di usarla.

Variabili Locali, Non Locali e Globali

- Oggetto Locale ad una procedura:
  - una costante, un tipo, una variabile, una procedura o una funzione
  - è definito ed usato solo all'interno di una determinata procedura.
- Oggetto Non locale ad una procedura:
  - Se è definito nell'unità di programma chiamante, ma risulta visibile alla procedura chiamata attraverso qualche meccanismo
- Oggetto Globale:
  - risulta definito nel programma principale,
    - tale unità è l'unica che può rendere visibili i propri oggetti a tutte le altre in quanto rappresenta la radice della gerarchia di unità di programma
    - Il programma principale è l'unica unità che chiama tutte le altre e che non è chiamata da nessun'altra

Allocazione Dinamica

- Variabili Locali
  - oggetti locali, cioè validi nell'ambito ristretto dell'unità di programma in cui sono definiti,
  - Una volta terminata l'esecuzione del sottoprogramma, tutte le sue variabili locali diventano inaccessibili fino a quando il sottoprogramma non viene richiamato;
    - Ovviamente, nella chiamata successiva, i valori iniziali di queste variabili saranno diversi da quelli finali della chiamata precedente.
  - Life time delle variabili locali
- L'allocazione in memoria centrale delle variabili locali avviene all'atto della chiamata del sottoprogramma
  - Così, al termine di un sottoprogramma l'area di memoria occupata dalle sue variabili locali viene resa disponibile per l'allocazione delle variabili locali di un diverso sottoprogramma.
    - In altre parole, la corrispondenza tra identificatore di variabile e indirizzo del registro di memoria viene determinata dinamicamente durante l'esecuzione.

Allocazione Statica vs Dinamica

- Allocazione statica
  - prevede che l'allocazione venga fissata all'inizio dell'esecuzione del programma e non cambi finché il programma non sia terminato.
- Nella gerarchia di unità di programma,
  - STATICHE: le variabili globali (quelle definite nel programma principale)
  - DINAMICHE: le variabili locali ai vari sottoprogrammi.