La programmazione
ovvero
esprimersi "chiaramente"

This subject is aimed at students with little or no programming experience. It aims to provide students with an understanding of the role computation can play in solving problems. It also aims to help students, regardless of their major, to feel justifiably confident of their ability to write small programs that allow them to accomplish useful goals. (Description of "6.00 Introduction to Computer Science and Programming", course offered at MIT, fall 2008)

§ 0 (Pseudo)Linguaggio (ovvero scrivere in maniera disciplinata)

§ 0.0 Introduzione

Fino ad adesso si sono esaminate delle classi di problemi e sono stati descritti *informalmente* (cioè in linguaggio naturale) alcuni *metodi* per risolvere i problemi di quella classe; l'importante era *saper fare i conti*: quello che si chiedeva, alla fine, era il *risultato* (numero naturale, stringa, lista, ecc.), cioè la soluzione del problema.
Un ulteriore importantissimo passo concettuale, dopo quello di (sapere) *risolvere* un problema, è quello di (sapere) *descrivere il metodo per risolverlo*: questa attività si chiama *programmazione* e il risultato è un *programma*.
Come dice il sottotitolo, un *programma* non è altro che la descrizione "chiara" di un procedimento di soluzione di un problema, cioè è una descrizione che non lascia alcun dubbio sulla interpretazione delle azioni da eseguire. Per ottenere questo risultato, si usa di solito uno strumento linguistico, con una sintassi rigorosamente definita, detto *linguaggio di programmazione*.

Per evitare le difficoltà formali ed arrivare rapidamente al cuore della questione, in questo manuale si studia la programmazione utilizzando particolari forme linguistiche dette *pseudolinguaggio* (*di programmazione*): questo consiste in un uso molto particolare (*limitato* e *disciplinato*) del *linguaggio naturale* (quindi, come si vedrà in seguito, con una sintassi non "completamente definita").

Una maniera "facile" per imparare uno pseudolinguaggio è discutere alcuni esempi, per apprenderne i principali costrutti.

In questo manuale si insegna principalmente **a "leggere" (e "capire") lo pseudolinguaggio. O procedimenti descritti con pseudolinguaggio**

§ 0.1 Primi esempi: variabili e scatole

Si consideri il seguente Esempio 1, dove il simbolo '*' (asterisco) è usato per indicare la moltiplicazione.

```
inizio procedura AreaTrapezio;
      acquisire i valori di Bmin, Bmag, H;
      porre X = (Bmin + Bmag) * H / 2;
      rendere disponibile il valore di X;
       fine procedura;
```

Esempio 1

È facile *intuire* che questa "procedura" calcola un valore di X che può essere pensato come l'area di un trapezio di basi Bmin e Bmag e di altezza H: se all'inizio della procedura vengono acquisiti i seguenti valori:

5 per Bmin; 12 per Bmag; 7 per H;

alla fine (a procedura "eseguita"[¥]) X vale 77.

Per rendere più sistematica la "intuizione" del significato di un testo scritto in pseudo linguaggio, viene presentato un elenco di "regole" (prescrittive) che occorre rispettare (indicate con **R** e un numero, da 1 fino a 7).

R1. La descrizione del procedimento risolutivo di un problema si chiama *procedura* e *deve* avere un nome ("AreaTrapezio" nell'esempio sopra visto); una procedura ha sempre un inizio e una fine.

R2. Una procedura parla essenzialmente di oggetti che si chiamano *variabili*[†]. Per capire cosa sia una variabile si può pensare a una *scatola* (si veda la Figura 1) che ha:

- un *nome*,
- un contenuto o *valore*.

Il nome consiste in una sigla composta di lettere e numeri: compaiono in prima posizione solo lettere *maiuscole*: ad esempio

A, B, Dare, Z4, H1N1, Base, AlfaBeta

sono nomi corretti; *non* lo sono

2A, alfa, betA

Il valore può essere di vari *tipi*: un numero intero (per esempio - 6), un numero razionale (per esempio 6,28), una stringa (per esempio 'ABC').

Nel seguito, per un po', si userà indifferentemente il termine "scatola" o "variabile"

[¥] Per una (più) precisa definizione di *esecuzione*, si veda in seguito.

[†] Esercizio (linguistico, consigliato). Studiare i vari significati che ha la parola "variabile", come aggettivo e come sostantivo; prendere in considerazione (almeno) il linguaggio corrente, quello della matematica e quello della fisica, aiutandosi con dizionari ed enciclopedie su Internet. Ripetere lo stesso esercizio per la parola "costante".

(anche in una medesima frase) per intendere la stessa cosa; successivamente si parlerà solo di variabili.

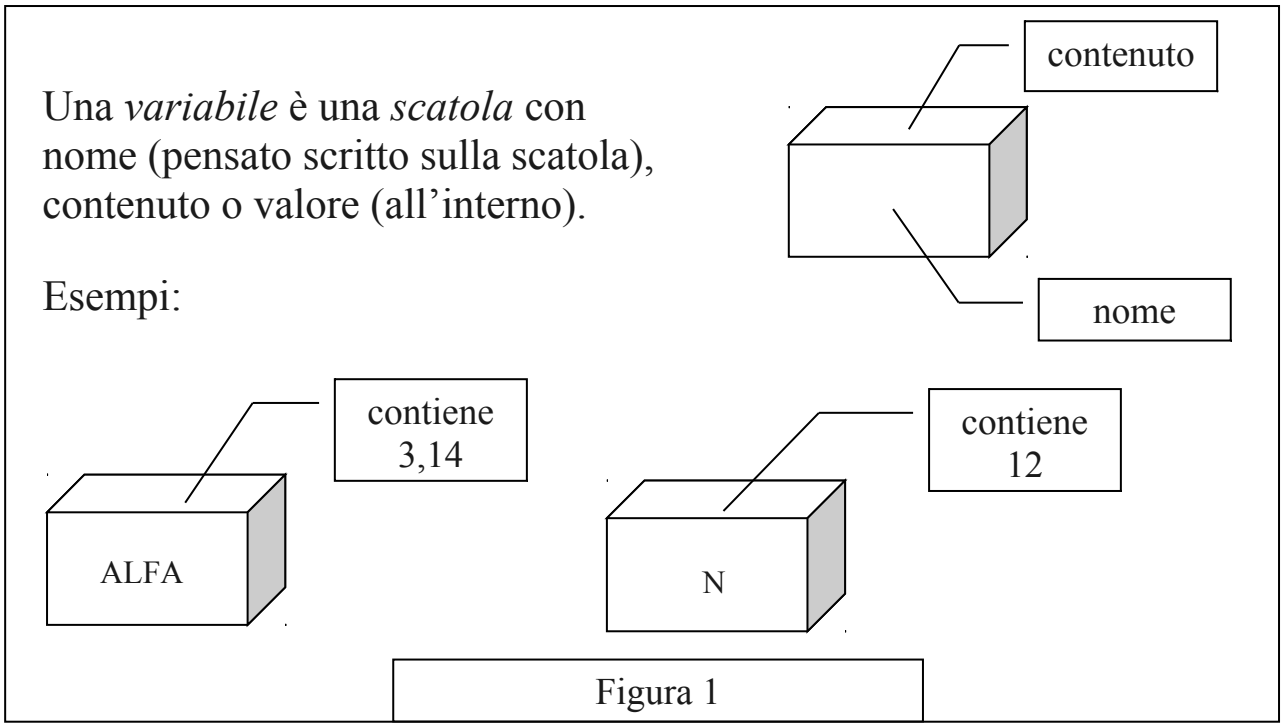


Figura 1

Nella Figura 1, ALFA ha contenuto di tipo razionale (numero con la virgola) e N ha contenuto di tipo intero.

R3. <u>All'inizio di ogni procedure, le scatole sono vuote. Il valore di una variabile (contenuto nella scatola) può essere definito o cambiato:</u>

- "acquisendolo" (dall'esterno),
- ponendolo uguale a quello di altra scatola,
- ponendolo uguale al risultato di (un calcolo descritto da) una espressione.

Il contenuto di una scatola può essere reso "disponibile" (all'esterno).
Si possono manipolare (sommare, sottrarre, unire, ecc.) i valori delle varie scatole in *espressioni*, in maniera congrua con il loro contenuto (per esempio si possono sommare due numeri ma non due stringhe). Nelle espressioni si usano sempre i nomi delle scatole per indicare i contenuti: per esempio

A + B

indica la somma del *contenuto* della scatola A e di quello della scatola B.

Si noti che spesso sono usate scritture del tipo

porre X = X + Z;

(in cui il nome X compare a destra e a sinistra del segno di eguale); queste espressioni sono comprensibili solo se si pensa alle variabili come scatole con un contenuto; si dà senso alla scrittura interpretandola come: *sommare i contenuto delle scatole X e Z* (espressione a destra del simbolo "=") e *porre il risultato nella scatola X*.

N.B. Ogni volta che viene cambiato il valore di una variabile (cioè il contenuto di una scatola), quello precedente viene "perso".

R4. È opportuno (per facilità di lettura) all'inizio, elencare il nome delle scatole, col tipo (o natura) del contenuto che può essere (per esempio):

- numero intero,
- numero razionale,
- stringa.

Tale elenco si chiama *dichiarazione delle variabili*.

Si consideri il seguente Esempio 2, in cui il simbolo '**' (doppio asterisco) indica l'elevamento a potenza.

```
 inizio procedura AreaCerchio;
dichiarazione delle variabili: X, R razionali;
acquisire il valore di R;
porre X = 3.14*R**2;
rendere disponibile il valore di X;
 fine procedura;
```

Esempio 2

È naturale interpretare il valore di X come misura dell'area del cerchio il cui raggio è contenuto nella scatola **R (utilizzando per pi greco il valore approssimato 3.14).**

Una operazione "delicata" è scambiare il contenuto di due scatole, per esempio A e B (supposte dello stesso tipo). Si osservi la Figura 2.

Scambiare il contenuto delle scatole A e B
Attenzione all'ordine (numero sulle frecce!)

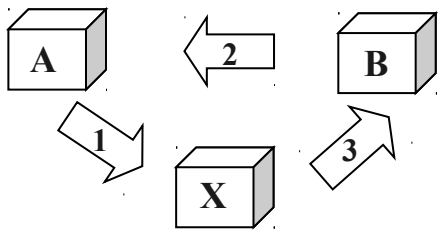


Figura 2

Non si può semplicemente porre il contenuto di B in A: il contenuto di quest'ultima andrebbe perso. Per lo scambio occorre una ulteriore varabile X (dello stesso tipo del altre due) in cui "conservare" il valore di A (talvolta è detta variabile "di appoggio"). Nello pseudolinguaggio, lo scambio si ottiene quindi nel modo seguente:

porre X = A;
porre A = B;
porre B = X;

§ 0.2 Come esprimere le scelte

Il seguente esempio illustra un *costrutto sintattico* (cioè un modo di esprimersi!) "nuovo", ***detto struttura condizionale***.

```
 inizio procedura Maggiore;
dichiarazione delle variabili: A, B, C intere;
acquisire i valori di A e B;
se A > B
        allora              porre C = A;
        altrimenti    porre C = B;
fine del condizionale;
rendere disponibile il valore di C;
fine procedura;
```

Esempio 3

È facile intuire che questa procedura "calcola" C come il maggiore dei due numeri dati A e B.
Più esattamente: vengono acquisiti i valori per le due scatole A e B; successivamente viene confrontato il valore delle due scatole: se il valore della prima è maggiore di quello della seconda si trasferisce il suo contenuto nella scatola C, altrimenti in C viene trasferito il contenuto di B. Alla fine si rende disponibile il contenuto di C (che è quindi uguale al maggiore tra i contenuti di A e B).

Quanto visto di nuovo in questo secondo esempio viene formalizzato nella seguente regola 5.

R5. Nelle procedure si può usare il costrutto linguistico:

se *predicato*
allora *alternativa 1*
altrimenti *alternativa 2*
fine del codizionale;

detto *costrutto condizionale*: per renderne più evidente la struttura si è indicata in grassetto la parte fissa e in corsivo la parte (variabile) che dipende dal procedimento che si deve descrivere. Questo costrutto ha il significato intuitivo **di valutare il predicato e** di eseguire le azioni descritte in *alternativa 1* o quelle di *alternativa 2*, a seconda se *predicato* è, rispettivamente, vero o falso.

N.B. Si dice *predicato* una *proposizione* che può essere (solo!) vera o falsa; un esempio tipico di predicato è il "confronto" tra (i valori di) due scatole: per esempio *asse-*

rire che il contenuto di una è maggiore del contenuto dell'altra (oppure minore, oppure eguale).

Il costrutto condizionale può anche essere impiegato nella forma ridotta:

se *predicato*
allora *alternativa 1*
fine del condizionale;

in cui manca, appunto, la seconda alternativa.

Consideriamo un ulteriore esempio.

```
inizio procedura Confronto;
        dichiarazione delle variabili: B1, H1, B2, H2, A1, A2 razionali;
        dichiarazione delle variabili:  X intera;
        acquisire il valore di B1, H1, B2, H2;
        porre A1 = B1*H1;
        porre A2 = B2*H2;
       se A1>A2
              allora                  porre X = 1;
        fine condizionale;
        se A1<A2
              allora          porre X = 2;
        fine condizionale;
        se A1=A2
              allora porre X = 3;
        fine condizionale;
        rendere disponibile il valore di X;
fine procedura;
```

Esempio 4

Il valore di X può essere interpretato per verificare il rapporto fra le aree dei due rettangoli con dimensioni (contenute nelle scatole) B1, H1 e B2, H2.

Una caratteristica dello **pseudolinguaggio** molto utile alla sua comprensione *è la indentazione*: cioè la possibilità (in realtà l'obbligo!) di scrivere le varie righe con l'inizio spostato a destra e opportunamente incolonnate, per identificare più facilmente le varie componenti dei costrutti sintattici usati.

§ 0.3 Scatole (o variabili) particolari: le costanti

Negli esempi precedenti si sono usate delle costanti numeriche: 1, 2, 3.14; queste sono "spiegate" dalla seguente regola.

R6. Nelle procedure si possono usare le costanti necessarie; queste sono, in realtà,

nomi di scatole il cui contenuto è "corrispondente" al nome; si veda la Figura 3.

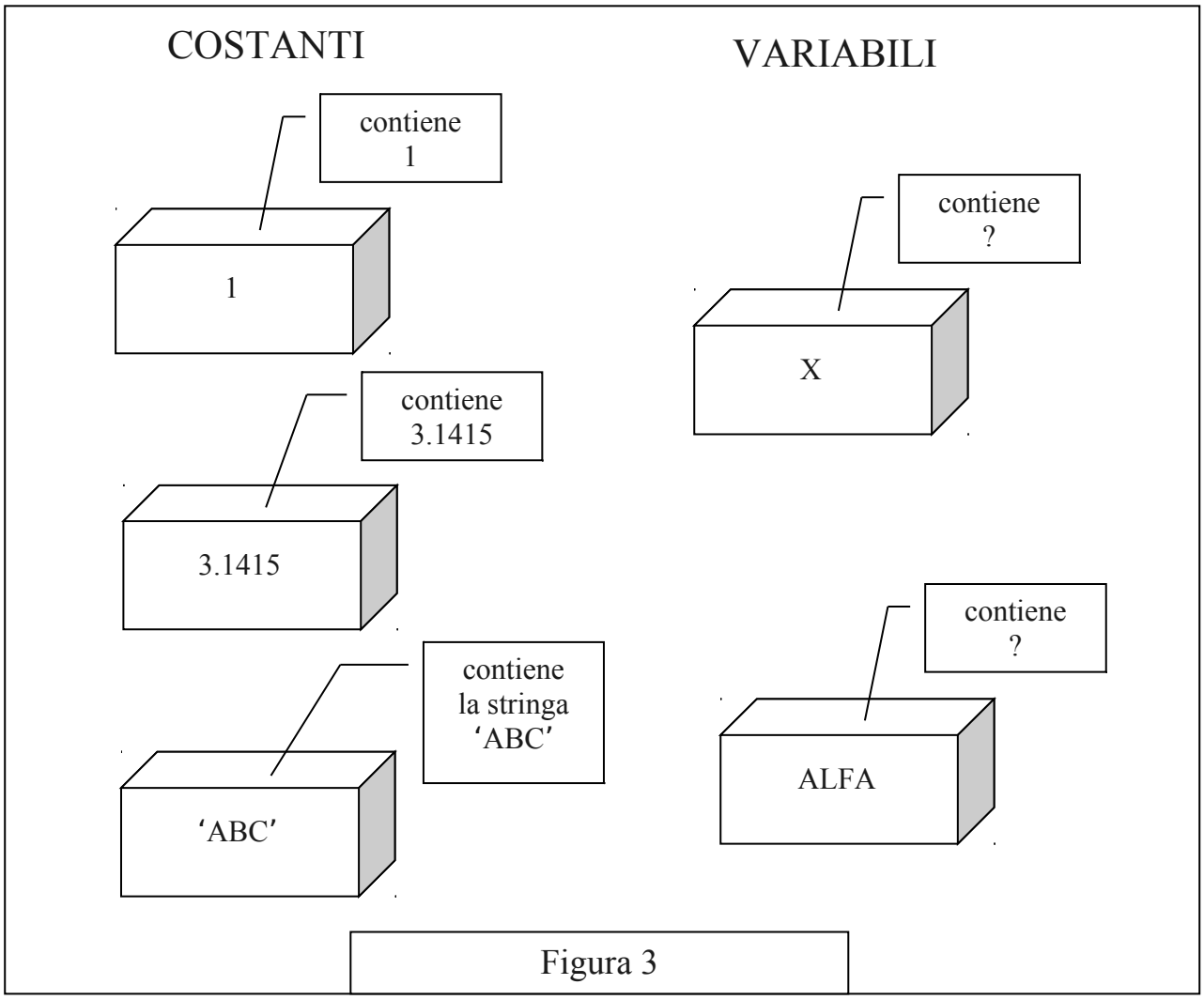


Figura 3

La parola "costante" indica appunto che la scatola ha contenuto fisso, determinato a priori e congruente col suo nome; la parola "variabile" indica che la scatola non ha un contenuto fisso, che è determinato e manipolato dalla procedura.

N.B. Prima di passare ad un altro esempio è bene chiarire perché lo pseudolinguaggio (qui trattato) parli essenzialmente di scatole (o variabili). L'obbiettivo di descrivere in maniera "chiara" come risolvere un problema deve essere raggiunto (o almeno perseguito) per *due* (classi di) *interlocutori*: le persone e i *computer*. Mentre le persone possono ritenere le variabili una complicazione (inutile), i *computer* "ragionano" (quasi solamente) in termini di manipolazioni di scatole con un nome e un contenuto. Potrebbe sembrare quindi che la introduzione (esplicita) delle variabili pensate come scatole sia una compiacenza verso interlocutori "poco intelligenti": in realtà c'è una ragione molto più profonda, che in questa sede può essere solo accennata: si può dire che l'informatica sia nata (e coincida) con lo studio di quali problemi siano risolubili

in maniera "effettiva"[†]: usare un linguaggio "comprensibile" (in linea di principio anche) da un *computer* significa, in realtà, assicurarsi che la soluzione del problema è "effettivamente realizzabile".

§ 0.4 Come ripetere

Si consideri il seguente Esempio 5.

```
inizio procedura Sommaquadrati;
        dichiarazione delle variabili: N, S, H, K intere;
        acquisire il valore di N;
        porre S = 0;
        per tutti i valori interi di K a partire da 1 fino a N ripetere il ciclo
                porre H = K*K;
                porre S = S+H;
        fine ciclo;
        rendere disponibile il valore di S;
fine procedura;
```

Esempio 5

È abbastanza intuitivo che la procedura calcola la somma dei quadrati dei numeri interi da 1 al valore di N. Se per N viene acquisito il valore 4, il ciclo viene ripetuto 4 volte; i successivi valori di H sono 1, 4, 9, 16; il risultato finale, contenuto in S è quindi 30.

R7. Nelle procedure si può usare il costrutto linguistico:

per tutti i valori interi di *variabile* **a partire da** *scatola1* **fino a** *scatola2* **ripetere**
elaborazioni ***del ciclo*****;**
fine ciclo;

detto *costrutto ripetitivo*; per renderne più evidente la struttura si è indicata in grassetto la parte fissa e in corsivo la parte che può cambiare e dipende dal procedimento che deve essere descritto; esso ha il significato di eseguire le ***elaborazioni del ciclo*** più volte: la prima volta con *variabile* con contenuto uguale a *scatola1*; le volte successive il contenuto di *variabile* viene aumentato di 1 e si smette di ripetere il ciclo quando tale contenuto supera quello di *scatola2*. Si noti che *variabile* deve essere una variabile, mentre *scatola1* e *scatola2* possono essere variabili o costanti.

N.B. Per evitare "patologie" si supporrà sempre che il valore contenuto in *scatola1* sia minore o eguale a quello di *scatola2*; con questa ipotesi le *elaborazioni* sono ese-

[†] Per il momento è sufficiente il significato intuitivo di *effettivo*: cioè che possa essere effettivamente calcolato.

guite almeno una volta. Si fa inoltre l'ipotesi che alla fine del ciclo *variabile* abbia lo stesso valore di *scatola2*.

Esercizio *in itinere* (obbligatorio!). In relazione alla procedura mostrata in esempio 3, considerare la scatole relative alle variabili usate ed *eseguire* la procedura: questo consiste nell'effettuare le varie attività, specificate dalla procedura, che cambiano valore alle variabili e annotare tale cambiamento. Per "aiutarsi" si usi una tabella (vedere sotto) con tante colonne quante sono le variabili e tante righe quanti sono i cambiamenti di valore di (almeno) una variabile.

N.B. Il valore di una variabile "all'inizio" è indefinito e indicato con un '?'; la tabella è detta talvolta "storia computazionale".

N	K	H	S	
?	?	?	?	inizio
4	?	?	?	acquisire il valore di N
4	?	?	0	porre S=0
4	1	?	0	per… K uguale a 1
4	1	1	0	porre H = K*K
4	2	1	1	per… K uguale a 2
4	2	4	1	porre H = K*K
4	2	4	5	porre S = S+H
4	3	4	5	per… K uguale a 3
4	3	9	5	porre H = K*K
4	3	9	14	porre S = S+H
4	4	9	14	per… K uguale a 4
4	4	16	14	porre H = K*K
4	4	16	30	porre S = S+H

Ripetere l'esercizio 2 volte, supponendo che il valore acquisito per la variabile N sia rispettivamente 5 e 6.

N.B. La capacità di *eseguire una procedura* (si dice anche *eseguire manualmente*) per capire o verificare cosa fa, è l'*abilità fondamentale* che si acquisisce leggendo questo manuale e costituirà la forma della maggior parte degli esercizi successivi.

§ 0.5 Scatole (o variabili) con struttura

Un ulteriore evoluzione del concetto scatole/variabili è quello di dotarle di *struttura*: cioè di pensare che una scatola sia suddivisa in comparti. Un primo semplice esempio di variabile con struttura è mostrato in Figura 4.
In essa è mostrata una *struttura* regolare: tutti i comparti hanno la medesima organizzazione nel senso che sono variabili "semplici" e hanno contenuti della stessa natura (o tipo); questa struttura è detta *vettore*; nell'esempio è mostrata una variabile di nome A con 6 comparti (ciascuno dei quali si chiama anche *componente* del vettore); il numero di comparti si dice anche *lunghezza* (o *dimensione*) del vettore. È inoltre esemplificata la regola per costruire il nome di un comparto (dato il nome della scatola).

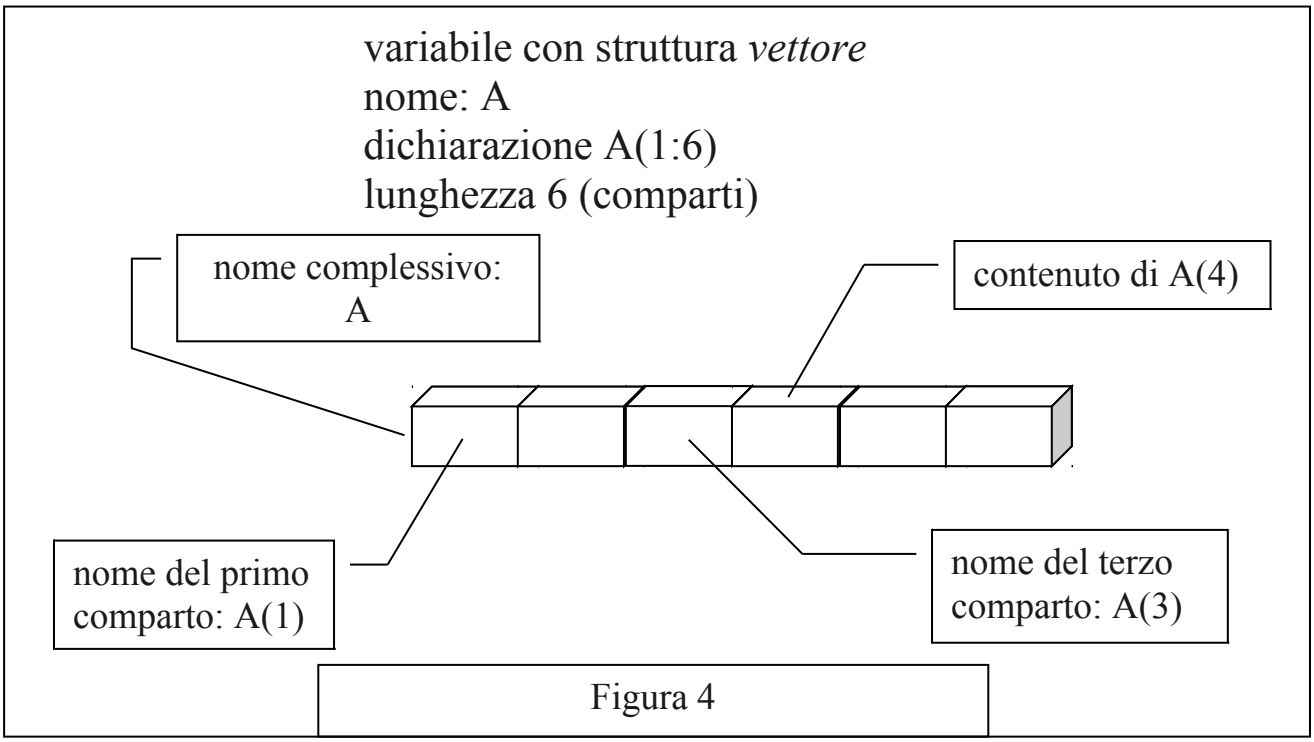


Figura 4

L'importanza dei vettori nasce dall'osservazione che nel nome di un comparto, ad esempio il terzo:

A(3)

compare una costante (appunto 3 nell'esempio); ricordando che una costante è (comunque!) una scatola, si può facilmente dare significato alla scrittura:

A(J)

Se J è una variabile che assume valori interi, allora A(J) è il (nome di un) comparto del vettore A il cui numero d'ordine è (contenuto nel)la variabile J. Per esempio se J contiene 2, allora A(J) è il (nome del) secondo comparto; se J contiene 5 allora A(J) indica il quinto comparto. Si dice che J è un *indice* per il vettore A.

N.B. Se la variabile J vale 1000, allora A(J), che equivale ad A(1000), indica un comparto non esistente (nell'ipotesi che il vettore A abbia solo 6 componenti o dimensione 6): quindi è un nome (di comparto) *illecito*.

Si consideri il seguente Esempio 6.

```
inizio procedura SommeVettore;
        dichiarazione delle variabili A(1:6), S1, S2 e K intere;
        acquisire i valori di A(1), A(2), A(3), A(4), A(5), A(6);
        porre S1=0;
        porre S2=0;
        per tutti i valori interi di K a partire da 1 fino a 6 ripetere
                se A(K)> 0
                      allora          porre S1=S1+A(K);
                      altrimenti      porre S2=S2+A(K);
                fine del condizionale;
        fine del ciclo;
        rendere disponibile i valori di S1 e S2;
fine procedura;
```

Esempio 6

È facile verificare che questa procedura calcola S1 come somma dei valori positivi delle componenti del vettore A e S2 come somma di quelli negativi. La variabile K si dice *indice* del costrutto ripetitivo.

È bene notare ancora il ruolo, fondamentale per la leggibilità, della indentazione.

Il costrutto, visto prima, "per tutti i valori interi … fine ciclo" è usato spesso in procedure che impiegano variabili vettori, perché consente di manipolare "facilmente" tutte (o una parte de) le sue componenti.

Esercizio *in itinere* (obbligatorio!). In relazione alla procedura mostrata in esempio 6, disegnare la scatole relative alle variabili usate ed *eseguire* la procedura; come si è visto questo consiste nell'effettuare le varie attività, specificate dalla procedura, che cambiano valore alle variabili; si può usare una tabella con tante colonne quante sono le variabili che si intende "tenere sotto controllo".
Ripetere l'esercizio tre volte, supponendo che i (6) valori acquisiti per la componenti di A siano nell'ordine:

1, -2, 3, -4, 5, -6	la prima volta,
15, -1, 3, 4, 8, -1	la seconda volta,
0, -1, 1, 0, -1, 1	la terza volta;

e annotare sia la “storia” del valore per le variabili S1 e S2 durante l’esecuzione, sia i valori resi disponibili quando (l’esecuzione de) la procedura “termina”: cioè quando non vi sono più attività da eseguire.

§ 0.6 Esercizi: saper leggere (1)

ESERCIZIO 1
Sia data la seguente procedura:

```
inizio procedura Prodottoscalare;
        dichiarazione delle variabili A(1:5),  B(1:5),  K, S intere;
        acquisire i valori di A(1), A(2), A(3), A(4), A(5);
        acquisire i valori di B(1), B(2), B(3), B(4), B(5);
        porre S=0;
        per tutti i valori interi di K a partire da 1 fino a 5 ripetere
                porre S=S+A(K)*B(K);
        fine del ciclo;
        rendere disponibile il valori di S;
fine procedura;
```

Se per le componenti della variabile A vengono acquisiti i valori 1, 1, 3, 2, 1 e per quelle della variabile B i valori 1, 7, 1, 2, 9 quale è il valore reso disponibile per S?

S	

S	24

ESERCIZIO 2
Sia data la seguente procedura:

```
inizio procedura Incognita1;
        dichiarazione delle variabili A(1:10), K, M intere;
        acquisire i valori di A(1), A(2), A(3), A(4), A(5), A(6), A(7), A(8), A(9),
        A(10);
        porre M = A(1);
        per tutti i valori interi di K a partire da 2 fino a 10 ripetere
                se A(K) > M
                        allora          porre M = A(K);
                fine del condizionale;
        fine del ciclo;
        rendere disponibile il valori di M;
fine procedura;
```

Se per le componenti della variabile A vengono acquisiti i valori 1, 1, 3, 2, 1, 12, 2, 18, 15, 7 quale è il valore reso disponibile per M?

M	

M	18

ESERCIZIO 3
Sia data la seguente procedura:

```
inizio procedura Incognita2;
        dichiarazione delle variabili A(1:10), K, M1, M2 intere;
        acquisire i valori di A(1), A(2), A(3), A(4), A(5), A(6), A(7), A(8), A(9),
        A(10);
        porre M1 = A(1);
        porre M2 = A(1);
        per tutti i valori interi di K a partire da 2 fino a 10 ripetere
                se A(K) > M1
                        allora          porre M1 = A(K);
                fine del condizionale;
                se A(K) < M2
                        allora          porre M2 = A(K);
                fine del condizionale;
        fine del ciclo;
        rendere disponibile il valori di M1, M2;
fine procedura;
```

Se per le componenti della variabile A vengono acquisiti i valori 1, -1, 51, -2, 1, -12, 2, 26, 15, -7 quale sono i valori resi disponibili per M1 e M2?

M1	
M2	

M1	51
M2	-12

ESERCIZIO 4
Sia data la seguente procedura:

```
inizio procedura Incognita3;
        dichiarazione delle variabili A(1:10), N, K, M1, M2, M intere;
        acquisire il valore di N;
        per tutti i valori interi di K a partire da 1 fino a N ripetere
                acquisire il valore di A(K);
        fine del ciclo;
        porre M1 = A(1);
        porre M2 = A(1);
        per tutti i valori interi di K a partire da 2 fino a N ripetere
                se A(K) > M1
                        allora          porre M1 = A(K);
                fine dell'alternativa;
                se A(K) < M2
                        allora          porre M2 = A(K);
                fine dell'alternativa;
        fine del ciclo;
        porre M = M1-M2;
        rendere disponibile il valore di M;
fine procedura;
```

Se per N viene acquisito il valore 7 e per le componenti della variabile A vengono acquisiti i valori 1, -1, 3, -2, 1, -17, 2 qual è il valore reso disponibile per M?

M	

M	20

Argomenti di riflessione.

1. cosa "succede" se il valore acquisito per N fosse 24?
2. come si potrebbe modificare la procedura per "comportarsi bene" anche in questo caso?

§ 1 *Syntactic sugar* (ovvero una maniera "elegante" di scrivere)

§ 1.0 Statement

I testi di procedure scritti in pseudolinguaggio si possono chiamare *pseudoprogrammi* (anche se quelli più semplici vengono indicati semplicemente come "descrizione di un algoritmo")
Per proseguire la descrizione dello pseudolinguaggio è importante capire (almeno approssimativamente) cosa è uno *statement*; questo concetto, in un linguaggio (di programmazione, che ha una sintassi *formale*) è facilmente definibile. In uno pseudolinguaggio (dalla sintassi non completa), con riferimento agli esempi e agli esercizi sopra riportai si può dire che "grossomodo" (non è importante in questa sede una definizione precisa) uno *statement* è:

1. una riga di pseudolinguaggio oppure
2. ciò che termina con ';' (e inizia dopo un altro ';') oppure
3. ogni costrutto linguistico "semplice" con significato definito.

§ 1.1 Forma "standard" dello pseudolinguaggio

Esiste una maniera molto usata di "scrivere" (e usare) lo pseudolinguaggio: è particolarmente utile perché contribuisce alla sinteticità e alla comprensione dei testi.
La nuova maniera di scrivere viene presentata dapprima come "traduzione" (in una sorta di inglese!) della vecchia, poi viene estesa (a nuovi costrutti).

VECCHIA FORMA	NUOVA FORMA
procedura	**proc**
fine procedura	**endproc**
dichiarazione delle variabili	**var**
intero	**ineger**
razionale	**floating**
stringa	**string**
porre … = …	… **:=** …
se … allora … altrimenti … fine condizionale	**if** … **then** … **else** … **endif**

per tutti i valori interi di … a partire da … fino a … ripetere
…
fine del ciclo
	for … **from** … **to** … **do**
	…
	endfor

acquisire i valori …	**read**
rendere disponibili i valori di	**write**

Come si vede la maggior parte delle parole usate è stata tradotta (o abbreviata) in inglese e scritta in grassetto; (scrivendo a mano invece del grassetto si può usare la sottolineatura). Una notevole eccezione sono gli *statement* come i seguenti:

porre F2 = TTRE;
porre CIRCOFERENZA = RAGGIO * 6,28

che si traslitterano rispettivamente:

F2 **:=** TTRE;
CIRCOFERENZA **:=** RAGGIO * 6,28

Tali *statement* (poiché assegnano il valore a una variabile) si dicono di *assegnazione*: in essi si è soppresso 'porre' e si è sostituito il simbolo '=' (che è di natura "simmetrico") con il simbolo **':='** (che è "asimmetrico").

N.B. Come si è osservato la sintassi dello pseudo linguaggio non è definita rigorosamente e in maniera completa; in particolare è bene seguire i seguenti "consigli":

1. è bene mettere un ';' alla fine di ogni *statement* (indicazione "circolare" con un punto della definizione di *statement*);
2. è bene indentare la scrittura per renderla più leggibile;
3. si può mischiare la vecchia e la nuova forma;
4. si possono usare "commenti": cioè frasi in linguaggio naturale racchiuse tra parentesi quadre; *non fanno parte della procedura, ma possono aiutarne la lettura o spiegarne lo scopo*.

Con questi accorgimenti, la procedura dell'esercizio 3 può essere riscritta nella maniera seguente.

```
1   proc Incognita2;
2         var A(1:10), K, M1, M2 integer;
3
4         [acquisizione del valore per le componenti del vettore A]
5         read A(1), A(2), A(3), A(4), A(5), A(6), A(7), A(8), A(9), A(10);
6
7         [inizializzazione delle variabili M1 e M2 al valore della prima
8         componente di A]
9         M1 :=  A(1);
1         M2 :=  A(1);
0
1         [ciclo per esaminare le altre componenti di A]
1         for K from 2 to 10 do
1
2               [test per trovare il più grande]
1               if A(K) > M1
3                     then   M1 :=  A(K);
1               endif;
```

```
4
1                [test per trovare il più piccolo]
5                if A(K) < M2
1                      then   M2 :=  A(K);
6                endif;
1
8          endfor;
1          write M1, M2;
9    endproc;
2
0
2
1
2
2
2
3
2
4
2
5
2
7
2
8
2
9
3
0
```

Si notino i commenti e le spaziature; i primi, aggiunti per facilitare la comprensione del testo, si riferiscono a quanto segue, le seconde separano i vari "passi" dello pseudopregramma. Da adesso in poi, per facilitare la discussione degli esempi *possono* essere aggiunti i numeri di riga.

Analogamente, la procedura dell'esercizio 4 può essere riscritta nella maniera seguente.

```
1   proc Incognita3;
2         var A(1:10), N, K, M1, M2, M integer;
3         read N;
4         for K from 1 to N do
5               read A(K);
6         endfor;
7         M1 := A(1);
8         M2 := A(1);
9         for K from 2 to N do
10              if A(K) > M1
11                    then  M1 := A(K);
12              endif;
13              if A(K) < M2
14                    then  M2 := A(K);
15              endif;
16        endfor;
17        M := M1-M2;
18        write M;
19  endproc;
```

§ 1.2 Un costrutto più generale per ripetere

Un altro costrutto "per ripetere" è il seguente:

while *predicato* **do**
elaborazioni;
endwhile;

Ha il significato seguente:

valutare *predicato*,
se *predicato* è vero allora eseguire *elaborazioni*,
valutare *predicato*,
se *predicato* è vero allora eseguire *elaborazioni*,
…
valutare *predicato*,
se *predicato* è falso eseguire lo *statement* successivo (a **endwhile**).

Si può anche dire che si eseguono (ripetono) le *elaborazioni* fintantoché il *predicato* è vero. Naturalmente, durante la esecuzione di *elaborazioni*, per evitare un ciclo infinito, bisogna far cambiare valore di almeno una variabile che compare in *predicato*, in modo da farlo diventare falso.

N.B. Per spiegare il "significato" del costrutto, come già fatto per i precedenti, si è descritto come "viene eseguito".

Per esempio il seguente frammento (supposto che le variabili A, J, N siano state dichiarate opportunamente ed N abbia un opportuno valore maggiore di 1e non maggiore della dimensione di A):

```
1  …
2          for J from 1 to N do
3                  A(J) := 0;
4          endfor;
5  …
```

pone a 0 alcune componenti del vettore A; ha lo stesso significato del seguente frammento:

```
1  …
2          J := 1;
3          while  J ≤ N do
4                  A(J) := 0;
5                  J := J+1;
6          endwhile;
7          J := J-1;
8  …
```

Esercizio *in itinere* (obbligatorio!). In relazione alla equivalenza dei due precedenti frammenti, spiegare perché è necessario lo statement alla riga 7; come lo si può cambiare, senza alterare il "significato" del frammento?

§ 1.3 Come finire una procedura

Dal punto di vista puramente "testuale" una procedura inizia con **proc** (nella nuova notazione) e finisce con **endproc**. Molto spesso, comunque, con le parole "inizio" e "fine" di una procedura si intende l'inizio e la fine dell'*esecuzione* di una procedura. Quindi (l'*esecuzione* di) una procedura inizia con l'esecuzione del primo *statement* che può essere eseguito: nello pseudolinguaggio qui esposto *non* sono eseguibili gli statement che cominciano per **proc** e **endproc**. L'esecuzione di uno statement che comincia con **var** "*crea*" le scatole che vengono nominate di seguito. Degli altri statement si è già detto. Concludendo, per il momento, (l'*esecuzione* di) una procedura termina quando non ci sono più *statement* da eseguire.
A volte però è utile poter indicare la *fine dell'esecuzione* in maniera *esplicita*: si usa per questo lo statement **stop**. Un suo uso è mostrato dal seguente esempio che è il frammento iniziale di uno pseudoprogramma.

```
1   proc EsempioS;
2         var A(1: 100), N, J integer;
3         read N;
4         [controllo del valore acquisito]
5         if N > 100
6               then  write  'dato errato', N;
7                     stop;
8         endif;
9         for J from 1 to N do
1               ....
0               A(J) := …
…   …
    endproc;
```

Si osservi lo **stop** inserito nello statement **if** (dalla riga 5 alla 8). Senza questo **stop** se il valore acquisito per la variabile N è maggiore di 10, nello *statement* **for** successivo si possono costruire dei nomi per i comparti di A illeciti (riga 10). In questo caso un *buon* (pseudo)programma deve terminare la sua esecuzione.
Si noti lo *statement* **write**: mette a disposizione il contenuto di due scatole: il valore di una costante (la stringa 'dato errato') e quello della variabile N appena acquisita, in modo che "chi" ha fornito il valore (errato) per N "si accorga dell'errore" e non aspetti che venga reso disponibile un risultato utile.
Si noti inoltre l'utilità, per una facile lettura, del *commento* (riga 4).

Esercizio *in itinere* (obbligatorio!). Riscrivere il frammento sopra visto senza usare lo *statement* **stop**, ma prevenendo comunque la costruzione di nomi illeciti durante l'esecuzione.

§ 1.4 Esercizi: saper leggere (2)

ESERCIZIO 1

```
proc EsempioS;
      var A, S,  J integer;
      read A;
      S:=0;
      J:=1;
      while  S ≤  A do
            S := S + J**2;
            J := J+1;
      endwhile;
      J := J-1;
```

```
        write J
endproc;
```

Nella tabella sotto riportata, scrivere di fianco ad ogni valore acquisito per A il corrispondente valore di J reso disponibile ad ogni esecuzione.

A	J
100	
0	
-2	
1	

§ 1.5 Un nuovo "tipo" di variabile

Si sono visti vari tipi di contenuto per le scatole (cioè di *valore* per le variabili):

- numeri interi (**integer**),
- numeri razionali (**float**),
- stringhe (**string**).

Si è anche parlato delle *costanti*: cioè scatole il cui contenuto non è modificabile ed è "corrispondente" al loro nome; naturalmente esistono infinite costanti per ognuno dei tipi di contenuto appena visti.
Viene ora descritto un nuovo tipo di scatola, detto logica o **boolean**, che può contenere valori scelti da un insieme di due elementi: {*true*, *false*} (*vero*, *falso*).
Un esempio d'uso è mostrato dal seguente frammento di pseudoprogramma.

```
1   proc EsempioL;
2           var A(1: 100), N, J integer;
3           var Alfa, Continua boolean;
4           …
5           Alfa := false;
6           …
7           if Alfa
8                   then  …
9                   else  …
1           endif;
0           …
1           Continua := true;
1           while Continua do
1           …
2           endwhile;
1           …
3   endproc;
1
4
1
5
1
6
1
7
```

Si noti la dichiarazione delle variabili **boolean** Alfa e Continua; alla riga 5 nella scatola Alfa viene posto il valore *false* e alla riga 12 alla scatola Continua viene dato il valore *true*. L'uso comune delle variabili **boolean** è come *predicato* nei costrutti **if** e **while**; se il valore di Alfa non viene cambiato (nella parte omessa: riga 6 …), nell'eseguire il costrutto **if** (riga 7 e seguenti) saranno compiute le attività specificate dagli *statement* del "ramo" **else**. Analogamente gli *statement* (omessi, riga 14) del "corpo"

del costrutto **while** (dalla riga 13 alla 15) devono intendersi eseguiti ciclicamente finché (al loro interno!) sarà cambiato il valore di Continua.
Si possono usare espressioni del tipo:

NOT Alfa
Alfa **AND** Continua
Alfa **OR** Continua

cioè si possono costruire espressioni con variabili logiche mediante i "connettivi" logici **NOT**, **AND**, **OR** (che mantengono il significato usuale). Se del caso, le espressioni logiche possono contenere delle parentesi ed essere usate nei costrutti **if** e **while** al posto dei *predicati*.

§ 1.6 Una nuova struttura

Nel paragrafo 0 si è visto un solo esempio di scatola (o variabile) con struttura: il vettore, che si può immaginare come una scatola con tanti scomparti *messi in* (una) *fila*.
Un esempio più complesso sono le *matrici*: scatole con comparti disposti su più file *eguali* (si può pensare ai contenitori per le uova oppure a quelli per gli ornamenti per l'albero di Natale).
Un esempio, con la relativa dichiarazione è mostrato in Figura 5.

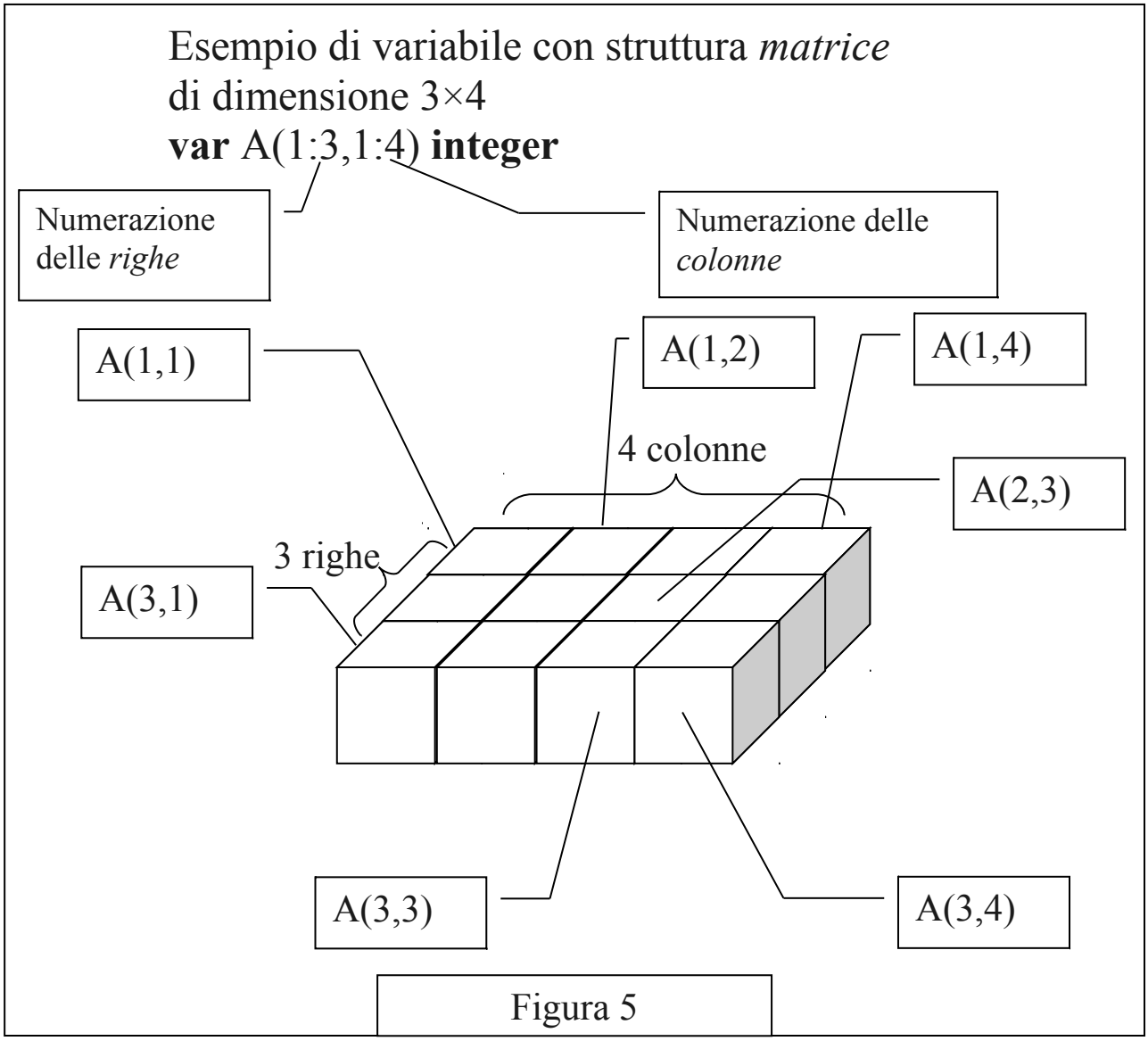


Figura 5

Come si vede i vari comparti della scatola A vengono indicati con un "*doppio indice*": ci sono i seguenti 12 comparti (di seguito *elencati per righe*, cioè la prima riga, seguita dalla seconda, ecc.):

A(1,1), A(1,2), A(1,3), A(1,4),
A(2,1), A(2,2), A(2,3), **A(2,4)**,
A(3,1), A(3,2), A(3,3), A(3,4).

Come per i vettori, naturalmente anche per indicare i comparti delle matrici si possono usare *indici* (che sono) variabili, invece di indici (che sono) costanti. Se per esempio, le variabili *intere* I, J contengono i valori 2 e 4 allora la scrittura

A(I,J)

indica il comparto che sta all'incrocio della seconda riga (primo indice) e quarta colonna (secondo indice), in grassetto nell'elenco precedente.

Una scatola (matrice) come quella appena vista è "rettangolare", cioè ha più colonne che righe: possono esistere, ovviamente, scatole "quadrate" (tante righe quante colonne) e scatole rettangolari nell'altro senso (più righe che colonne).
Supponiamo di dover "riempire" col valore (intero) 1 tutti i comparti di una matrice (intera): si può ottenere lo scopo con il seguente frammento di (pseudo)programma.

```
1   proc EsempioM;
2           var A(1: 10,1:10), I, J integer;
3           …
4           for I from 1 to 10 do
5                   for J from 1 to 10 do
6                           A(I,J) := 1;
7                   endfor;
8           endfor;
9           …
1   endproc;
0
```

Si osservi che sono presenti due statement **for**: ciascuno si "chiude" con l'**endfor** corrispondente come chiaramente mostrato dalla indentazione (quello "in verticale"): si dice che gli statement **for** sono *annidati* (cioè sono "uno dentro l'altro"). Si consideri lo *statement* di assegnazione di riga 6: nell'eseguirlo vengono di volta in volta costruiti i nomi degli scomparti della scatola A nell'ordine sopra visto (per righe).

Esercizio *in itinere* (obbligatorio!). Riscrivere il frammento precedente costruendo i nomi dei comparti di A (ordinati) per *colonna* (cioè: A(1,1), A(2,1), A(3,1), …, A(1,2), A(2,2), …). (Questo è "più facile" se prima si "esegue" la procedura EsempioM.)

Nelle matrici quadrate (scatole con lo stesso numero di righe e colonne) sono particolarmente importanti i comparti sulla diagonale "principale", cioè (nell'esempio appena visto) A(1,1), A(2,2), A(3,3), … A(10,10), o in generale quei comparti che hanno l'indice di riga uguale a quello di colonna. Il seguente frammento (che si può immaginare al posto dei puntini della riga 9 di EsempioM) pone a 0 tali comparti.

```
1           …
2           for I from 1 to 10 do
3                   A(I,I) := 0;
4           endfor;
5           …
```

N.B. I comparti di un vettore si chiamano componenti e il loro numero lunghezza (o dimensione) del vettore; i comparti di una *matrice* si dicono anche *elementi* e si dice

dimensione (o, talvolta, *taglia*) della matrice il numero di elementi *indicati come prodotto del numero di righe per quello delle colonne*; la matrice del frammento precedente ha dimensione 10×10.

§ 1.7 Esercizi: saper leggere (3)

ESERCIZIO 1

Si consideri il seguente pseudoprogramma.

```
1   proc EsempioL;
2           var A, N, S integer;
3           var Alfa boolean;
4           Alfa := true;
5           S  := 0;
6           N := 0;
7           while Alfa do
8                   read A;
9                   if A>0
1                           then
0                                   S := S+A;
1                                   N := N+1;
1                           else Alfa := false;
1                   endif;
2           endwhile;
1           write S, N;
3   endproc;
1
4
1
5
1
6
1
7
```

Se la sequenza di valori acquisiti per la variabile A è la seguente:

1, 6, 21, 3, 0, 5, -6

quali sono i valori di S e N resi disponibili al termine della esecuzione?

S	
N	

Se la sequenza di valori *acquisibili* per la variabile A è la seguente:

1, 6, 21, 3, 0, 5, 6

come “si comporta” la procedura?

ESERCIZIO 2

Si consideri il seguente pseudoprogramma.

```
1   proc EsempioM;
2       var A(1: 10,1:10), N, M, I, J, Imax, Jmax integer;
3       read M, N;
4       if (M>10) OR (N>10)
5           then  write  'dati errati', M, N;
6                 stop;
7       endif;
8       for I from 1 to N do
9           for J from 1 to M do
10          read A(I,J);
11          endfor;
12      endfor;
13      Max := A(1,1);
14      Imax := 1;
15      Jmax := 1;
16      for I from 1 to N do
17          for J from 1 to M do
18              if A(I,J) > Max
19                  then  Max := A(I,J);
20                        Imax := I;
21                        Jmax := J;
22              endif;
23          endfor;
24      endfor;
25      Imax := Imax−1;
26      Jmax := Jmax+1;
27      write  Imax, Jmax
28  endproc;
```

Se per M e N vengono letti 4 e 5, se per gli elementi di A vengono acquisiti i valori:

1, 0, 2, 0, 3, 0, 4, 0, 50, 0, 8, 9, 2, 34, 65, 4, 8, 66, 99, 0

quali sono i valori resi disponibili?

Imax	
Jmax	

§ 1.8 Procedura e processo
(ovvero ancora sul concetto di "esecuzione")

Lo pseudoprogramma è una descrizione del procedimento risolutivo (di una classe) di problemi; la *esecuzione* (manuale), come si è visto, è la realizzazione (da parte di una *persona*) delle attività descritte. È particolarmente utile, per approfondire l'argomento, il concetto di *processo*: una definizione è mostrata nella Figura 6.

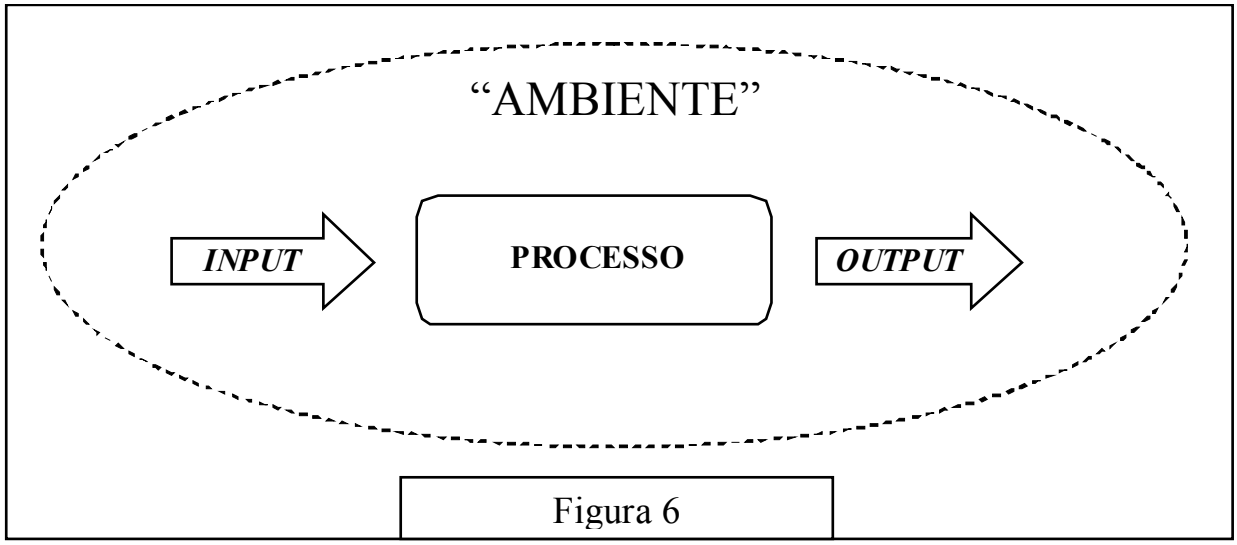


Figura 6

L'usuale definizione è la seguente "*Un processo è un insieme (di risorse e) di attività tra loro interconnesse che trasformano elementi in ingresso (o input) in elementi in uscita (o output)*". Per quello che ci riguarda in questa sede:

1) trascuriamo (per il momento, almeno) le risorse,
2) le attività del processo riguardano le scatole/variabili (e sono quelle descritte dallo pseudo programma),
3) gli "elementi" (rispettivamente in *input* e *output*) sono "dati", cioè valori per variabili (rispettivamente acquisiti e resi disponibili)[#].

Si può anche dire, quindi, che un programma è la descrizione delle attività di un processo: svolgere quelle attività costituisce la sua esecuzione.

Una caratteristica che un processo *può* avere è la *terminazione*, cioè tutte le sue attività terminano *sicuramente* dopo un tempo finito, *qualunque* siano i dati di *input*.

N.B. La terminazione non è una proprietà *sempre* desiderabile (od opportuna) per i processi associati ai programmi: tuttavia in questo contesto verrà sempre richiesta.

[#] Da adesso in poi i valori (che saranno) acquisiti vengono detti anche *dati di input* e quelli messi a disposizione *dati di output*. Si noti che in generale gli "elementi" possono essere altre "cose": per esempio azioni compiute da particolari meccanismi detti sensori o attuatori presenti, come risorse, all'interno del processo.

Una convenzione adottata in questo manuale è che un processo associato a uno pseudoprogramma termina in tre casi (due sono già visti):

1) non ci sono più attività da eseguire,
2) viene eseguita l'attività associata allo statement **stop**,
3) viene eseguita l'attività di acquisizione di dati di *input* che *non* sono *disponibili*,
4) non è possibile eseguire una attività specificata in uno *statement*.

Come esempio del terzo punto (da usare con estrema parsimonia) si veda l'esercizio 1 del paragrafo 1.7.

Un caso del quarto punto è l'uso di scatole non "riempite", cioè di variabili a cui non è stato assegnato un valore, per esempio

write A;

se la scatola A è "vuota".
Altri esempi sono gli *statement* di assegnazione che usano, a destra del simbolo ':=' espressioni non valutabili:
A**B se B è una variabile **boolean**,
(A+B) * (D−F) se le variabili A, B, C sono intere e la variabile F è di tipo **string** (e contiene, per esempio 'dati errati'),
(A+B) * (D−F) se qualcuna delle variabili A, B, C, D non ha valore.
Un ultimo esempio (in realtà già visto) è la costruzione di nomi illeciti per le variabili con struttura (in qualunque statement compaiono):
A(J) con J che contiene 101 e la variabile A è stata dichiarata di 100 comparti,
B(RA,RB) con almeno una delle variabili RA, RB con valore fuori dei limiti stabilti nella dichiarazione di B
Si noti che in alcuni degli esempi appena visti ci si può accorgere dell'errore con un accurato esame (del testo) della procedura; altri sono rilevati solo durante l'esecuzione (e con particolari dati di *input*).

È bene notare che, in generale, le attività di un processo non necessariamente si svolgono in maniera sequenziale; questa è una caratteristica dovuta allo pseudolinguaggio qui descritto: altri linguaggi (di programmazione) possono avere costrutti linguistici che descrivono lo svolgimento di attività "parallele".

Inoltre un processo si deve immaginare immerso in un "ambiente" che:

- fornisce le "risorse" per lo svolgimento delle attività,
- spravvede la creazione (e la distruzione) del processo,
- fornisce i dati di *input* (altrimenti il processo termina),
- "assorbe" (ed utilizza) i dati di *output*.

N.B. Nel caso descritto da questo manuale (pseudolinguaggio ed esecuzione manuale, cioè compiuta da una persona) i concetti di processo e "ambiente" sono (*relativamen-*

te) poco importanti: così non è nel caso di un (vero) linguaggio di programmazione che deve essere eseguito da (risorse presenti in) un *computer*: in questo caso l'ambiente prende il nome di *sistema operativo* e permette la "creazione", la "distruzione" e la "convivenza" di più processi contemporaneamente (oltre ad essere indispensabile ad ogni uso pratico di un *computer*).

N.B. In Informatica spesso la parola 'ambiente' ha significati diversi.

§ 1.9 Due problemi classici (e qualche soluzione)

In questo sottoparagrafo vengono descritti (qualcuno) dei metodi per risolvere il problema dell'ordinamento (o *sort*) e il problema della fusione (o *merge*).

Sort (o ordinamento). *Dato un vettore* (con componenti numeriche, cioè con comparti che contengono numeri) *ordinare (in maniera crescente) le sue componenti* (cioè scambiare i valori, contenuti nei vari comparti, in modo che siano disposti in ordine *crescente* con l'indice della componente).

Il seguente frammento è la parte iniziale e finale di (tutti) gli pseudoprogrammi di ordinamento considerati in questo sottoparagrafo.

```
1   proc Sort;
2   [semplice procedura per ordinare le componenti di un vettore]
3           var A(1: 100), B, N, I, J, K integer;
4
5           [acquisire la lunghezza del vettore nella variabile N]
6           read N;
7
8           [controllare che la lunghezza sia coerente con la dichiarazione]
9           if N<2 OR N>100
1                   then  write  'dato errato', N;
0                         stop;
1           endif;
1
1           [acquisire il numero previsto di componenti del vettore]
2           for I from 1 to N do
1                   read A(I);
3           endfor;
1           …
4           [ordinare le componenti di A, dalla prima alla N-esima]
1
5           [rendere disponibili le componenti ordinate del vettore]
1           for I from 1 to N do
6                   write A(I);
```

```
1       endfor;
7   endproc;
1
8
1
9
2
0
2
1
2
2
2
3
2
4
2
5
```

"Scheletro" di una procedura per il *sort*

Il vettore da ordinare viene dichiarato di dimensione 100: il programma risolve i problemi che trattano (vettori con) un numero minore o uguale di componenti. Poi si acquisisce la lunghezza del caso da trattare (e ci si assicura che sia minore di N e almeno 2: per N eguale a 1 l'ordinamento è banale!). Successivamente si acquisiscono (ad uno ad uno) i valori (per le componenti del vettore) da ordinare. Dopo l'ordinamento si rendono disponibili all'esterno le componenti di A.

Adesso bisogna pensare come fare (e scriverlo in pseudolinguaggio: cioè mettere gli *statement* opportuni fra i due commenti di riga 19 e 21).

SelectionSort. Una delle maniere più semplici, per ordinare le componenti di un vettore, è osservare che nella prima componente A(1), dopo l'ordinamento, dovrà essere contenuto il valore più piccolo: quindi occorre cercare questo valore e, supponendo di trovarlo nello K-esimo comparto, scambiare i contenuti delle componenti A(1) e A(K). Si continua così per la seconda componente e le successive.
Si consideri il seguente frammento (che può essere pensato posto successivamente alla riga 17 nel precedente).

```
1     …
2     [fissiamo l'attenzione sulla prima componente]
3     J := 1;
4     K := J;  [ipotesi che la prima componente sia anche la più piccola]
5     for I from K+1 to N do
6           [correggere l'ipotesi, se del caso]
7           if A(K) <A(J)
8                 then
9                       K := I;  [correzione]
10          endif;
11    endfor;
12    [adesso (il valore di) K è l'indice della componente più piccola]
13
14    [scambiare le componenti A(K) e A(J): B è la scatola di appoggio]
15    B := A(K);
16    A(K) := A(J);
17    A(J) := B;
      …
```

Frammento 1

N.B. Si è introdotta una nuova notazione, nello statement **for**: ciò che segue from o to può essere (oltre a una costante o una variabile) anche una espressione. Ormai il lettore dovrebbe trovare "naturali" queste (piccole) variazioni di scrittura ed attribuire loro il corretto significato (cioè dovrebbe "saperle eseguire").

Esercizio *in itinere* (obbligatorio!). Eseguire il frammento precedente e controllare che, dopo l'esecuzione, la componente A(1) contiene effettivamente il valore più piccolo rispetto alle *successive* (cioè quelle che sono a destra).

Nel frammento precedente solo lo statement di riga 3 "dice" che l'operazione descritta si deve svolgere per "sistemare" la *prima* componente: in realtà le righe da 4 a 16 riguardano la generica componente J-esima: quindi si possono "ripetere" (con un costrutto **for** … **endfor**) per sistemare tutte le componenti da J = 1 fino alla *penultima* componente: infatti se ciascuna di queste è più piccola delle successive, allora (tutto) il vettore è ordinato (in maniera crescente). Si osservi il seguente frammento,

```
1       …
2       [fissiamo via via l'attenzione sulla componente 1, 2 , …, N-1]
3       for J from 1 to N-1 do
4           [ipotesi che la componente J-esima sia la più piccola]
5           K := J;
6           for I from K+1 to N do
7           [correggere l'ipotesi, se del caso]
8               if A(K) <A(J)
9                   then
10                      K := I;  [correzione]
11              endif;
12          endfor;
13          [adesso (il valore di) K è l'indice della componente più piccola
14          a destra della componente di indice J]
15
16          [scambiare le componenti A(K) e A(J)]
17          B := A(K);
18          A(K) := A(J);
19          A(J) := B;
        endfor;
        …
```

Frammento 2

che realizza l'obbiettivo appena descritto; questo frammento inserito fra i due commenti di riga 19 e 21 risolve il problema (di scrivere una procedura di *sort* col metodo *selectionsort*).

Esercizi *in itinere* (obbligatori!). Scrivere la procedura Sort completa (composta dallo Scheletro e da Frammento 2) ed eseguirla nei seguenti casi:

- il valore acquisito per N è 7 e quelli per le componenti sono 11, 7, 3, 13, 32, 5, 8;
- il valore acquisito per N è 7 e quelli per le componenti sono 3, 5, 7, 8, 11, 13, 32;
- il valore acquisito per N è 7 e quelli per le componenti sono 32, 13, 11, 8, 7, 5, 3;

Inoltre:

- Quanti confronti specificati nel predicato di riga 8 di Frammento 2 vengono eseguiti nei tre casi?
- Quanti confronti specificati nel predicato di riga 10 di Frammento 2 vengono eseguiti nei tre casi?
- Quante volte lo scambio specificato dalla riga 17 alla 19 di Frammento 2 viene eseguito nei tre casi?
- Se non viene mai eseguita la assegnazione della riga 10 di Frammento 2, nell'esecuzione del costrutto ripetitivo **for** da riga 6 a riga 12, lo scambio specificato

dalla riga 17 alla 19 è inutile: pensare a come si potrebbe modificare lo pseudo-programma per evitare di eseguirlo.

BubbleSort. Questo metodo, per ordinare le componenti di un vettore, utilizza (solamente) lo scambio di una componente del vettore con la successiva, cioè lo scambio di contenuto di due comparti *adiacenti*. Può essere descritto come una successione di "passate": una "passata" è una successione di confronti tra componenti adiacenti del vettore, ciascun confronto è seguito da uno scambio tra i contenuti dei comparti se questi non sono nell'ordine "giusto": si scambiano, cioè, quando il primo è maggiore del secondo. Il seguente frammento illustra una "passata".

```
1     …
2     for I from 1 to N - 1 do
3         if A(I) >A(I+1)
4             then
5                 B := A(I);
6                 A(I) := A(I+1);
7                 A(I+1) := B;
8         endif;
9     endfor;
10    …
```

Frammento 3

N.B. Negli *statement* di rigo 3 e rigo 6 è stata usata una *espressione* per costruire il nome di un comparto (cioè I+1); il "significato" è ovvio: si indica il contenuto del comparto successivo allo I-esimo (cioè, appunto lo (I+1)-esimo).

Intuitivamente dopo una "passata" il vettore è "più ordinato di prima": comunque ci si può chiedere quante "passate" sono necessarie per ordinare (completamente) il vettore? Per rispondere basta osservare che (con la prima "passata") il valore più alto delle componenti (dovunque si trovi) è stato via via scambiato di posto fino ad occupare l'ultimo comparto, che quindi è sistemato: la "passata" successiva, quindi, può essere più corta (cioè non riguardare l'ultima componente) e così via.

```
1       …
2       for J from 1 to N-1 do              [ripetizione della "passata"]
3           for I from 1 to N - J do        ["passata", sempre più breve]
4               if A(I) >A(I+1)             [confronto di comparti successive]
5                   then                    [scambio]
6                       B := A(I);
7                       A(I) := A(I+1);
8                       A(I+1) := B;
9               endif;
10          endfor;
11      endfor;
12      …
```

Frammento 4

Il Frammento 4 mostra la scrittura di quanto detto.

Esercizi *in itinere* (obbligatori!). Scrivere la procedura Sort completa (composta dallo Scheletro e da Frammento 4) ed eseguirla nei seguenti casi:
- il valore acquisito per N è 7 e quelli per le componenti sono 11, 7, 3, 13, 32, 5, 8;
- il valore acquisito per N è 7 e quelli per le componenti sono 3, 5, 7, 8, 11, 13, 32;
- il valore acquisito per N è 7 e quelli per le componenti sono 32, 13, 11, 8, 7, 5, 3;

Inoltre:
- Quanti confronti specificati nel predicato di riga 4 di Frammento 4 vengono eseguiti nei tre casi?
- Quante volte lo scambio specificato dalla riga 6 alla 8 di Frammento 4 viene eseguito nei tre casi?
- Se in una "passata" non viene *mai* eseguito lo scambio da riga 6 a riga 8, allora non verrà più eseguito neppure nelle passate successive: perché?

Sfruttando l'osservazione dell'ultimo esercizio si può "migliorare" il Frammento 4, aggiungendo una variabile logica (Disordinato) e trasformando il ciclo "esterno" **for** in un ciclo **while**, per
- incrementare l'indice che conta le "passate" *e*
- tener conto del fatto che durante la passata precedente non sono stati eseguiti scambi (cioè il vettore non è più disordinato).

Per fare questo si usa un predicato "composto" nello *statement* **while**, come mostrato nel successivo Frammento 5.

```
1   …
2   var Disordinato boolean;
3   …
4   Disordinato := true;                    [si suppone il vettore disordinato]
5   J :=1;
6   while (J ≤ N-1) AND Disordinato do      [ripetizione della "passata":]
7                                           [solo se il vettore è disordinato]
8       Disordinato := false;               [si ipotizza che il tratto da esami-]
9                                           [nare nella passata sia ordinato]
10      for I from 1 to N - J do            ["passata", sempre più breve]
11          if A(I) >A(I+1)                 [confronto di comparti successive]
12              then                        [scambio]
13                  B := A(I);
14                  A(I) := A(I+1);
15                  A(I+1) := B;
16                  [correzione dell'ipotesi:]
17                  [il tratto esaminato nella passata è disordinato]
18                  Disordinato := true;
19          endif;
20      endfor;
21  J := J+1;
22  endwhile;
    …
```

Frammento 5

Esercizi *in itinere* (obbligatori!). Scrivere la procedura Sort completa (composta dallo Scheletro e da Frammento 5: attenzione alla dichiarazione di una variabile in più) ed eseguirla nei seguenti casi:

- il valore acquisito per N è 7 e quelli per le componenti sono 11, 7, 3, 13, 32, 5, 8;
- il valore acquisito per N è 7 e quelli per le componenti sono 3, 5, 7, 8, 11, 13, 32;
- il valore acquisito per N è 7 e quelli per le componenti sono 32, 13, 11, 8, 7, 5, 3;

Inoltre:

- Quanti confronti specificati nel predicato di riga 11 di Frammento 5 vengono eseguiti nei tre casi?
- Quante volte lo scambio specificato dalla riga 13 alla 15 di Frammento 5 viene eseguito nei tre casi?

Merge (o fusione). *Dati due vettori, ciascuno con le componenti ordinate (in maniera crescente), fonderli* (cioè costruire un nuovo vettore che abbia tutte le componenti dei due dati, disposte in ordine crescente).

Prima ancora di pensare “come” risolvere il problema si può scrivere lo “scheletro” dello pseudoprogramma che risolve il problema.
Si consideri il seguente frammento.

```
1   proc Merge;
2   [semplice procedura per fondere le componenti di due vettori già ordinati]
3           var A(1: 100), B(1: 100), C(1: 200) integer;
4           var LA, LB, LC, JA, JB, JC integer;
5
6           [acquisire la lunghezza dei vettori A e B nelle variabile LA, LB]
7           read LA, LB;
8
9           [controllare che la lunghezza sia coerente con la dichiarazione]
1           if LA<1 OR LA>100 OR LB<1 OR LB>100
0                   then  write  ‘dati errati’, LA, LB;
1                         stop;
1           endif;
1
2           [acquisire il numero previsto di componenti del vettore A]
1           for JA from 1 to LB do
3                   read A(JA);
1           endfor;
4           [acquisire il numero previsto di componenti del vettore B]
1           for JB from 1 to LB do
5                   read B(JB);
1           endfor;
6
1           [verificare l’ipotesi di ordinamento delle componenti del vettore A]
7           for JA from 1 to LA−1 do
1                   if A(JA)> A(JA+1)
8                         then  write  ‘primo vettore non ordinato’;
1                         stop;
9                   endif;
2           endfor;
0           [verificare l’ ipotesi di ordinamento delle componenti del vettore B]
2           for JB from 1 to LB−1 do
1                   if B(JB)> B(JB+1)
2                         then  write  ‘secondo vettore non ordinato’;
2                         stop;
2                   endif;
3           endfor;
2           …
4           [fondere le componenti di A e B in C, dalla prima alla N-esima]
2           …
```

```
5       [rendere disponibili le componenti ordinate del vettore C]
2       for JC from 1 to LC do
6           write C(JC);
2       endfor;
7   endproc;
2
8
2
9
3
0
3
1
3
2
3
3
3
4
3
5
3
6
3
7
3
8
3
9
4
0
4
1
4
2
4
3
4
4
4
5
```

"Scheletro" di una procedura per il *merge*

In esso:

- vengono dichiarati i vettori A e B (che conterranno le componenti da fondere), il vettore C (che sarà il *merge* di A e B); vengono dichiarate inoltre LA, LB LC (scatole che conterranno le lunghezze dei vettori A, B, C) insieme con JA, JB, JC (scatole che serviranno da indice per i vettori A, B, C);
- viene acquisita la lunghezza dei vettori e si controlla che siano minori di quelle (massime) previste dalla dichiarazione;
- vengono acquisite le componenti di A e B;
- viene controllato che, effettivamente, le componenti dei due vettori sono ordinate (*occorre sempre controllare che i dati soddisfino le ipotesi!*);
- viene lasciato uno spazio "vuoto" per risolvere il problema (costruire C);
- vengono rese disponibili le componenti di C.

N.B. Il precedente elenco di punti (scritto in linguaggio naturale) *è* (sostanzialmente) *equivalente* allo "scheletro" (scritto in pseudolinguaggio). È *assolutamente importante* rendersi conto che la differenza tra i due sta nel fatto che per "eseguire" lo pseudo-programma sono necessarie (molto!) meno "nozioni" che per "eseguire" l'elenco: le due descrizioni sono, *in realtà*, equivalenti *solo* per una *persona* (di media cultura) con (tutto) il suo (complesso) bagaglio di conoscenze.
Si osservi la Figura 7.

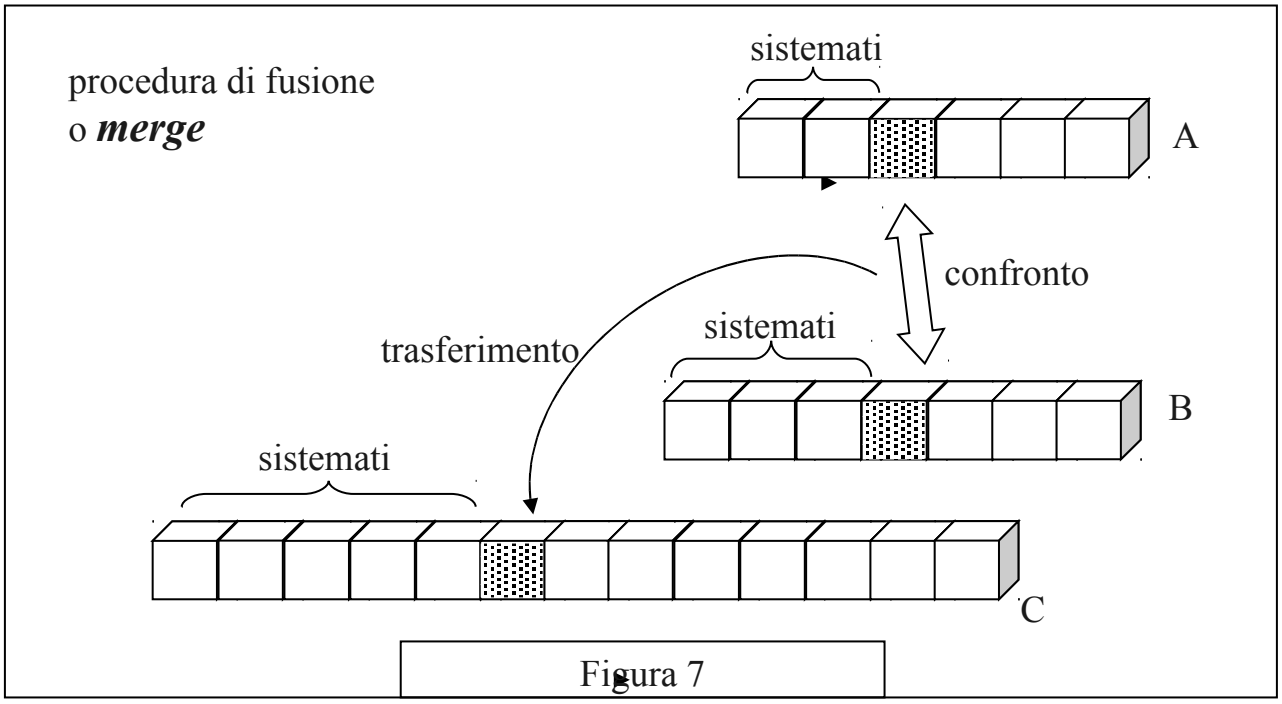


Figura 7

Si può descrivere (in linguaggio naturale) il procedimento di fusione, dicendo che si fissa l'attenzione su gli elementi di A e B non ancora sistemati (all'inizio, ovviamente sui primi elementi): il più piccolo dei due (o quello di A, se sono uguali) viene "sistemato", cioè copiato nel primo comparto libero (che non ha ancora valore) di C. Successivamente si continua (considerando sempre i primi elementi di A e B non ancora sistemati). Naturalmente, quando uno dei vettori di partenza è "esaurito" (cioè tutti i

suoi elementi sono stati “sistemati”), si finisce copiando i rimanenti elementi dell’altro.

Si osservi il Frammento 6, che dice la “stessa cosa” in pseudolinguaggio.

```
1     …
2     JA :=1;
3     JB :=1;
4     LC := LA+LB;                          [calcolare la lunghezza di C]
5
6     for JC from 1 to LC do                [esaminare ogni elemento di C]
7           if JA > LA AND JB ≤ LB          [A è “esaurito” ]
8                 then
9                       C(JC) := B(JB);     [trasferire la componente di B]
10                      JB :=JB+1;          [e passare a quella successiva]
11          endif;
12          if JA ≤ LA AND JB > LB          [B è “esaurito” ]
13                then
14                      C(JC) := A(JA);     [trasferire la componente di A]
15                      JA :=JA+1;          [e passare a quella successiva]
16          endif;
17          if JA ≤ LA AND JB ≤ LB          [né A né B sono “esauriti” ]
18                then
19                      if A(JA ) ≤ B(JB)   [scegliere tra A e B]
20                            then          [trasferire la componente di A]
21
22                                  C(JC) := A(JA);
23                                  JA :=JA+1;
24                            else          [trasferire la componente di B]
25                                  C(JC) := B(JB);
26                                  JB :=JB+1;
27                      endif;
28          endif;
29    endfor;
      …
```

Frammento 6

Nello pseudolinguaggio *conviene* esaminare successivamente ogni elemento di C e trasferirvi l’opportuno elemento di A o B (distinguendo il caso “normale”, da quelli in cui uno dei due vettori è “esaurito).
Si faccia attenzione ai predicati negli statement **if** di rigo 7, 12 e 17. Si noti inoltre che l’indice JC (del vettore C), è incrementato “automaticamente” (*statement* **for** di

rigo 6, cui corrisponde l'**endfor** di rigo 28), mentre gli indici JA (del vettore A) e JB (del vettore B) sono incrementati esplicitamente, quando è necessario.

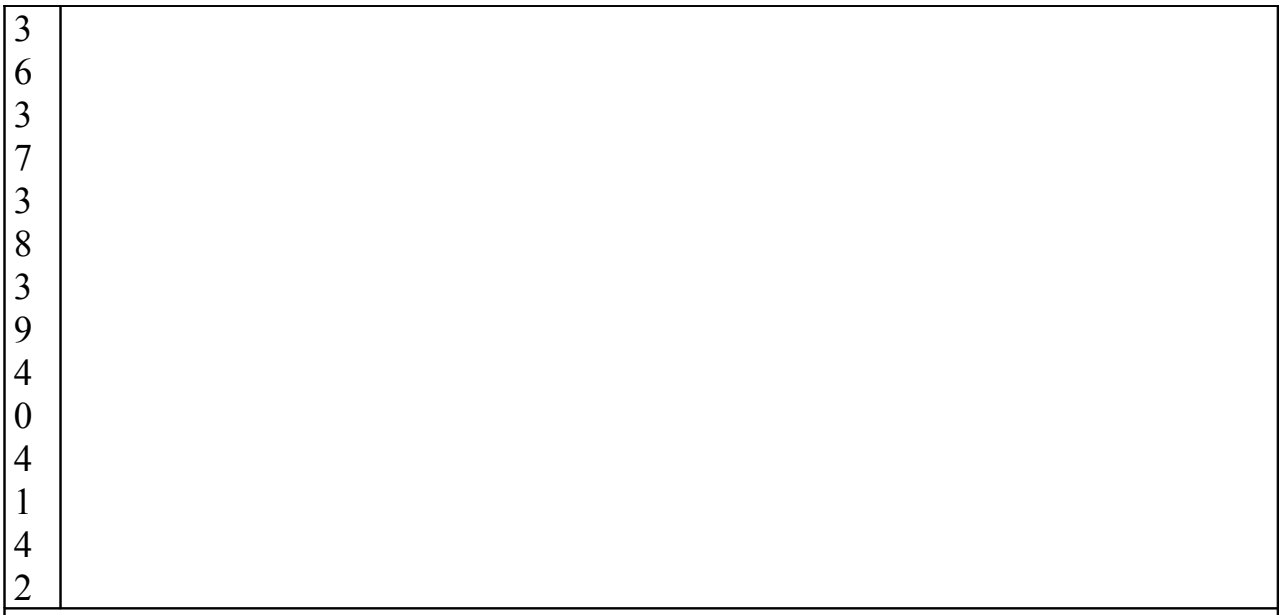
Esercizi *in itinere* (obbligatori!). Scrivere la procedura Merge completa (composta dallo Scheletro e da Frammento 6 opportunamente inserito).
Eseguire la procedura una prima volta con:
– il valore 6 per la lunghezza di A e come valori delle componenti: 3, 5, 7, 9, 11, 13;
– il valore 6 per la lunghezza di B e come valori delle componenti: 1, 2 ,4, 4, 4, 10;
Eseguire la procedura una seconda volta con:
– il valore 9 per la lunghezza di A e come valori delle componenti: 3, 5, 7, 9, 11, 13, 15, 17, 18;
– il valore 7 per la lunghezza di B e come valori delle componenti: 1, 2, 4, 4, 4, 10, 13;
Nel secondo caso, quante volte viene eseguito il "ramo" **then** di rigo 8?
Dopo quale esecuzione di quale *statement* è vero il predicato (JA > LA) **AND** (JB > LB)? (Si osservi che non compare nello pseudo programma.)

Merge in place. *In un vettore, le prime* p *componenti sono ordinate* (*in maniera crescente*)*, le successive* s *componenti sono pure ordinate* (*in maniera crescente*); *ordinare tutte le componenti* (cioè fondere le prime *p* con le successive *s* senza *spostarle* in un nuovo vettore).

Prima ancora di pensare "come" risolvere il problema si può scrivere lo "scheletro" dello pseudoprogramma che risolve il problema.
Si consideri il seguente frammento.

```
1   proc MergeInPlace;
2   [Semplice procedura per fondere due segmenti successivi ordinati di un vettore]
3         var A(1: 200), B integer;
4         var P, S, LA, JS, J integer;
5
6         [acquisire la lunghezza dei due segmenti di vettore nelle variabili P e S]
7         read P, S;
8
9         [controllare che la lunghezza totale sia coerente con la dichiarazione]
1         LA :=P+S;
0         if LA > 200
1               then  write  'dati errati', P, S;
1                     stop;
1         endif;
2
1         [acquisire il numero previsto di componenti del vettore A]
3
```

```
1       for J from 1 to LA do
4           read A(J);
1       endfor;
5
1       [verificare l'ordinamento delle prime P componenti del vettore A]
6       for J from 1 to P−1 do
1           if A(J)> A(J+1)
7               then  write  'primo segmento non ordinato';
1               stop;
8           endif;
1       endfor;
9       [verificare l'ordinamento delle seconde S componenti del vettore A]
2       for J from P+1 to P+S−1 do
0           if A(J)> B(J+1)
2               then  write  'secondo segmento non ordinato';
1               stop;
2           endif;
2       endfor;
2       …
3       [fondere le componenti di A senza spostarle]
2       …
4       [rendere disponibili le componenti ordinate del vettore A]
2       for J from 1 to LA do
5           write A(J);
2       endfor;
6   endproc;
2
7
2
8
2
9
3
0
3
1
3
2
3
3
3
4
3
5
```



“Scheletro” di una procedura per il *merge in place*

Valgono sostanzialmente le stesse osservazioni fatte a proposito dello scheletro precedente.

Si può descrivere (in linguaggio naturale) il procedimento di *merge in place*, dicendo che si esamina successivamente ogni elemento del secondo segmento e lo si sposta inserendolo “al posto giusto” (cioè rispettando l’ordine) nel primo segmento che così “aumenta” via via di lunghezza, mantenendosi ordinato; il secondo segmento diventa ogni volta più corto fino a “scomparire” (quando tutti i suoi elementi sono stati spostati e inseriti al loro posto). Per spostare un elemento lo si scambia via via con quello a sinistra se è (di questo) più piccolo.

Si osservi il Frammento 7, che dice la “stessa cosa” in pseudolinguaggio.

```
1     …
2     [esaminare e spostare ogni elemento del secondo segmento,]
3     [che comincia da P+1 e finisce a P+S (= LA)]
4     for JS from P+1 to LA do
5           J := JS;
6
7           [confrontare l’elemento esaminato del secondo segmento]
8           [con quelli a sinistra, e scambiarli, fin quando è il caso]
9           while J > 1 AND A(J) < A(J−1) do
10                B := A(J−1);
11                A(J−1) := A(J);     [scambio]
12                A(J) := B;
13                J := J−1;      [continuare con l’elemento precedente]
14          endwhile;
```

15 16	**endfor**; [il primo segmento ha lunghezza LA] …

Frammento 7

Si noti come il Frammento 7 (soluzione del problema del *merge in place*) sia più "semplice" (o almeno più corto) del Frammento 6 (soluzione del problema del *merge*).

Esercizi *in itinere* (obbligatori!). Scrivere la procedura MergeInPlace completa (composta dallo Scheletro e da Frammento 7 opportunamente inserito).
Eseguire la procedura una prima volta con:
– il valore 6 per la lunghezza del primo segmento,
– il valore 6 per la lunghezza del secondo segmento,
– valori delle componenti di A: 3, 5, 7, 9, 11, 13, 1, 2 ,4, 4, 4, 15;
Eseguire la procedura una seconda volta con:
– il valore 9 per la lunghezza del primo segmento,
– il valore 7 per la lunghezza del secondo segmento,
– valori delle componenti di A: 1, 5, 7, 9, 11, 13, 15, 17, 18, 2, 3, 4, 4, 4, 10, 13;
Esaminare il programma completo (Scheletro + Frammento 7): c'è una parte di *statement* che è superflua: quale?
Tolta la parte superflua la procedura risolve anche un altro problema: quale e come?

§ 1.10 Esercizi: saper leggere (4)

ESERCIZIO 1

Si consideri il seguente pseudoprogramma.

```
1   proc EsempioProdMM;
2       var A(1:5,1:5), B(1:5,1:5), C(1:5,1:5) integer;
3       var RA, CA, RB, CB, RC, CC, K integer;
4
5       [acquisire i valori per A e B]
6       for RA from 1 to 5 do
7           for CA from 1 to 5 do
8               read A(RA,CA);
9           endfor;
1       endfor;
0       for RB from 1 to 5 do
1           for CB from 1 to 5 do
1               read B(RB,CB);
1           endfor;
2       endfor;
1
3       [calcolo di C]
1       for RC from 1 to 5 do
4           for CC from 1 to 5 do
1               C(RC,CC) := 0;
5               for K from 1 to 5 do
1                   C(RC,CC) := C(RC,CC) + A(RC,K)*B(K,CC);
6               endfor;
1           endfor;
7       endfor;
1
8       [rendere disponibili i valori di C]
1       for RC from 1 to 5 do
9           for CC from 1 to 5 do
2               wirte A(RA,CA);
0           endfor;
2       endfor;
1   endproc;
2
2
2
3
2
```

4 2 5 2 6 2 7 2 8 2 9 3 0 3 1 3 2	

Se la sequenza di valori acquisiti per le variabili A e B sono

A: 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1

B: 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1,0

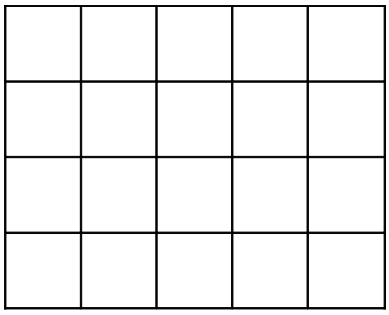
determinare quali sono i valori di C resi disponibili al termine della esecuzione, riempiendola seguente tabella che rappresenta, riga per riga, i valori della matrice.

Se la sequenza di valori acquisiti per le variabili A e B sono

A: 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1

B: 1, 2, 3, 4, 5, 5, 4, 3, 2, 1, 1, 2, 3, 4, 5, 5, 4, 3, 2, 1, 1, 2, 3, 4, 5

determinare quali sono i valori di C resi disponibili al termine della esecuzione, riempiendola seguente tabella che rappresenta, riga per riga, i valori della matrice.



ESERCIZIO 2

Si consideri il seguente pseudoprogramma.

```
1   proc EsempioProdMV;
2       var A(1:5,1:5), X(1:5), Y(1:5) integer;
3       var RA, CA, JX, JY, K integer;
4
5       [acquisire i valori per A e X]
6       for RA from 1 to 5 do
7           for CA from 1 to 5 do
8               read A(RA,CA);
9           endfor;
1       endfor;
0       for JX from 1 to 5 do
1           read X(JX);
1       endfor;
1
2       [calcolo di Y]
1       for JY from 1 to 5 do
3           Y(JY) := 0;
1           for JX from 1 to 5 do
4               Y(JY) := Y(JY) + A(JY,JX) ∗X(JX);
1           endfor;
5       endfor;
1
6       [rendere disponibili i valori di Y]
1       for JY from 1 to 5 do
7           wirte A(RA,CA);
1       endfor;
8   endproc;
1
9
2
0
2
1
2
2
2
3
2
4
2
```

5 2 6 2 7	

Se la sequenza di valori acquisiti per la variabile A e X sono

A: 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1

X: 0, 1, 0, 1, 1, 0

quali sono i valori di Y resi disponibili al termine della esecuzione?
Riempire la seguente tabella che rappresenta, in orizzontale, i valori del vettore Y.

Se la sequenza di valori acquisiti per la variabile A e X sono

A: 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1

X: 1, 2, 3, 4, 5

quali sono i valori di Y resi disponibili al termine della esecuzione?
Riempire la seguente tabella che rappresenta, in orizzontale, i valori del vettore Y.

§ 1.11 Complementi e commenti

Conversione tra tipi Nello pseudolinguaggio sono usati due tipi (di valore che possono avere le variabili) numerici: il tipo intero (**integer**) e il tipo decimale (**float**): relativamente (solo) a questi due tipi sono posibili: la *conversione* e le *espressioni miste*.

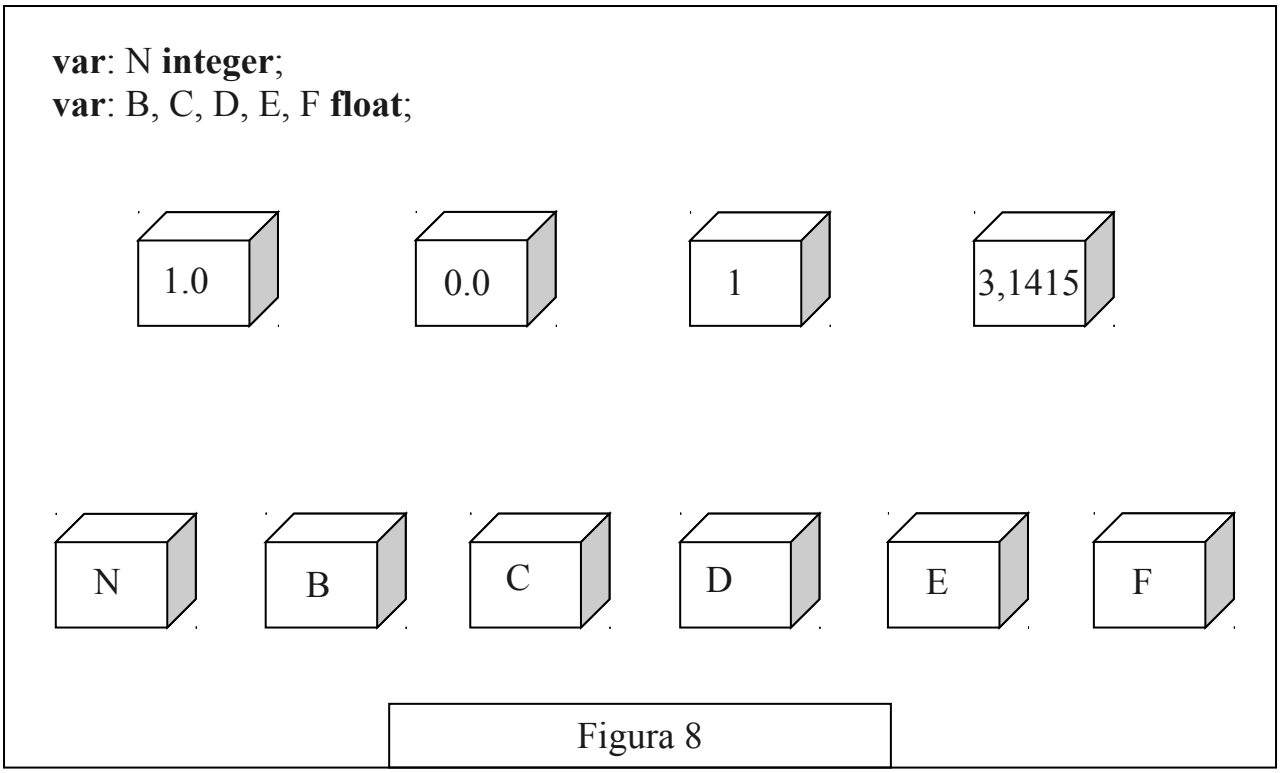


Figura 8

Nel seguito ci si riferisce alla figura 8.
Si parla di *conversione* quando un valore di un tipo viene trasferito in una scatola di tipo diverso. Sono possibili conversioni tra tipo **integer** e **float** e viceversa. Gli esempi seguenti illustrano i possibili casi.

N := 3,1415; [N contiene il valore intero 3: si ha *troncamento* dei decimali]

B := 1; [B contiene il valore decimale 1,0]

C := 3,1415 * 2,0; [C contiene il valore decimale 6.2830: non c'è conversione]
N := C; [N contiene il valore intero 6: si ha *troncamento* dei decimali]

Si parla di *espressioni miste* quando variabili dei due tipi compaiono nella stessa espressione. I linguaggi di programmazione hanno regole precise (e molto dettagliate) per la valutazione di espressioni miste: per lo pseudolinguaggio basti dire che *in linea di principio* una espressione mista ha valore decimale; per esempio se

E ha valore 4,2
D ha valore 3,0

N ha valore 5
allora l'espressione

E*2 + D**2 − N

ha valore 12,4 che rimane immutato se assegnato a una variabile **float**, ma viene troncato se assegnato a una variabile **integer**.

N.B. Nello pseudolinguaggio non sono ammesse conversioni o espressioni miste tra altri tipi.

Manipolazione di stringhe Nello pseudolinguaggio illustrato le stringhe sono formate da caratteri (stampabili): questi sono "ordinati" nel seguente modo:

- il carattere ' ' (cioè lo *spazio*, talvolta indicato anche con 'b') è il più piccolo;
- le cifre decimali (col loro ordine) sono "più piccole" dei caratteri maiuscoli;
- i caratteri maiuscoli (col loro ordine) sono "più piccoli" dei caratteri minuscoli.

Gli altri caratteri (segni di interpunzione, caratteri speciali, ecc.) sono sistemati tra i tre segmenti: cifre decimali, caratteri maiuscoli, caratteri minuscoli[¥]. Le stringhe formate da questi caratteri sono ordinate col comune ordine lessicografico e si possono confrontare con gli operatori =, >, <. Per esempio:

'a' < 'b'	vero
'a' > 'b'	falso
'alfa' < 'ALFA'	vero
'alfa' < ' ALFA'	falso
'dare' < 'datore'	vero
'bambolina' < 'bambolo'	vero

Non sono esprimibili (direttamente) operazioni tra caratteri o stringhe.

Linguaggi procedurali Lo pseudolinguaggio illustrato fa parte della classe dei linguaggi cosiddetti *procedurali*. L'idea di fondo (caratteristica di questi linguaggi) è che esista uno *stato*: questo è costituito dalla *memoria*, cioè l'insieme delle scatole (variabili o costanti) coinvolte e un particolare *contenuto* della memoria L'insieme di tutti i valori delle variabili); l'effetto (principale) dei vari *statement* del linguaggio è di cambiare lo stato (cioè creare scatole e variarne il contenuto). Un "programma" è la descrizione di una successione di cambiamenti di stato che raggiunge uno scopo "utile": per esempio il contenuto di certe scatole, alla fine (dell'esecuzione) del programma risolve un certo problema.
È bene abituarsi a pensare che un programma (in generale) non risolve un *singolo* problema, ma le istanze[¥¥] (dipendenti dai dati di *input*) di una classe di problemi "simili" (individuata dall'algoritmo o metodo di soluzione usato).

[¥] Per i dettagli si consulti Internet, per esempio la voce ASCII su Wikipedia.
[¥¥] Usualmente in informatica *istanza* significa un esempio, un caso particolare; è un anglicismo sul calco di *instance*.

Una osservazione importante (che introduce l'argomento del prossimo paragrafo) è che la "lettura" di un programma (nello pseudolinguaggio, come descritto fino ad adesso) molto "lungo" (per esempio 4 – 5 pagine) diventa "molto difficile" (e la sua esecuzione manuale molto laboriosa). La "ineluttabilità" dei programmi lunghi, d'altra parte, in qualche modo discende dalla tecnologia: l'esecutore *computer* compie centinaia di milioni di operazioni al secondo sulle scatole, quindi incoraggia la soluzione di problemi molto "complessi" (cioè che richiedono "molte" operazioni). Occorrono quindi nuovi strumenti linguistici per poter descrivere e padroneggiare facilmente procedimenti di calcolo molto complessi

Prima di chiudere questo paragrafo bisogna, altresì, notare come i *linguaggi procedurali* sono eminentemente adatti a descrivere "algoritmi di calcolo" anche molto complessi, ma quest'ultimi sono ben lungi dall'esaurire le possibilità dei programmi utili; in particolare si ricordano, a mo' di esempio, tre domini in cui il paradigma procedurale mette a disposizione costrutti linguistici insufficienti:

- le interfacce e le applicazioni multimediali,
- il maneggio di grandi quantità di dati (per esempio 10^{10}-10^{20} caratteri),
- l'uso "diretto" della rete.

§ 1.12 Esercizi: saper leggere (5)

ESERCIZIO 1

Si consideri il seguente pseudoprogramma.

```
1   proc EsempioArro;
2           var A(1:10) float;
3           var RA(1:10), TA(1:10), J integer;
4
5           [acquisire i valori per A]
6           for J from 1 to 10 do
7                   read A(J);
8           endfor;
9
1           [calcolo di RA e TA]
0           for J from 1 to 10 do
1                   TA(J) := A(J);
1                   RA(J) := A(J) + 0.5;
1           endfor;
2
1           [rendere disponibili i valori di TA e RA]
3           for J from 1 to 10 do
1                   wirte TA(J);
4           endfor;
1           for J from 1 to 10 do
5                   wirte RA(J);
1           endfor;
6   endproc;
1
7
1
8
1
9
2
0
2
1
2
2
2
3
```

Se la sequenza dei 10 valori acquisita per A è:
0,7 4,8 5,3 9,2 18,0 45,6 4,54 7,39 8,3456 1,987654

quali sono le sequenze rese disponibili per TA a RA?

TA										
RA										

TA	0	4	5	9	18	45	4	7	8	1
RA	1	5	5	9	18	46	5	7	8	2

ESERCIZIO 2

Si consideri il seguente pseudoprogramma.

```
1   proc Incognita6;
2       var A(1: 200), B string;
3       var LA, K, J integer;
4
5       [acquisire e controllare il numero previsto di componenti del vettore A]
6       read LA;
7       if LA > 200
8           then  write  'dato errato', LA;
9                 stop;
1       endif;
0
1       [acquisire il numero previsto di componenti del vettore A]
1       for J from 1 to LA do
1           read A(J);
2       endfor;
1
3       for K from 2 to LA do
1           J := K;
4           while J > 1 AND A(J) < A(J−1) do
1               B := A(J−1);
5               A(J−1) := A(J);     [scambio]
1               A(J) := B;
6               J := J−1;
1           endwhile;
7       endfor;
1
8
1       [rendere disponibili le componenti ordinate del vettore A]
9       for J from 1 to LA do
2           write A(J);
0       endfor;
2   endproc;
1
2
2
2
3
2
4
2
```

5 2 6 2 7 2 8 2 9 3 0 3 1	

Se viene acquisito il valore 7 per LA e se la sequenza dei valori acquisiti per A è:

“nel”,“mezzo”,“del”,“cammin”,“di”,“nostra”,“vita”

qual è la sequenza di valori messa a disposizione?

A							

A	“del”	“di”	“cammin”	“mezzo”	“nel”	“nostra”	“vita”

§ 2 (Pseudo)linguaggio come guida al ragionamento (ovvero costruire programmi “complessi”)

Scomposizione di problemi in sottoproblemi
Funzioni e Subroutine
Ricorsività
Quicksort e mergesort
Altro modo di dare significato ai programmi (ricorsivi)