Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

Pisa

# Introduzione al Linguaggio di Programmazione Pascal

*Daniela Musto*

Nota Interna B4-25
Giugno1990

# *Introduzione*

Questa dispensa offre una presentazione introduttiva e schematica del linguaggio di programmazione Pascal, destinata a coloro che hanno gia' acquisito una certa familiarita' con un linguaggio di programmazione *imperativo* (e.g., Basic, Fortan, Cobol, etc.) e che desiderano avvicinarsi al Pascal.

La dispensa illustra le principali caratteristiche del linguaggio Pascal, ed introduce la sintassi di un programma Pascal attraverso semplici esempi. Sono presentate le classi di istruzioni disponibili per la creazione dei programmi, ed i relativi *diagrammi di flusso* (detti anche *diagrammi a blocchi*). Viene evidenziata, in particolare, la struttura modulare dei programmi Pascal ottenuta attraverso la definizione di *procedure,* che permettono la strutturazione del programma in unita' logiche distinte che ne facilitano la stesura, la leggibilita' e la modificabilita'. Le *funzioni* e il ricco assortimento di *strutture dati* offerto dal linguaggio Pascal saranno oggetto di una sucessiva dispensa.

La dispensa e' costituita dai lucidi da me presentati durante il ciclo di seminari di introduzione al Pascal che ho tenuto nell'ambito del Corso di Perfezionamento in Applicazioni Informatiche dell'Universita' degli Studi di Pisa durante l'Anno Accademico 1989-90.

Pisa, Giugno 1990

## *Indice*

# Cenni Introduttivi

# Cenni Storici

Il linguaggio di programmazione Pascal prende il nome dal matematico e filosofo francese **Blaise Pascal** (1623-1662), a cui si deve, fra l'altro, l'invenzione di un calcolatrice meccanica capace di realizzare semplici operazioni aritmetiche.

Il linguaggio Pascal e' stato ideato in Svizzera dal Prof. **Niklaus Wirth**, alla fine degli anni 60. Wirth ha sviluppato il Pascal dal linguaggio di programmazione Algol 60, con lo scopo di creare uno strumento adatto per l'insegnamento della programmazione dei calcolatori e, nello stesso tempo, uno strumento efficiente per la scrittura di programmi di grosse dimensioni.

Il Pascal ha incontrato un grande successo, influenzando la maggior parte dei linguaggi di programmazione piu' recenti. E' implementato su un gran numero di calcolatori, sia di grandi che di piccole dimensioni.

# ◗ Caratteristiche

❒ ricco assortimento di **strutture dati**

❒ metodologia di progettazione **top-down**

(raffinamenti successivi)

❒ ambiente di **programmazione strutturata**

(costrutti per la strutturazione sistematica del programma in sottoparti significative)

- ♦ comprensibilita'
- ♦ maneggiabilita' ⟹ semplicita' & potenza
- ♦ modificabilita'

# Primo Programma

```
program costo (input, output);

{
  Scritto da: XXXXXXXXXXX XX/XX/XX
    scopo: calcolare un importo comprensivo di IVA
}
const
    aliquota= 0.19;          {aliquota IVA}
var
    prezzo: real;    {prezzo netto}
    imposta: real;   {imposta}
    totale: real;    {importo includente l'imposta}

begin
    write('indicare il prezzo: ');
        {segnalazione per l'utente del programma}
    readln(prezzo);
        {lettura del prezzo}
    imposta := aliquota * prezzo;
        {calcolo dell'imposta}
    totale := prezzo + imposta;
        {calcolo dell'importo totale}
    writeln ('L''importo totale e'': ', totale)
        {stampa dell'importo totale}
end.
```

# Analisi del Primo Programma

- ❒ intestazione
- ❒ dichiarazioni
  - ◆ costanti
  - ◆ variabili
- ❒ corpo del programma
  - ◆ istruzioni
    - • di scrittura
    - • di lettura
    - • di assegnamento
- ❒ commenti

# Elementi di Sintassi

- **parole riservate**
  (i.e., parole con un significato predefinito per il compilatore Pascal)

  program const var begin real
  write readln input output

- **identificatori**
  (i.e., parole definite dal programmatore; indicano nomi)

  costo
  aliquota
  prezzo imposta totale

✎ Il primo carattere e' una lettera;
gli altri caratteri sono una lettera o una cifra;
lunghezza max.127
(CompilatoreTurbo Pascal);

☞ Il compilatore non distingue le lettere maiuscole dalle lettere minuscole

- **costanti numeriche**

0.19

✎ Interi - senza punto decimale
(es.: 3, -7, ecc.);

Reali - con punto decimale
(es.: 3.5, -2.9, ecc.);

o in forma esponenziale
(es.:2.3E5, 5E-2, ecc.).

- **costanti di tipo carattere e di tipo stringa**

'Indicare il prezzo' ,

'L''importo totale e'':'

✎ Carattere - singolo carattere racchiuso tra apostrofi

Stringa - sequenza di caratteri racchiusa tra apostrofi(Comp. Turbo Pascal)

☞ Si noti che:
il singolo apostrofo e' interpretato come limite di stringa, mentre il doppio apostrofo viene interpretato come singolo apostrofo entro la stringa (e.g.: ' e'':')

- **dichiarazione di costanti:**

☞
```
const
    aliquota = 0.19;
```

✎ Forma: *identificatore* = *costante;*

la *costante* puo' essere una **costante numerica**, di **tipo carattere** o di **tipo stringa ...**

- **dichiarazione di variabili:**

☞
```
var
    prezzo : real;
```

✎ Forma: *identificatore*: *tipo;*
oppure
*lista di identificatori* : *tipo* ;

il *tipo* puo' essere **real**, **integer**, **char**, ... ;

il *tipo* puo' essere **string**[*lunghezza max*], dove *lunghezza max* e' compresa tra 1 e 255 (Compilatore Turbo Pascal);
il *tipo* puo' essere ..... ;

- **operatori aritmetici**

  +, *

  ✎ per reali: + (somma),
  - (sottrazione),
  * (moltiplicazione),
  / (divisione);

  per interi: + (somma),
  - (sottrazione),
  * (moltiplicazione),
  DIV(divisione con troncamento);
  MOD(modulo o resto)

- **espressione aritmetica**

  prezzo + imposta

  ✎ uno o piu' operandi combinati per mezzo di operatori aritmetici

- **istruzione di assegnamento**

  totale := prezzo + imposta

  ✎ Forma: *variabile* := *espressione*

- **istruzione di lettura**

readln(prezzo)

✎ Forma: readln(*lista*)

La *lista* e' formata da una o piu' variabili

- **istruzione di scrittura**

writeln('L''importo totale e'':', totale)

✎ Forma: write (*lista*)
writeln (*lista*)
writeln

La *lista* e' formata da uno o piu' elementi, che possono essere costanti, variabili, e/o espressioni.

- **commento**

{lettura del prezzo},
{stampa dell'importo totale}

✎ Forma: {*stringa*};

La *stringa* puo' essere formata da una qualsiasi sequenza di caratteri (inclusi i segni di interpunzione).

| *Programma* | *Stampa su video* |
|---|---|
| ..... | |
| **writeln**('Riga1'); | Riga1 |
| **writeln**('Riga2'); | Riga2 |
| | |
| **write**('Riga1'); | Riga1Riga2 |
| **writeln**('Riga2'); | |
| | |
| **write**('Riga1'); | Riga1 |
| **writeln**; | Riga2 |
| **writeln**('Riga2'); | |

```
N1:= 3;
N2 := 7;
SUM := N1+N2;
writeln('Il valore della somma e'': ', SUM);
```

Il valore della somma e':10

```
writeln(N1, '+', N2, '=', N1+N2);
```

3+7=10

```
writeln('N1', '+', 'N2', '=', N1+N2);
```

N1+N2 =10

## ❒ struttura del programma

```
program nomeprogramma (input, output);
const
    dichiarazione delle costanti
var
    dichiarazione delle variabili
begin
    istruzioni
end.
```

## ☞ Ricordarsi di:

1. Dichiarare tutte le variabili.

2. Terminare tutte le dichiarazioni con un ";".

3. Non usare "=" in un assegnamento, o ":=" in una dichiarazione di costanti.

4. Non lasciare spazi nella scrittura dell'operatore ":=".

5. Separare ciascuna istruzione dalla precedente con un ";".

6. Terminare il programma con "end" e ".".

## Secondo Programma

```
program area (input, output);

const
    finedati= 0;
var
    base: real;       {base del rettangolo (input)}
    altezza: real;    {altezza del rettangolo (input)}
    area: real;       {area del rettangolo (output)}
begin
    writeln('Questo programma calcola l'area ');
    writeln('dei rettangoli. Digitare la base');
    writeln('e l'altezza del rettangolo separati');
    writeln('da un bianco. ');
    writeln('Per terminare il processo, digitare ');
    writeln('il valore 0 per la base e un valore);
    writeln('qualsiasi per l'altezza. ');

    repeat
        writeln;
        writeln ('Digitare la base e l''altezza');
        readln(base, altezza);
        if base <> finedati then
            begin
                area := base * altezza;
                writeln('L''area e'' ', area:7:2)
            end
    until base = finedati
end.
```

# Elementi di Sintassi

- **parole riservate**
  (i.e., parole con un significato predefinito per il compilatore Pascal)

  repeat until if then

- **operatori relazionali**

  <> =

  ✎ = (uguale),
  <> (diverso),
  > (maggiore)
  >= (maggiore o uguale)
  < (minore)
  <= (minore o uguale)

- **espressione booleana semplice**

  base <> finedati

  ✎ espressione *op. relazionale* espressione

- **istruzione composta**

```
begin
    area := base * altezza;
    writeln('L''area e'' ', area:7:2)
end
```

✎ Forma: begin
*lista di istruzioni separate da ';'*
end

- **istruzione if**

```
if base <> finedati then
    begin
        area := base * altezza;
        writeln('L''area e'' ', area:7:2)
    end
```

✎ Forma: if *condizione* then
*istruzione* (α) (β)

if *condizione* then
*istruzione1* (α) (β)
else
*istruzione2* (α) (γ)

(α) l'*istruzione* puo' essere composta
(β) l'*istruzione* e' eseguita se la *condizione* e' vera
(γ) l'*istruzione* e' eseguita se la *condizione* e' falsa

## ☞ Ricordarsi che:

L' istruzione

```
if P > 0 then if P = 1 then writeln ('ok')
   else  writeln ('errore')
```

e' equivalente a

```
if P > 0 then
        begin
            if P = 1 then writeln ('ok')
                    else  writeln ('errore')
        end
```

Dunque "else" viene automaticamente associato dal compilatore all' "if" precedente piu' vicino . Per associare "else" al primo "if" e' necessario scrivere come segue:

```
if P > 0 then
          begin
            if P = 1 then writeln ('ok')
          end

        else  writeln ('errore')
```

## • istruzione repeat

```
repeat
    writeln;
    writeln ('Digitare la base e l''altezza');
    readln(base, altezza);
    if base <> finedati then
        begin
            area := base * altezza;
            writeln('L''area e'' ', area:7:2)
        end
until base = finedati
```

✎ Forma: repeat
*lista di istruzioni da ripetere*
until *condizione*

## • output formattato

```
writeln('L''area e'' ', area:7:2)
```

✎ Forma: *espressione* (α) *a.d.c.* (γ)
*espressione* (β) : *a.d.c.*: *a.d.c.*(γ)

(α) l'*espressione* e' di tipo intero o stringa
(β) l'*espressione* e' di tipo reale
(γ) l'*ampiezza di campo* (*a.d.c.*) e' una espressione intera

*Programma* *Stampa su video*

```
.....
    .
    .
    .
N:= 123;
writeln('Inizio', N, 'Fine');

                                    Inizio123Fine
writeln('Inizio', N:5, 'Fine');

                                    Inizio   123Fine



R:= 15.63;
writeln('Inizio', R, 'Fine');

                                    Inizio1.563000E01Fine

writeln('Inizio', R:7:2, 'Fine');

                                    Inizio   15.63Fine

    .
    .
    .
```

## ❒ struttura del programma

```
program nomeprogramma (input, output);
const
    dichiarazione delle costanti
var
    dichiarazione delle variabili
begin
    istruzioni
end.
```

## ☞ Ricordarsi che:

Le istruzioni in Pascal si dividono in :

1. **semplici** (assegnamento, write, read, ..)
2. **strutturate**
   - **2.1 composte**
   - **2.2 condizionali** (if, ...)
   - **2.3 ripetitive** (repeat, ..)

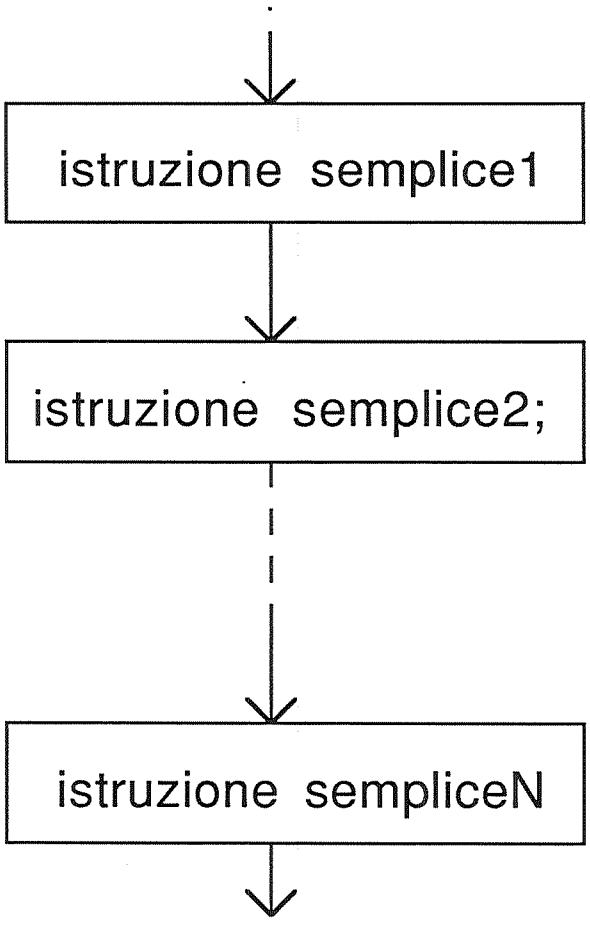
# Le Istruzioni

## ◗ Istruzioni Semplici

(e.g., assegnamento, write, read, etc.)

## ⇨ Diagramma di Flusso

```
begin
istruzione semplice1;
istruzione semplice2;
  .....
istruzione spliceN;
  .....
end.
```

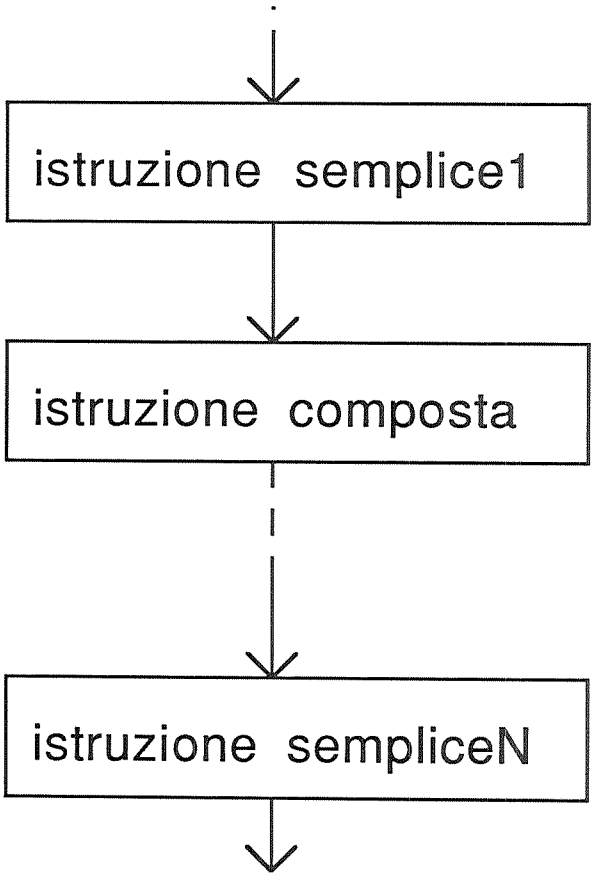


# ➧ Istruzioni Composte

(vedi pag.14)

## ⇨ Diagramma di Flusso

```
begin
istruzione semplice1;

istruzione composta;

  .........

istruzione spliceN;

  .........

end.
```



# Istruzioni Condizionali

## • istruzione if

✎ Forma: if *condizione* then
*istruzione* (α) (β)

if *condizione* then
*istruzione* A (α) (β)
else
*istruzione* B (α) (γ)

(α) l'*istruzione* puo' essere composta
(β) " " e' eseguita se la *condizione* e' vera
(γ) " " e' eseguita se la *condizione* e' falsa

## • istruzione case

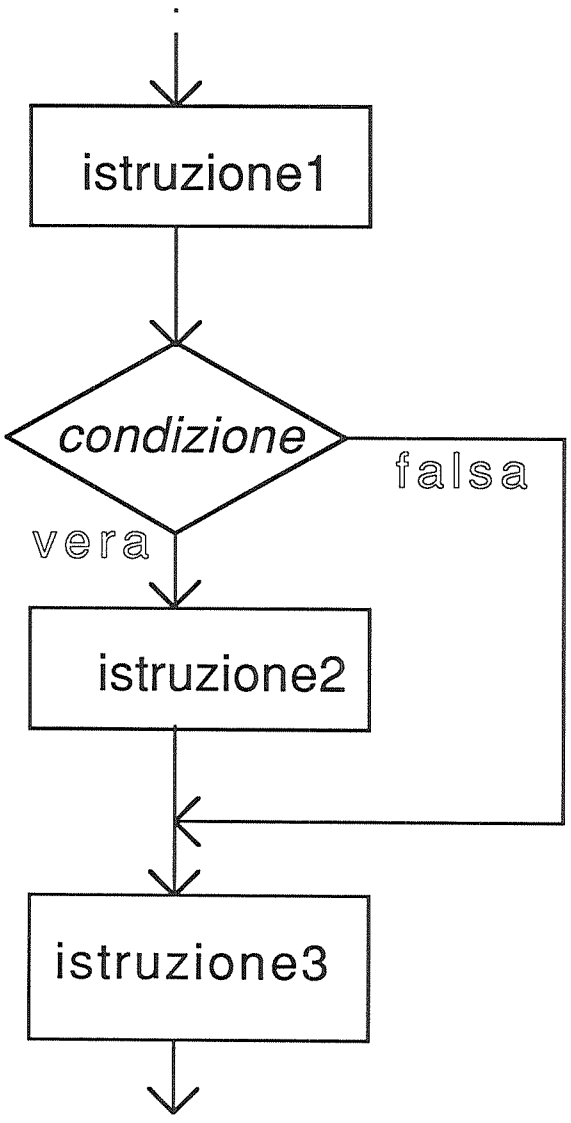
✎ Forma: case *espressione* of
*lista A di valori* : *istruzione A* (α)
*lista B di valori* : *istruzione B* (α)

..... : .....
*lista N di valori* : *istruzione N* (α)
end

(α) l'*istruzione* puo' essere composta, ed e' eseguita solo se il valore dell'*espressione* rientra tra quelli inclusi nella *lista di valori* ad essa corrispondente.

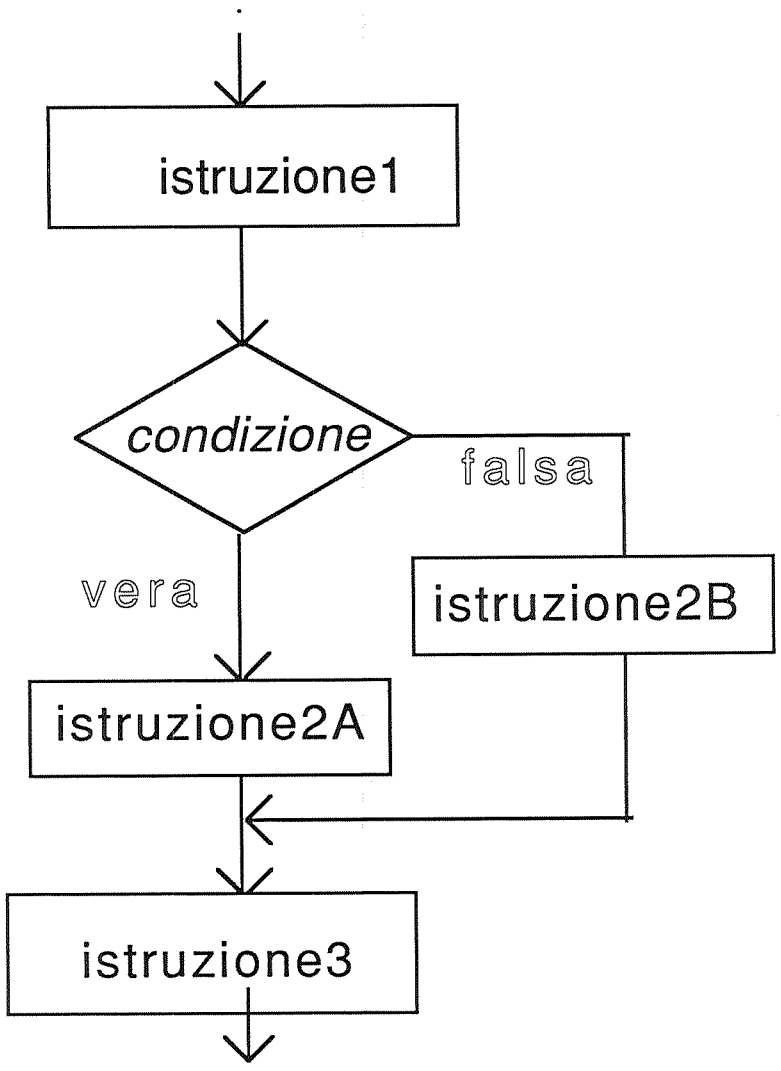
# ⇨ Diagramma di Flusso di IF-THEN

```
.....
istruzione1;
if condizione then
   istruzione2;
istruzione3;

.....
```



# ⇨ Diagramma di Flusso di IF-THEN-ELSE

```
.........
istruzione1;
if condizione then
   istruzione2A
                  else
   istruzione2B;
 istruzione3;
.........
```



# ⇨ Diagramma di Flusso di CASE

.....

istruzione1;

**case** *espressione* **of**

*lista A di valori : istruzione2A;*

*lista B di valori : istruzione2B;*

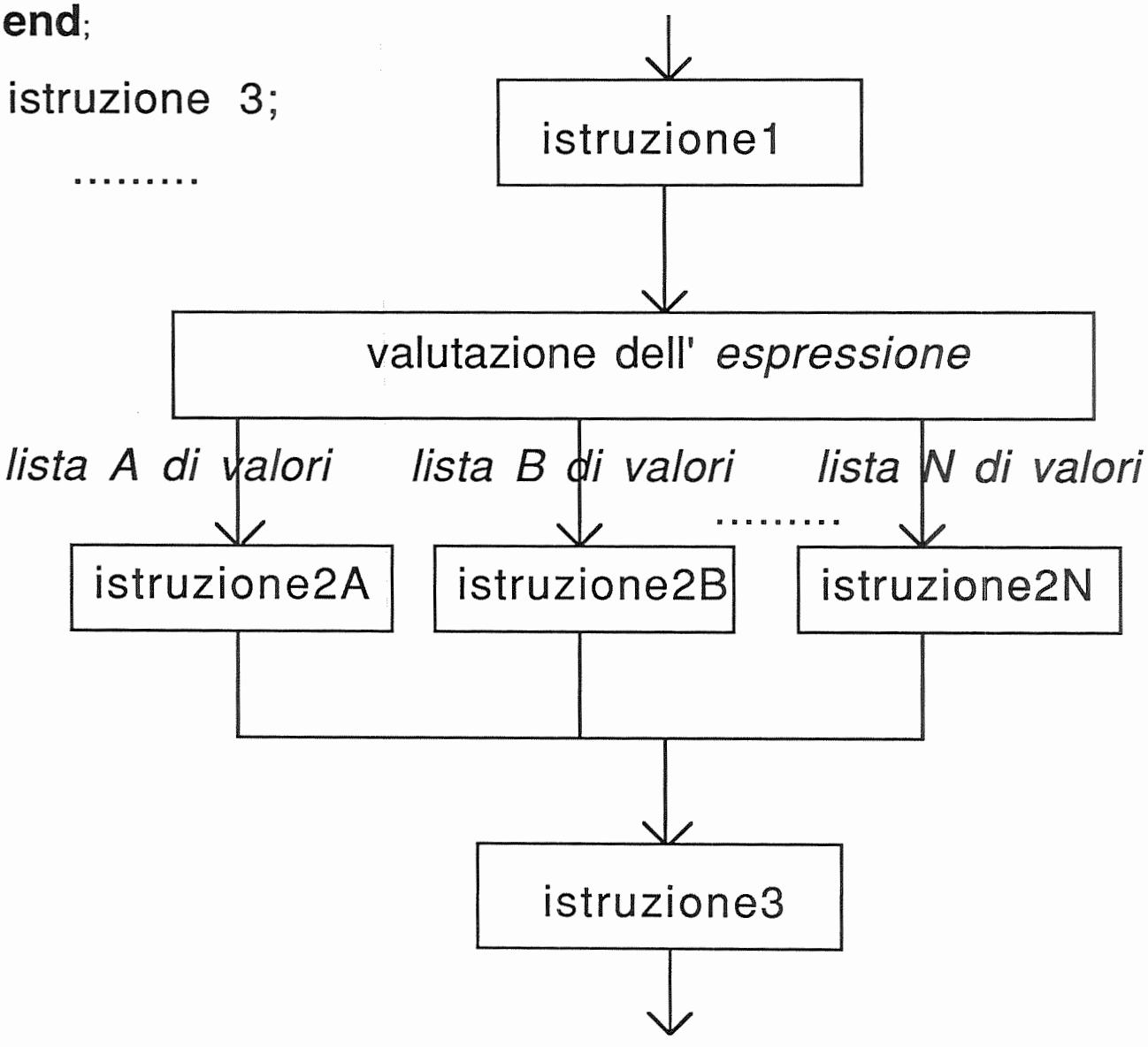
.....

*lista N di valori : istruzione 2N;*

**end**;

istruzione 3;

.....



# Esempio ❶ : uso delle Istruzioni Condizionali

```
program triangolo1 (input, output);
{Determina la natura di un triangolo, e, a scelta,
 calcola il perimetro o stampa la misura dei lati}
var
    L1, L2, L3: integer;
    x : char;
    perim: integer;
begin
 write ('Scrivere 3 numeri interi');
 writeln (' in ordine crescente.');
 readln (L1,L2,L3);
 if L1 = L3 then
              writeln('Il triangolo e'' equilatero.')
            else
              if (L1=L2) or (L2=L3)
               then writeln('Il triangolo e'' isoscele.')
               else writeln('Il triangolo e'' scaleno.');
 writeln ('Scrivere "P" per calcolare il');
 writeln ('perimetro del triangolo, o "L" ');
 writeln ('per stampare la misura dei lati.');
 readln (x);
 case x of
   'P': begin
         perim := L1+L2+L3;
         writeln ('Il perimetro e'': ', perim)
        end;
   'L': writeln ('La misura dei lati e'':', L1, L2, L3);
 end {case}
end.
```

## ☞ Si noti che:

L'istruzione (𝔸)

```
if L1 = L3 then
            writeln('Il triangolo e'' equilatero.')
          else
            if (L1=L2) or (L2=L3)
             then writeln('Il triangolo e'' isoscele.')
             else writeln('Il triangolo e'' scaleno.')
```

non e' equivalente alla seguente (𝔹):

```
begin
 if L1 = L3 then
            writeln('Il triangolo e'' equilatero');
 if (L1 = L2) or (L2 = L3)
           then writeln('Il triangolo e'' isoscele.')
           else writeln('Il triangolo e'' scaleno.')
end
```

☞ ESEMPIO: L1 = L2 = L3 = 10

*Stampa su video*

(𝔸) Il triangolo e' equilatero.

(𝔹) Il triangolo e' equilatero.
Il triangolo e' isoscele.

## ☞ Si noti che:

L'istruzione (ℂ)

```
case x of

   'P': begin
         perimetro := L1+L2+L3;
         writeln ('Il perimetro e": ', perim)
        end;

   'L': writeln ('La misura dei lati e":', L1, L2, L3);

end {case}
```

e' equivalente alla seguente (𝔻):

```
begin
if x = 'P' then
           begin
            perimetro := L1+L2+L3;
            writeln ('Il perimetro e": ', perim)
           end;

if x = 'L' then
      writeln ('La misura dei lati e":', L1, L2, L3);
end
```

## ☞ Si noti che:

L'istruzione (ℂ')

```
case x of

    'P': begin
           perimetro := L1+L2+L3;
           writeln ('Il perimetro e'': ', perim)
         end;

'M','L': writeln ('La misura dei lati e'':', L1, L2, L3);

end {case}
```

e' equivalente alla seguente (𝔻'):

```
begin
if x = 'P' then
            begin
             perimetro := L1+L2+L3;
             writeln ('Il perimetro e'': ', perim)
            end;

if (x = 'L') or (x = 'M') then
           writeln ('La misura dei lati e'':', L1, L2, L3);
end
```

# Istruzioni Ripetitive

- **istruzione repeat**

  ✎ Forma: repeat
  *lista di istruzioni da ripetere*
  until *condizione*

La *lista di istruzioni* e' ripetuta fino a che la *condizione* non e' verificata.

- **istruzione while**

  ✎ Forma: while *condizione* then
  *istruzione*

L'*istruzione* puo' essere composta, ed e' eseguita fintantoche' la *condizione* e' vera.

- **istruzione for**

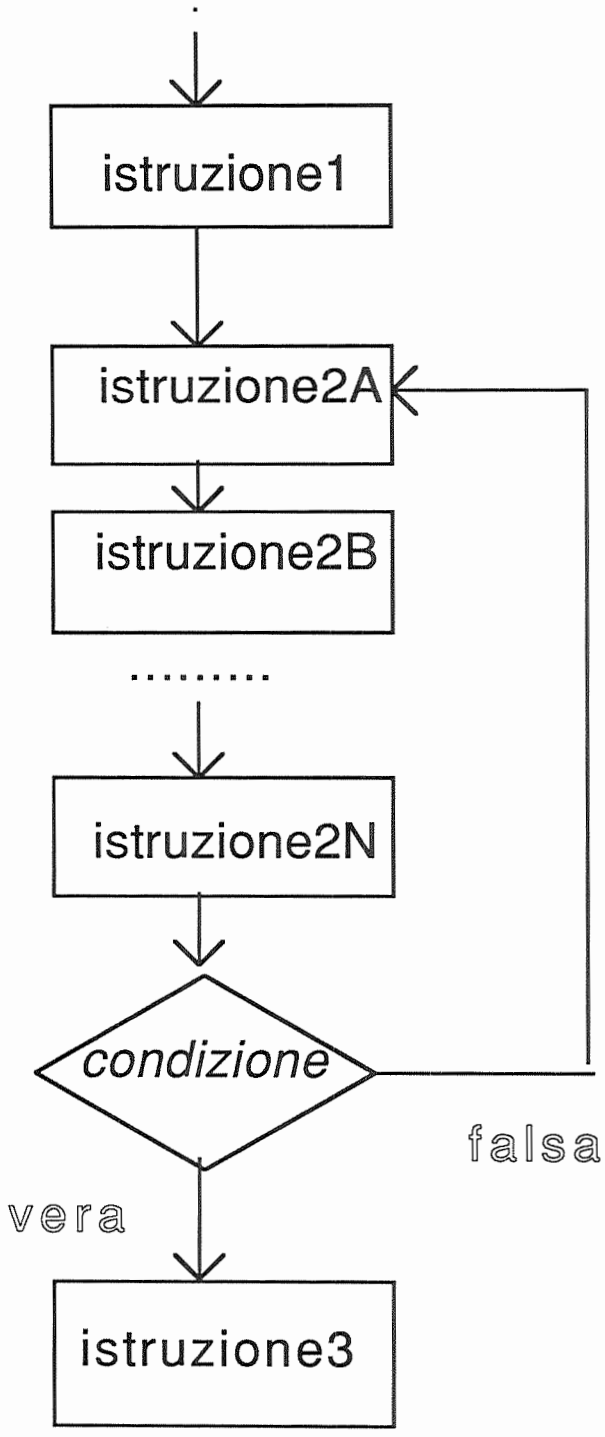
  ✎ Forma: for *var.di contr.* :=*espr.iniz.* to *espr.fin.*
  do *istruzione*

  for *var.di contr.* :=*espr.iniz.* downto
  *espr.fin.* do *istruzione*

L'*istruzione* puo' essere composta, ed e' eseguita un numero di volte pari ai valori che la *variabile di controllo* puo' assumere (calcolati in base al valore della *espressione iniziale* e di quella *finale*)

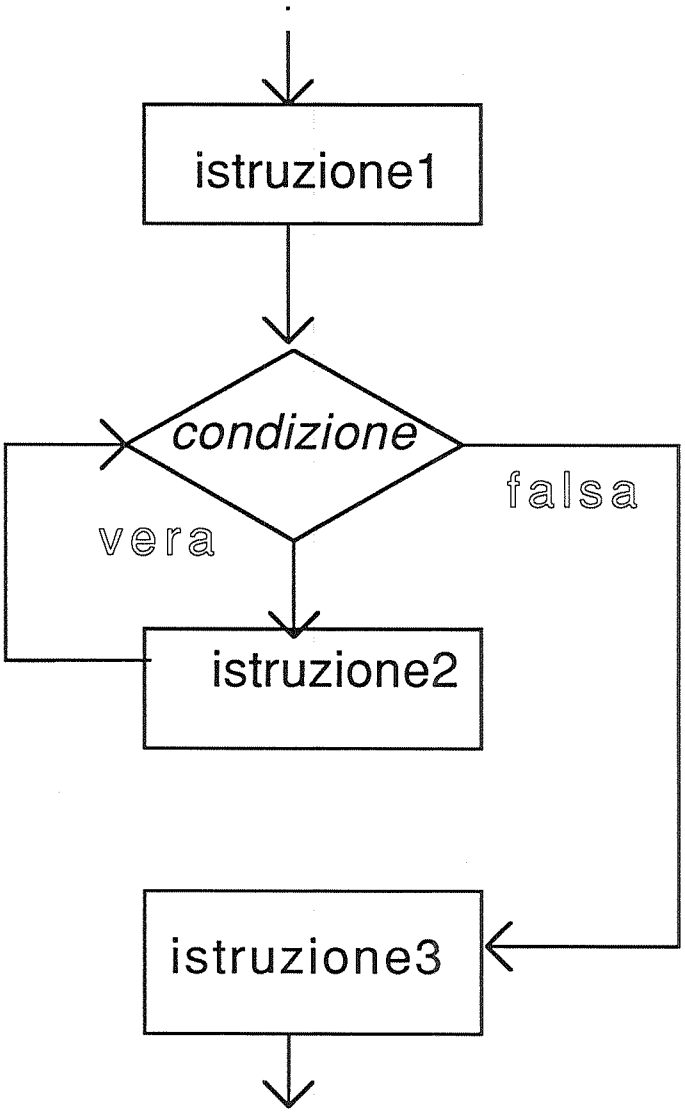
# ⇨ Diagramma di Flusso di REPEAT

```
.....
istruzione1;
repeat
     istruzione2A;
     istruzione2B;
       .....
     istruzione2N;
until condizione;
istruzione3;

.....
```



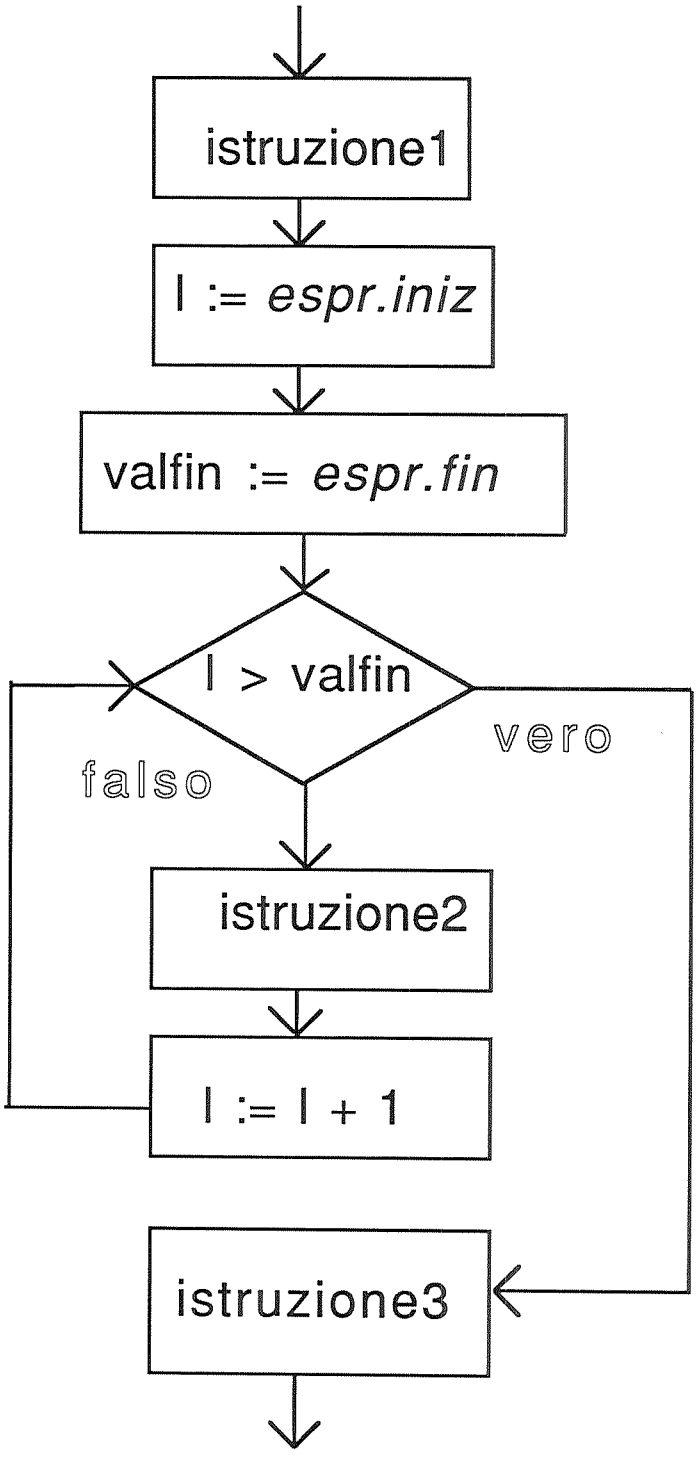
# ⇨ Diagramma di Flusso di WHILE

```
.........
istruzione1;
while condizione do
   istruzione2;
istruzione3;
.........
```



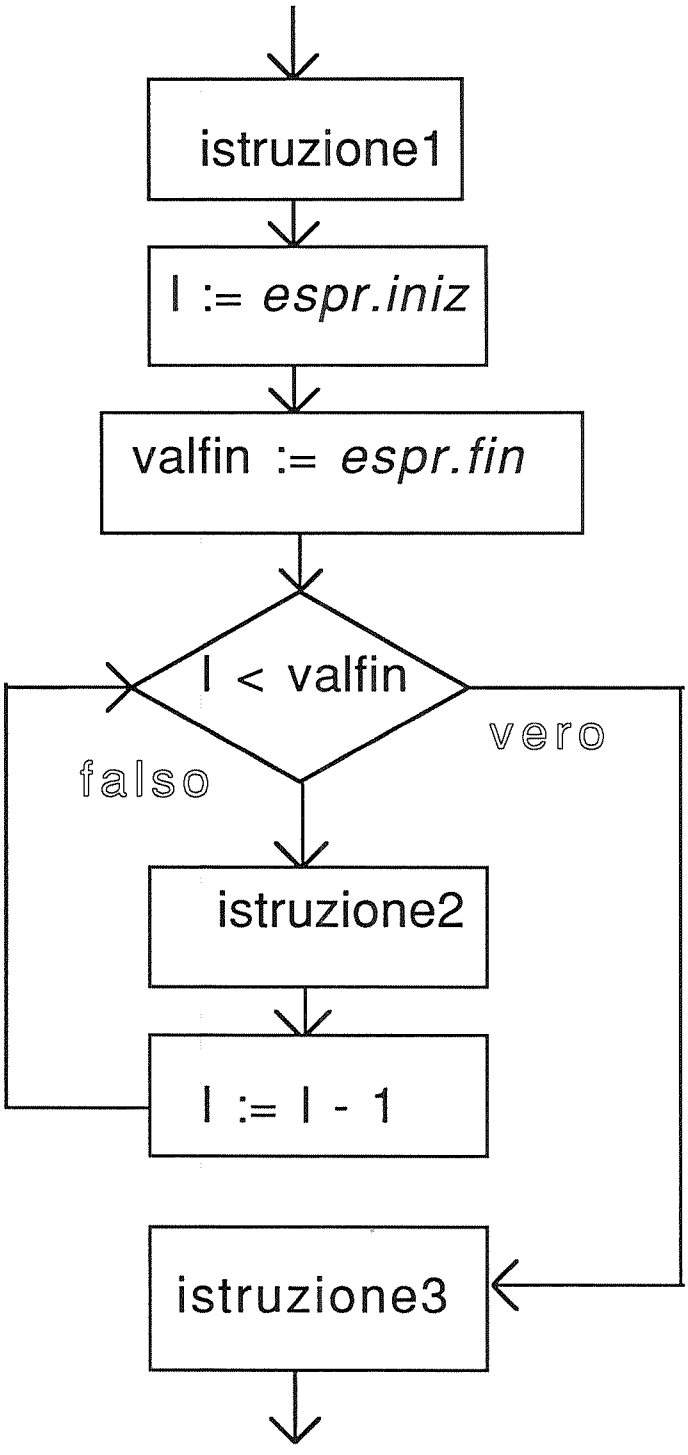
# ⇨ Diagramma di Flusso di FOR (TO)

```
.....
istruzione1;
for I := espr.iniz. to espr.fin. do
        istruzione2;

istruzione3;

.....
```



# ⇨ Diagramma di Flusso di FOR (DOWNTO)

```
   .........
istruzione1;
for I := espr.iniz. downto espr.fin. do
        istruzione2;

istruzione3;
   .........
```



# Esempio ❷ : uso delle Istruzioni Ripetitive

```
program triangolo2 (input, output);
{Determina la natura di un triangolo}
var
    L1, L2, L3: integer;
    I : integer;
    risposta : char;
begin
risposta := 's';
while risposta = 's' do
 begin
  repeat
    write ('Scrivere 3 numeri interi');
    writeln (' in ordine crescente.');
    readln (L1,L2,L3)
  until (L1 <= L2) and (L2 <= L3); {fine repeat}
  if L1 = L3 then
                writeln('Il triangolo e'' equilatero.')
               else
                if (L1=L2) or (L2=L3)
                  then writeln('Il triangolo e'' isoscele.')
                  else writeln('Il triangolo e'' scaleno.');
  for I := L1 to L2 do
    write ('TRIAN');
  for I := L3 downto L2 do
    write ('GOLO');
  writeln;
  writeln ('Vuoi continuare?');
    readln (risposta);
 end; {fine while}
end.
```

| *Dati in ingresso* | *Stampa su video* |
|---|---|
| L1 = L2 = L3 = 3 | TRIANGOLO |
| L1 = L2 = 3<br>L3 = 4 | TRIANGOLOGOLO |
| L1 = 3<br>L2 = L3 = 4 | TRIANTRIANGOLO |
| L1 = 3<br>L2 = 4<br>L3 = 5 | TRIANTRIANGOLOGOLO |

## ☞ Si noti che:

Le due seguenti istruzioni sono equivalenti:

(𝔸)

```
for I := E1 to E2 do
    istruzione1;
```

(𝔹)

```
begin
  I:= E1;
  valfin:= E2;
  while I <= valfin do
    begin
      istruzione1;
      I := I+1;
    end;
end
```

## ☞ Si noti che:

Le due seguenti istruzioni sono equivalenti:

(ℂ)

```
for I := E1 downto E2 do
    istruzione1;
```

(𝔻)

```
begin
  I:= E1;
  valfin:= E2;
  while I >= valfin do
    begin
      istruzione1;
      I := I-1;
    end;
end
```

# Le Procedure

# Procedure

✧ **caso A:** assegnazione di un nome ad un gruppo di istruzioni.

✎ Forma della *dichiarazione di procedura*:

```
procedure nome;
 begin
 istruzione 1;
 istruzione 2;
 ...
 istruzione N
 end;
```

- il *nome* e' un identificatore Pascal
- le *istruzioni* sono istruzioni Pascal

La dichiarazione della procedura va inserita fra le dichiarazioni del programma, dopo la dichiarazione delle costanti e delle variabili.

✎ Forma della *chiamata di procedura*:

```
nome;
```

La chiamata della procedura (**istruzione di tipo semplice**) deve avvenire nel corpo del programma.

## ❒ struttura del programma

```
program nomeprograma (input, output);
const
    dichiarazione delle costanti
var
    dichiarazione delle variabili

    dichiarazione delle procedure
    (e.g., procedura alfa)
  ☞ procedure alfa;
    begin
    istruzioni { corpo della procedura alfa}
    end;
  .....
begin {programma nome}
    .....
    istruzioni;
    alfa;
    istruzioni;
    .....
end. {programma}
```

## ❒ utilita' delle procedure:

..

◇ semplificare la scrittura del codice del programma (o **modulo principale**)

- ripetizione di porzioni di codice
- divisione del problema in sottoproblemi

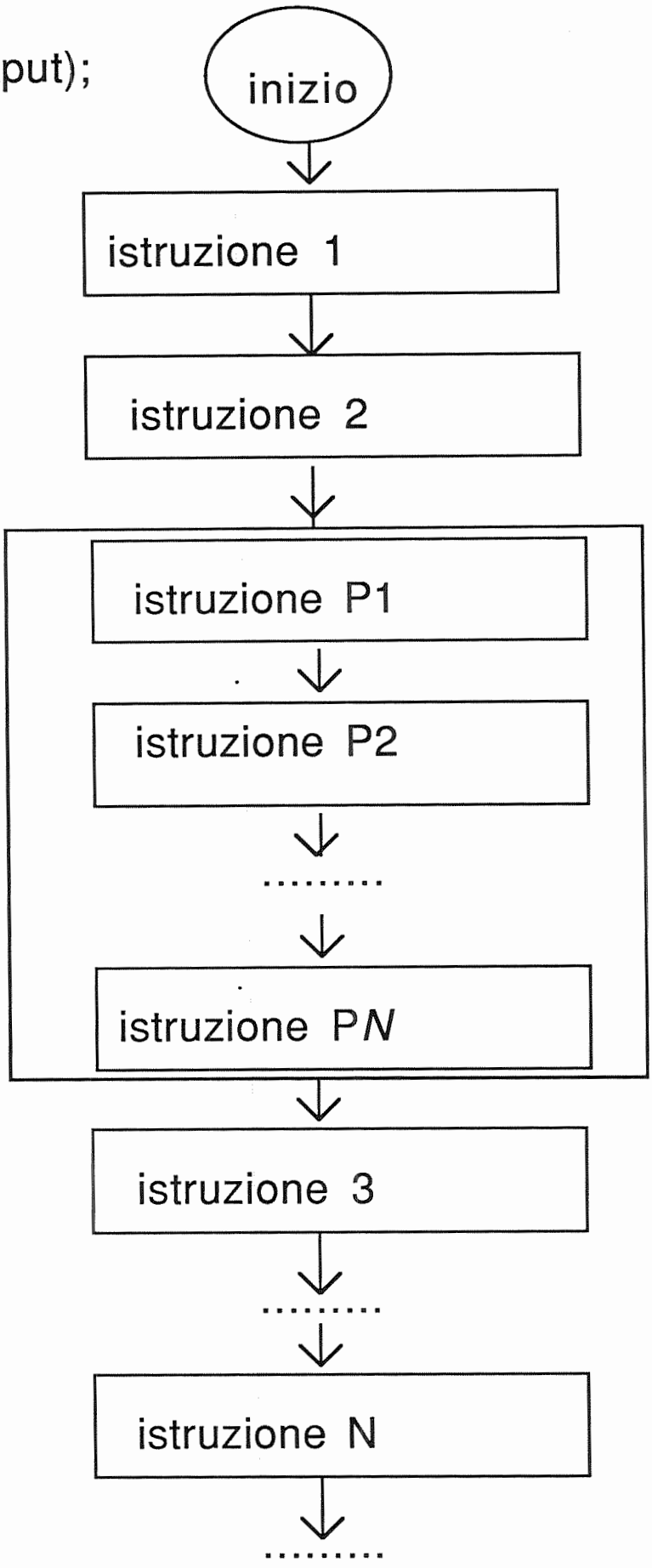
◇ aumentare la leggibilita' del programma

# Esempio ❸ : uso delle procedure

```
program triangolo1A (input, output);
{Determina la natura di un triangolo, e, a scelta,
 calcola il perimetro o stampa la misura dei lati}
var
    L1, L2, L3: integer;
    x : char;
    perim:  integer;

procedure  esamina;
 begin
  if L1 = L3 then
              writeln('Il triangolo e'' equilatero.')
              else
                if (L1=L2) or (L2=L3)
                   then writeln('Il triangolo e'' isoscele.')
                   else writeln('Il triangolo e'' scaleno.');
 end; {esamina}

procedure  calcola;
begin
 case x of
   'P': begin
          perimetro := L1+L2+L3;
          writeln ('Il perimetro e'': ', perim)
        end;
    'L': writeln ('La misura dei lati e'':', L1, L2, L3);
 end {case}
end;{calcola}
```

```
{fine  delle dichiarazioni e inizio del corpo del
programma}
begin
 write ('Scrivere 3 numeri interi');
 writeln (' in ordine crescente.');
 readln (L1,L2,L3);
 esamina;
 writeln ('Scrivere "P" per calcolare il');
 writeln ('perimetro del triangolo, o "L" ');
 writeln ('per stampare la misura dei lati.');
 readln (x);
 calcola ;
end.
```

```
program prova (input,output);
 const .....
 var   .....
procedure PPP;
 begin
 istruzione P1;
 istruzione P2;
   .....
 istruzione PN
 end;

begin
{corpo del programma}
istruzione 1;

istruzione 2;
PPP;
istruzione 3;
   .....
istruzione N;
   .....
end.
```



# Procedure

✧ **caso B:** assegnazione di un nome ad un gruppo di istruzioni, e utilizzazione di **parametri**.

✎ Forma della *dichiarazione di procedura*:

```
procedure nome (lista parametri formali);
 begin
 istruzione 1;
 istruzione 2;
 ...
 istruzione N
 end;
```

I *parametri formali* (*p.f.*) sono oggetti fittizzi introdotti per specificare la natura degli elementi manipolati nel corpo della procedura. Sono utilizzati per dare generalita' al codice della procedura. Possono assumere forme diverse, e servono ad indicare in modo esplicito lo scambio di informazioni fra il modulo chiamante e la procedura.

✎ Forma della *chiamata di procedura*:

*nome (lista parametri attuali)*;

I *parametri attuali* sono oggetti definiti nel modulo chiamante. Sono associati ai *parametri formali* in base all'ordine con cui sono specificati nella chiamata. Devono corrispondere ai *p. f.* per numero e per tipo.

# ➢ Parametri Formali

- **per variabile**

un parametro formale per variabile denota un parametro attuale il cui valore puo' essere alterato dalla procedura

- **per valore**

un parametro formale per valore denota un parametro attuale il cui valore e' passato alla procedura (il valore del parametro attuale non viene alterato dalla esecuzione della procedura)

✎ Specifica di *param. formali* per variabile:

☞ var lista di nomi : identificatore.tipo (α);

✎ Specifica di *param. formali* per valore:

lista di nomi : identificatore.tipo (α);

(α) e.g., integer, real, char, .....

## ➢ Parametri Attuali

- **per variabile**
- **per valore**

✎ Specifica di *param. attuale* per variabile:

variabile dichiarata nel modulo chiamante

✎ Specifica di *param. attuale* per valore

espressione definita nel modulo chiamante

## ☞ Ricordarsi che:

Il *modulo chiamante* puo' essere il programma principale oppure un modulo definito al suo interno.

## ☞ Si noti che:

- l'intestazione di procedura

```
procedure PPP (var P1: integer;
               var P2: real; var P3: real);
```

e' equivalente alla seguente:

```
procedure PPP (var P1: integer;
               var P2, P3: real);
```

- la parola riservata **var** deve essere ripetuta per ogni gruppo di parametri formali per variabile.

- l'intestazione di procedura

```
procedure PPP (var P1: integer;
               var P2: real;  P3: real);
```

non e' equivalente alla seguente:

```
procedure PPP (var P1: integer;
               var P2, P3: real);
```

# Esempio ❹ : uso delle procedure

```
program triangolo1B (input, output);
{Determina la natura di un triangolo, e, a scelta,
 calcola il perimetro o stampa la misura dei lati}
var
    L1, L2, L3: integer;
    x : char;
    perim: integer;

procedure esamina ( var a,b,c: integer);
begin
 if a = b then
            writeln('Il triangolo e" equilatero.')
          else
             if (a=b) or (b=c)
               then writeln('Il triangolo e" isoscele.')
               else writeln('Il triangolo e" scaleno.');
end; {esamina}

procedure calcola ( var a,b,c: integer; scelta: char;
                    var perim: integer;);
begin
 case scelta of
   'P': begin
         perim := a+b+c;
         writeln ('Il perimetro e": ', perim)
        end;
   'L': writeln ('La misura dei lati e":', a, b, c);
 end {case}
end;{calcola}
```

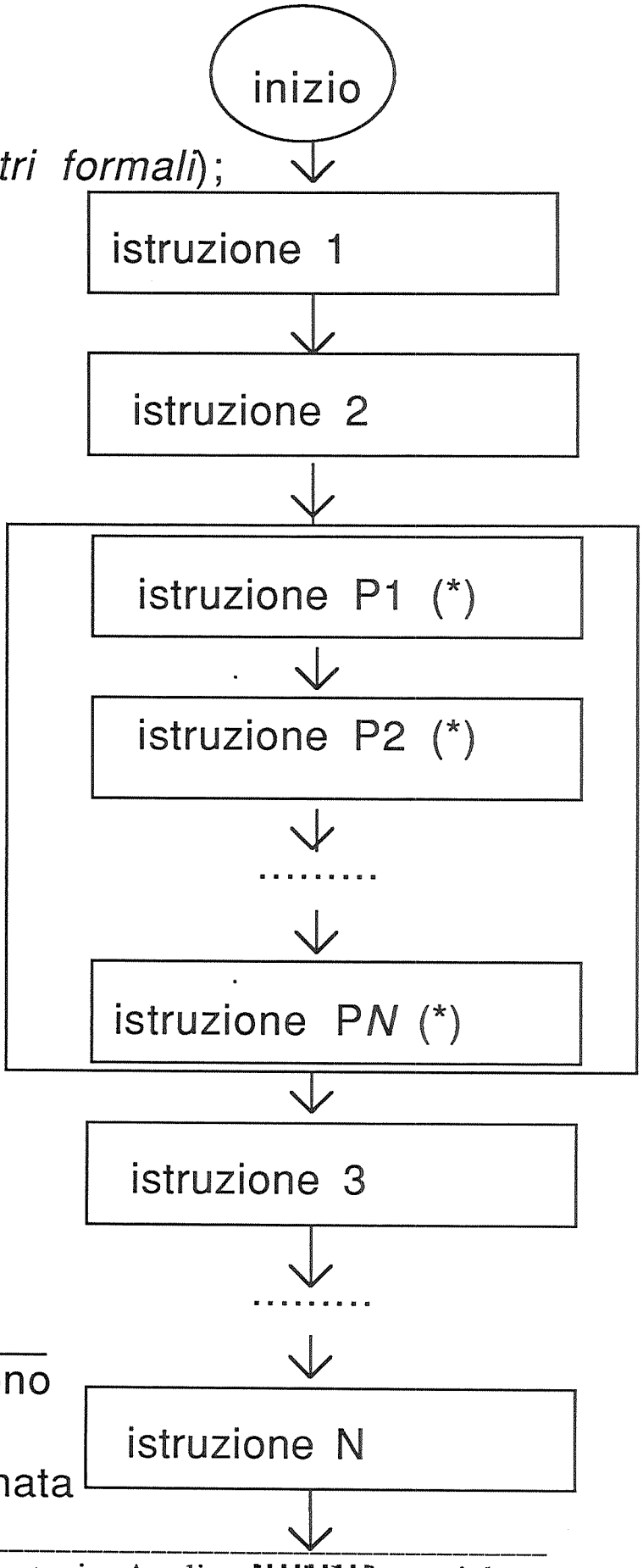
# Esempio ❺ : uso delle procedure

```
program triangolo1C (input, output);
    L1, L2, L3: integer;
    x : char;
    perim, M1, M2, M3, I: integer;

procedure esamina ( a,b,c: integer);
 begin
 ....
 end; {esamina}

procedure calcola ( a,b,c: integer; scelta: char;
                    var perim: integer;);
 begin
 .....
 end;{calcola}

procedure leggi3NUM (var L1,L2,L3: integer);
 begin
   write ('Scrivere 3 numeri interi');
   writeln (' in ordine crescente.');
   readln (L1,L2,L3);
 end;

procedure opzione (var z: char);
 begin
   writeln ('Scrivere "P" per calcolare il');
   writeln ('perimetro del triangolo, o "L" ');
   writeln ('per stampare la misura dei lati.');
   readln (z);
 end;
```

```
{fine delle dichiarazioni ed inizio del corpo del
programma }
begin
 write ('Scrivere 3 numeri interi');
 writeln (' in ordine crescente.');
 readln (L1,L2,L3);
 esamina (L1,L2,L3);
 writeln ('Scrivere "P" per calcolare il');
 writeln ('perimetro del triangolo, o "L" ');
 writeln ('per stampare la misura dei lati.');
 readln (x);
 calcola (L1,L2,L3, x, perim);
end.
```

```
{fine delle dichiarazioni ed inizio del corpo del
programma }
begin
   leggi3NUM (L1,L2,L3);
   esamina (L1,L2,L3);
   opzione (x);
   calcola (L1,L2,L3, x, perim);
   leggi3NUM (M1,M2,M3);
   esamina (M1,M2,M3);
   for I := 1 to 3 do:
     begin
       leggi3NUM (L1,L2,L3);
       esamina (L1,L2,L3);
       opzione (x);
       calcola (L1,L2,L3, x, perim);
     end;

   while perim < 100 do
     begin
       leggi3NUM (L1,L2,L3);
       opzione (x);
       calcola (L1,L2,L3, x, perim);
     end;

end.
```

```
program prova (input,output);
 const .....
 var   .....
procedure PPP (parametri formali);
 begin
 istruzione P1;
 istruzione P2;
   .....
 istruzione PN
 end;

begin
{corpo del programma}
istruzione semplice1;

istruzione semplice2;
PPP (parametri attuali);
istruzione semplice3;
   .....
istruzione N;
   .....
end.
```




(*) i *parametri formali* sono sostituiti con i *parametri attuali* indicati nella chiamata

## ➢ Sostituzione dei parametri formali con i parametri attuali

passo 1:. *associazione tra parametri attuali e parametri formali*

primo *p. a.* ------> primo *p. f.*
secondo *p. a.* -----> secondo *p. f.*
.....
ultimo *p. a.* -----> ultimo p. f.

passo 2: *sostituzione dei parametri formali nel corpo della procedura con i parametri attuali, al momento della chiamata della procedura*

☞ ciascuna occorrenza di un parametro formale nel corpo della procedura viene sostituito con il parametro attuale che corrisponde al parametro formale in base al passo 1.

## ☞ Ricordarsi che

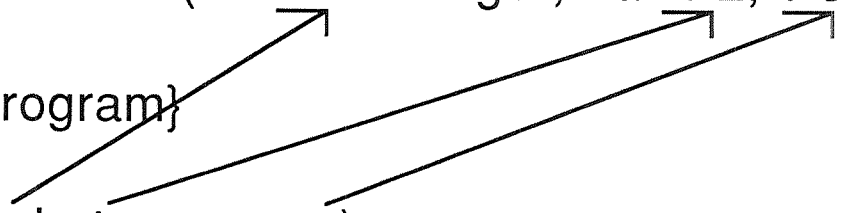
per ciascuna chiamata di procedura

- il numero di parametri attuali deve coincidere con il numero di parametri formali
- il tipo di ciascun parametro attuale deve coincidere con il tipo del parametro formale corrispondente

ESEMPIO :

```
program prova (input, output);
...
procedure PPP (var P1: integer; var P2, P3: real);
....
begin {program}
.....
PPP(alfa, beta, gamma);
....
end. {program}
```

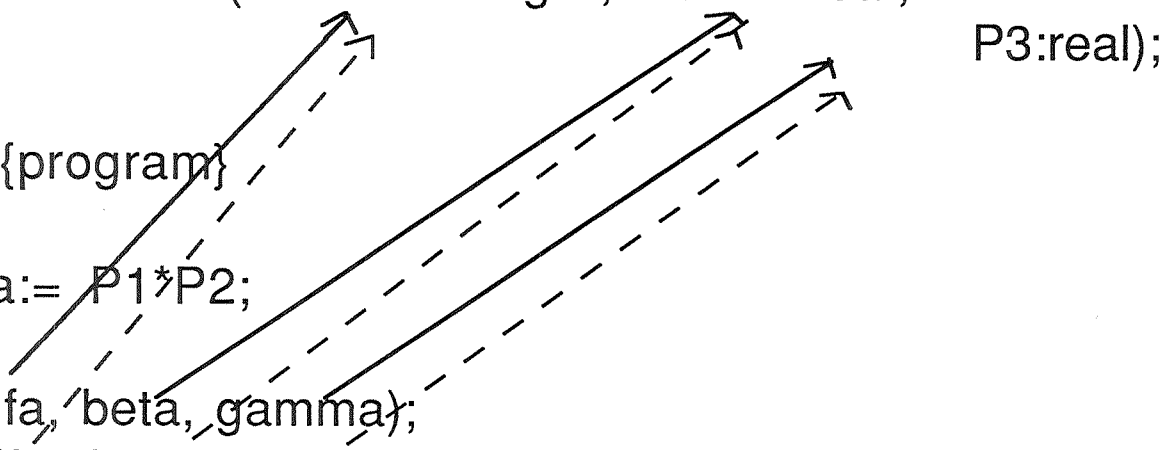
```
program prova (input, output);
...
beta, gamma: real;
alfa: integer;

procedure PPP (var P1: integer; var P2, P3: real);
....
begin {program}
.....
PPP(alfa, beta, gamma);
....
end. {program}
```



➪ ESEMPIO :

```
program prova (input, output);
...
procedure PPP (var P1: integer; var P2:real;
                    P3: real);
....
begin {program}
.....
PPP(alfa, beta, gamma);
....
end. {program}
```

```
program prova (input, output);
...
alfa: integer;
beta:real;
gamma:real;

procedure PPP (var P1: integer; var P2:real;
                                    P3:real);
...
begin {program}
.....
gamma:= P1*P2;
.....
PPP(alfa, beta, gamma);
PPP(alfa, beta, P1/P2);
....
end. {program}
```



# ➢ Esecuzione delle procedure con parametri

La chiamata della procedura produce l'esecuzione del corpo della procedura relativamente ai parametri attuali forniti.

Il corpo della procedura viene eseguito sostituendo:

- per ciascuna occorrenza di ciascun parametro formale per variabile la corrispondente variabile (parametro attuale) nella lista dei parametri attuali fornita dalla chiamata.

- per ciascuna occorrenza di ciascun parametro formale per valore il valore della corrispondente espressione (parametro attuale) nella lista dei parametri attuali fornita dalla chiamata.

# ☞ Riassumendo, ricordarsi che:

l'elaborazione di una chiamata di procedura con parametri avviene in tre passi:

passo 1: associazione tra parametri attuali e parametri formali.

passo 2: sostituzione della chiamata di procedura con il corpo della procedura opportunamente istanziato con i parametri attuali.

passo 3: esecuzione del corpo della procedura corrispondente alla particolare chiamata (vedi passo 2).

Questi passi sono effettuati per ciascuna chiamata di procedura. Le chiamate di procedura sono indipendenti le une dalle altre.

## ➢ Utilizzazione dei parametri *per variabile:*

consentono l'importazione e l'esportazione di valori tra la procedura e il modulo chiamante.

La chiamata di procedura produce una effettiva manipolazione delle variabili corrispondenti ai parametri formali, per ciascuna occorrenza di tali parametri nel corpo della procedura.

Le variabili che costituiscono i parametri attuali possono essere lette e/o modificate dalla procedura. Al termine dell'esecuzione della procedura il loro valore puo' risultare diverso da quello che avevano precedentemente. In tal caso il vecchio valore e' perduto.

ESEMPIO :

```
program es (input, output);
var N1, N2, N3, MAX: integer;
procedure maggiore (var P1, P2, MAX: integer);
  begin
   if P1 >= P2 then MAX := P1
               else MAX := P2;
  end;
begin {program}
   N1:= 3;
   N2:= 7;
   N3:= 4;
   MAX:=0;
   maggiore (N1,N2,MAX);
   writeln(MAX);                (α)
   maggiore (N2,N3,MAX);
   writeln(MAX);                (β)
   maggiore(N1,N3,MAX);
   writeln(MAX);                (γ)
end. {program}
```

.....

| | N1 | N2 | N3 | MAX |
|---|---|---|---|---|
| *Dati in ingresso* | 3 | 7 | 4 | 0 |
| *Stampa su video* | | | | |
| (α) | 3 | 7 | 4 | 7 |
| (β) | 3 | 7 | 4 | 7 |
| (γ) | 3 | 7 | 4 | 4 |

## ➢ Utilizzazione dei parametri *per valore*:

consentono di importare all'interno della procedura i valori di espressioni (possibilmente complesse), i cui componenti sono definiti nel modulo chiamante.

La chiamata di procedura produce la valutazione della espressione che compare come parametro attuale, e la utilizzazione di tale valore (senza rivalutare ogni volta l'espressione) per ciascuna delle occorrenza del parametro formale a cui tale espressione corrisponde.

☞ Nel caso particolare in cui l'espressione e' definita da un unico elemento che e' una variabile, la procedura utilizza il valore di tale variabile, ma non lo modifica in nessun caso.

ESEMPIO :

```
program es (input, output);
var N1, N2, N3, MAX: integer;
procedure maggiore  (P1, P2, MAX: integer);
 begin
   if P1 >= P2 then MAX := P1
               else MAX := P2;
 end;
begin {program}
  N1:= 3;
  N2:= 7;
  N3:= 4;
  MAX:=0;
  maggiore (N1,N2,MAX);
  writeln(MAX);
  maggiore (N2,N3,MAX);
  writeln(MAX);
  maggiore(N1,N3,MAX);
  writeln(MAX);
  maggiore(N1+3,N2,MAX);
  writeln(MAX);
  maggiore(N1+N3,3*N2,MAX);
  writeln(MAX);
  maggiore(10*(N1+N3),N2*5,MAX);
  writeln(MAX);
end. {program}
```

# ☞ Riassumendo, ricordarsi che:

- i parametri per variabile consentono di esportare nel modulo chiamante valori di variabili calcolati per mezzo della chiamata di procedura: *la procedura puo' quindi effettivamente modificare il valore delle varibili definite nel modulo chiamante.*

- i parametri per valore consentono di utilizzare espressioni complesse come parametri attuali: *la procedura, quindi, puo' essere attivata non solo per valori di singole variabili, ma anche per valori ottenuti da una combinazione (espressione) di valori di variabili e di costanti .*

✧ **caso C:** procedure con parametri aventi variabili locali.

✎ Forma della *dichiarazione di procedura*:

```
procedure nome (lista parametri formali);
  ☞ dich. variabili locali;
 begin
  istruzione 1;
  istruzione 2;
  ...
  istruzione N
 end;
```

Le *variabili locali* sono variabili la cui esistenza e il cui valore sono definiti solo all'interno della procedura che le dichiara.

✎ Forma della *chiamata di procedura*:

*nome (lista parametri attuali)*;

I *parametri locali* sono oggetti definiti nel modulo chiamante. Devono corrispondere ai *parametri formali* nel numero e nel tipo.

ESEMPIO :

```
program es (input, output);
var N1, N2: integer;
procedure scambia  (var P1, P2: integer);
  var      sos: integer;
  begin {scambia}
    sos:= P1;
    P1:= P2;
    P2:= sos
  end; {scambia}
 begin {program}
    N1:= 3;
    N2:= 7;
    scambia(N1,N2);
    writeln(N1,N2);
    scambia(N1,N2);
    writeln(N1,N2);
 end. {program}
```

☞ P1 e P2 potrebbero essere parametri per valore?

i.e.: procedure scambia ( P1, P2: integer);

Risposta: NO

☞ e' possibile utilizzare piu' volte uno stesso identificatore, ma solo in moduli diversi (vedi *campo di azione* dei singoli identificatori)

✍ ESEMPIO :

```
program es (input, output);
var N1, N2, sos: integer;
procedure scambia  (var P1, P2,: integer);
  var      sos: integer;
  begin
    sos:= P1;
    P1:= P2;
    P2:= sos
  end;
begin {program}
  sos:=0;
  N1:= 3;
  N2:= 7;
  scambia(N1,N2);
  writeln(N1,N2,sos);          {N1=7, N2=3, sos=0}
  sos:=15;
  scambia(N1,N2);
  writeln(N1,N2,sos);          {N1=3, N2=7, sos=15}
end. {program}
```

☞ i parametri per valore sono variabili locali implicite

# Procedure

✧ **caso D:** procedure con parametri aventi dichiarazioni locali includenti altre procedure.

✎ Forma della *dichiarazione di procedura*:

```
procedure nome (lista parametri formali);
   dich. variabili locali
   dich. procedure locali
      (e.g., procedura beta)
      ☞ procedure beta;
          begin
             istruzioni { corpo della procedura beta}
          end;

   begin
   istruzione 1;
   istruzione 2;
   ...
☞ beta;   <--- chiamata procedure locali
   istruzione N;
   ...
   end;
```

➪ ESEMPIO :

```
program es (input, output);
var N1, N2, N3, MAX: integer;
procedure maggiore (var P1, P2:, MAX: integer);
   procedure scambia  (var P1, P2: integer);
     var      sos: integer;
     begin {scambia}
        sos:= P1;
        P1:= P2;
        P2:= sos
     end; {scambia}
begin {maggiore}
   if P1 >= P2 then MAX := P1
                else begin
                       MAX := P2;
                       scambio(P1,P2)
                     end;
end; {maggiore}
begin {program}
  N1:= 3;
  N2:= 7;
  N3:= 4; MAX:=0;
  maggiore (N1,N2,MAX);
  writeln(N1,N2,N3,MAX);
  maggiore (N2,N3,MAX);
  writeln(N1,N2,N3,MAX);
  maggiore(N1,N3,MAX);
  writeln(MAX);
end. {program}
```

## ☞ Ricordarsi che:

Il *campo di azione* della dichiarazione di un identificatore (di variabile, di costanti o di procedura) e' definito dalle seguenti regole:

**R1.** Il campo di azione e' costituito dal modulo in cui compare la dichiarazione, e da tutti i moduli racchiusi in esso, a meno della regola R2.

**R2.** Quando un identificatore dichiarato in un modulo A e' ridefinito in un modulo B racchiuso da A, allora il modulo B, e tutti i moduli da esso racchiusi, sono esclusi dal campo di azione della dichiarazione dell'identificatore in A.

```
program principale (input, output);
  const U1= ...          campo di azione di U1, X
  var X : ...             A e C
  procedure A ...
     const U2 = ...      campo di azione di U2 , Y
     var Y : .....        e B,
        procedure B ...
              const U3 = ...   campo di azione di U3 e di Z
              var Z : ....
           begin {B}
           ....
           end; {B}
  begin {A}
  ...
  end; {A}
     procedure C ...
       const  U4 = ...     campo di azione di U4 e di W
       var W : ...
      begin {C}
     ...
      end; {C}
begin {principale}
....
end. {principale}
```

## *Riferimenti Bibliografici*

J. Winston Crawley, William G. McArthur, *Structured Programming Using Pascal*, Prentice-Hall International Editions, Inc., 1988.

Walter J. Savitch, *An Introduction to the Art and Science of Programmng*, The Benjamin/Cummings Publishing Co., Inc., 1986.

Jim Welsh, J. Elder, *Introduzione al Pascal*, Esa ed., 1983,