# PROGRAMMARE IN PASCAL

SECONDA RISTAMPA AGGIORNATA E CORRETTA 01/06/94

a cura di **Andrea '73**

## ⇨ STRUTTURA DI UN PROGRAMMA IN PASCAL:

Un programma nel linguaggio Pascal è così strutturato:

**Program *esempio*;** Inizia con la scritta Program seguita dal nome del programma

**Label** Si dichiarano le etichette
etichetta1,et2;

**Const** Si dichiarano le costanti
a=100;
b=' stringa ';
c=false;

**Type** Si dichiarano i tipi di variabili definiti a piacere dall'utente
stringa=string[25];

**Var** Si dichiarano le variabili e l'utilizzo che se ne fa
g,h,j : stringa;
k,l : integer;
r : real;
oo : boolean;

**Procedure** Si scrivono le procedure
es. Procedure linea(colore : integer; lineaz : string);

**Function** Si scrivono le funzioni
es. Function prodotto(a,b: integer;) : real;

**Begin;** Inizio del programma principale

programma principale

**End.** Fine del programma principale

## ⇨ ALCUNE REGOLE DI SINTASSI:

La prima istruzione del programma deve essere ' Program nome; '
Ogni istruzione deve essere seguita dal punto e virgola ' ; ' (tranne alcune istruzioni).
L' operazione di assegnamento viene codificata tramite il simbolo ' := '
Ogni sottoinsieme di più operazioni deve iniziare con 'Begin' ed finire con un 'End;'
L'ultima istruzione del programma deve essere 'End.'

## ⇨ PROCEDURE E FUNZIONI:

Il linguaggio Pascal segue la logica della programmazione modulare: ogni programma viene diviso in piccole parti, ognuna delle quali svolge un suo specifico compito. Queste parti sono chiamate *procedure* e *funzioni*, ***Procedure*** e ***Function***.
La differenza tra una procedura ed una funzione consiste nel modo in cui queste restituiscono i valori; si osservino i seguenti esempi:

```
Procedure Quadra(x : real; Var x2 : real);
Begin
 x2 := x * x;
End;

Function Quadrato(x : real) : real;
Begin
 Quadrato := x * x;
end;
```

Alla procedura Quadra sono passati due parametri. Il primo parametro *x* è il numero di cui calcolare il quadrato; il secondo *x2* è il risultato. Il parametro *x2* viene passato per *indirizzo*, ovvero al termine della procedura i cambiamenti subiti dal parametro sono gli stessi della variabile che sostituisce nel programma principale il parametro.
Il parametro x viene invece passato per *valore*. In seguito verrà spiegato questo concetto (vedere **PARAMETRI PASSATI PER INDIRIZZO O PER VALORE**).
La funzione quadrato ha invece un solo parametro *x* e restituisce il risultato attraverso la funzione stessa. Per capire meglio questa differenza se ne esamini l'uso in un programma:

*Uso della procedura Quadra*
```
Quadra(x , x2);
write(x2);
if x2>100 then ...
```

*Uso della funzione Quadrato*
```
x2 := Quadrato(x);
write(Quadrato(x));
if Quadrato(x)>100 then...
```

Si nota quindi che una funzione deve obbligatoriamente essere usata in una istruzione di assegnamento, stampa, di confronto o simili; mentre una procedura può essere eseguita come un qualsiasi comando. Quando non servono risultati (ad esempio per stampare delle scritte, utilizzare la grafica, etc.) si utilizzano le procedure, mentre quando servono risultati si potranno utilizzare indifferentemente funzioni o procedure.

## ⇨ PARAMETRI PASSATI PER INDIRIZZO O PER VALORE:

Osservare l'esempio della procedura Quadra: dopo che è stata richiamata nel programma sopra riportato, il valore del parametro *x2* risulta modificato, mentre il parametro *x* rimane immutato. Quindi *x* può essere considerato il parametro di ingresso, mentre x2 è il parametro in uscita, cioè il risultato.

Attenzione: la procedura può essere eseguita anche nel modo seguente:

Quadra(C,A):

In questo caso *C* rimane invariato, non modifica il suo valore, ma lo trasferisce momentaneamente al parametro o variabile *locale* x ; mentre *A* diventa a tutti gli effetti il sostituto di *x2* ed alla fine della procedura il suo valore diventa quello di *x2*, cioè il quadrato di *C*. Per gli esperti: il nome parametro passato per indirizzo indica il fatto che i puntatori della variabile x2 diventano gli stessi della variabile C, quindi ogni modifica eseguita su x2 risulterà eseguita anche su C.

## ⇨ TIPI DI DATI:

Il Pascal mette a disposizione alcuni tipi di dati standard:

**REAL** per numeri con cifre decimali (*reali*)
**INTEGER** per numeri senza cifre decimali (*interi*) da -32768 a +32767
**BOOLEAN** per esprimere condizioni logiche di vero o falso (*true* o *false*)
**STRING** per gestire frasi (è definito come *array [1..255] of char*)
**CHAR** per gestire i singoli caratteri
**WORD** per numeri interi positivi da 0 a 65535
**SHORTINT** per numeri interi da -128 a +127
**LONGINT** per numeri interi da -2147483648 a +2147483647

In aggiunta a questi, l'utente può definirne altri. Questi tipi di dati possono essere principalmente di tre categorie: scalari, record e array.

**SCALARI**: servono per codificare dati al fine di rendere più leggibile un listato e più semplice da realizzare.

Esempio:
Colore = (rosso, verde, blu, giallo, nero, bianco);

In questo caso si è definito il tipo Colore che potrà assumere solo uno degli elementi contenuti ( rosso , blu ...). Nel programma si otterrà quindi una comprensione immediata: si legge blu e non ad esempio 88.

**RECORD**: un record è un aggregato di più tipi di dati che formano un nuovo tipo di dati, ad esempio:

```
Cliente = Record
    Nome : string[30];
    Indirizzo : string[50];
    Eta : shortint;
    Particolarità : String[20];
    Reddito : Real;
   End;
```

L'utilizzo di un record comporta alcuni vantaggi: tutti gli elementi sono collegati tra loro; se lo si utilizza nella gestione dei file su disco, l'assegnamento, la scrittura e la lettura possono essere fatte sul record invece che sul singolo elemento; in caso di ordinamento alfabetico o altro, si compiono le operazioni solo sul record e non sui singoli elementi.
Nel programma i singoli componenti vengono così richiamati:

```
Cliente.nome : =' . . . t . . .  . . . . '
Cliente.Indirizzo : =' . . . . l . . . '
Cliente.Eta : = ' . . '
Cliente.Particolarità : =' . . . . i . . . . . '
```

Vengono cioè chiamati con il nome del record seguito da un punto ' . ' e dal nome del campo.

**ARRAY**: gli array è un insieme di dati di tipo uguale che si ripete per un certo numero di volte; esempio di array:

```
voto : array [1..450] of string;
```

Quindi un array si definisce cosi: ***nomevariabile*** **: array [*inizio .. fine*] of *tipodati***
Nel programma viene utilizzato così:

```
voto[73] : = '30+';
voto[74] : = '30+';
voto[75] : = '0';
```

Gli array possono essere pluridimensionali: si pensi ad una tabella, ed in questo caso vengono definiti così:

```
Tabellaxy : array [1..60,1..60] of Real;
```

Si costruisce cioè una tabella 60*60 di dati di tipo reale.

## ⇨ OPERATORI LOGICI:

= uguale
<> diverso
< minore
> maggiore
<= minore o uguale
>= maggiore o uguale
**not** negazione della condizione

## ⇨ STRUTTURE DI CONTROLLO:

Nel Pascal sono presenti alcune strutture di controllo:

**IF** *condizione* **THEN**
Begin
*eseguisevera*
End
**ELSE**
Begin
*eseguisefalsa*
End;

Da notare che le istruzioni che precedono **Else** non terminano con il punto e virgola, questo perchè il Pascal considera l' **Else** una continuazione del comando **If**.

**REPEAT**
Begin
*istruzioni*
End;
**UNTIL** *condizione*
Questo ciclo ripete *istruzioni* fino a quando la *condizione* non diventi vera.

**WHILE** *condizione* **DO**
Begin
*istruzioni*
End;
Questo ciclo fa in modo che fino a quando la *condizione* è vera esegua *istruzioni*.

La differenza tra REPEAT UNTIL e WHILE DO consiste che il ciclo REPEAT viene eseguito almeno una volta anche se la condizione non era soddisfatta, mentre WHILE DO verifica prima la condizione, e se è falsa non esegue *istruzioni*

**CASE** *variabile* **OF**
**1 :** *esegui1*
**2..5 :** *esegui2*
**6:** *esegui6*
**ELSE** *eseguielse*
**End;**
Questa istruzione è una alternativa alle istruzioni Else If multiple, a seconda del valore assunto da *variabile* esegue una delle istruzioni presenti dopo le etichette, quindi ipotizzando che *variabile* sia uguale a 1 questo comando fa si che si esegua l'istruzione *esegui1*, mentre se *variabile* ha un valore compreso tra 2 e 5 verrà eseguita l'istruzione *esegui* 3 .Se e fuori dal range, esegue *eseguielse*. *eseguix* deve essere racchiuso tra un Begin ed un End.

**FOR** *variabileintera* **:=** *inizio* **TO** *fine* **DO**
Begin
*istruzioni*
End;
Ciclo che ripete per *fine-inizio* volte *istruzioni.* Questo ciclo viene sviluppato quando si vuole eseguire un gruppo di istruzioni per un determinato numero di volte. *variabileintera* deve essere di tipo Integer o derivati.

### ⇨ COMANDI DI LETTURA E SCRITTURA:

**write(argomento);** scrive *argomento* e rimane sulla stessa riga
**writeln(argomento);** scrive *argomento* e va a capo

**read(variabile);** aspetta un input dall'utente e lo assegna a *variabile*
**readln(variabile);** come sopra, solo che va a capo

### ⇨ COMANDI PER LA GESTIONE DEI FILE:

La gestione dei file su disco è uno dei principali punti di forza di un moderno personal computer e di un moderno linguaggio di programmazione.
Definiamo cosa si intende per FILE: Serie di caratteri (o, più correttamente byte) immagazzinati su un disco o altro supporto. Un file può contenere programmi, database, immagini grafiche etc.
In Pascal la gestione dei file è realizzata con i seguenti comandi:

**ASSIGN(FileTxt,'TESTO.DAT')**; associa a *FileTxt* il file su disco TESTO.DAT
**RESET(FileTxt);** prepara *FileTxt* per la lettura
**REWRITE(FileTxt);** prepara *FileTxt* per la scrittura
**CLOSE(FileTxt);** chiude *FileTxt* e memorizza le modifiche
**APPEND(FileTxt);** riapre *FileTxt* per una ulteriore scrittura
**CLOSE(FileTxt);** chiude il file e memorizza i dati

La variabile FileTxt deve essere dichiarata come file di testo, ovvero nella dichiarazione delle variabili deve essere presente l'istruzione: **FileTxt :Text;** oppure deve essere dichiarata come **File of *tipodati*** .

Le operazioni di scrittura e lettura vengono fatte con le istruzioni WRITE e READ con questa particolare sintassi:

```
Begin;
 Assign(FileTxt, 'TESTO.DAT');
 Assign(File2, 'TESTO.TXT');
 Reset(FileTxt);
 Rewrite(File2);

 While Not Eof(FileTxt) Do
  Begin
   Readln(FileTxt,s1,s2,s3)
   s3:=' - - - - '
   Write(File2,s1,s2,s3)
  End;

Close(FileTxt);
Close(File2);

End.
```

In questo semplice esempio vengono aperti due file su disco, uno in scrittura e l'altro in lettura, si legge (istruzione Readln) il contenuto del primo file, lo si modifica, ed infine viene scritto (istruzione Writeln) il secondo file.
La condizione **Not Eof(FileTxt)** indica che devono essere eseguite le istruzioni racchiuse tra il Begin e l' End fino a quando non si è raggiunta la fine del file (Eof = End of file, fine del file). Da notare che si utilizza While Do e non Repeat Until per i motivi prima descritti; il ciclo Repeat Until in casi come questo genererebbe un errore.

### ⇨ ALCUNE FUNZIONI PREDEFINITE:

**UPCASE(*ce*)** questa funzione restituisce Il carattere contenuto nella variabile *ce* in formato maiuscolo (richiede che *ce* sia una variabile di tipo char e abbia un valore compreso tra 'a' e 'z'.

**CONCAT(*va1*,*va2*,*va3*)** questa funzione restituisce come valore il concatenamento delle variabili ***va1*, *va2*, *va3***: ovvero unisce il contenuto della variabile ***va1***, con il contenuto della variabile ***va2*** e con il contenuto della variabile ***va3***. Esempio:

```
s := 'AAA';
s2 := 'CCC';
s3 := ' ALTRO';
tutto :=Concat(s,s1,s2);
```

Il valore di ***tutto*** sarà: 'AAACCC ALTRO'. Le variabili ***va1, va2, va3*** devono essere di tipo stringa o carattere.

**LENGTH(*var*)** restituisce il numero di caratteri che compongono la variabile ***var***

**DELETE(st,pos,n)** cancella ***n*** caratteri dalla stringa ***st*** a partire dalla posizione ***pos***.

**INSERT(st1,st2,pos)** inserisce nella stringa ***st2*** la stringa ***st1*** a partire da ***pos***.

**COPY(st,pos,n)** restituisce una sottostringa di ***n*** caratteri, estratta dalla stringa ***st***, a partire dalla posizione ***pos.***

**EOF(file)** funzione di tipo booleano: vale TRUE se il file è finito

**ODD(n)** funzione di tipo booleano: vale TRUE se l'intero ***n*** è dispari

## ⇨ OPERATORI BOOLEANI:

| X | Y | NOT X | OR X+Y | AND XY | XOR X+Y |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |

Dove 1 indica la condizione di vero (*true*), e 0 quella di falso (*false*).
Questi operatori vengono usati in istruzioni che richiedano una condizione, e vengono interpretati nel modo seguente:
AND come congiunzione 'e' ad esempio se piove e vado a piedi prendo l'ombrello
OR come scelta es: se piove o nevica prendo l'ombrello
NOT come negazione 'non' es: se non c'è il sole prendo l'ombrello
XOR come or esclusivo es: se piove o se nevica prendo l'ombrello, se piove e nevica contemporaneamente non prendo l'ombrello.
Per chiarimenti vedere la tabella della verità sopra riportata.

## ⇨ STRUTTURE DINAMICHE DI DATI:

Una struttura di dati è *statica* quando la sua dimensione rimane fissa per tutto il periodo in cui viene usata. I tipi ARRAY e RECORD del Pascal ci permettono di definire strutture statiche. Invece le strutture dinamiche di dati sono particolari strutture che durante l'esecuzione del programma cambiano la loro dimensione.
Il Pascal permette di gestire strutture dinamiche di dati tramite l'utilizzo di speciali tipi di dato chiamati puntatori, ma prima di parlare dei puntatori, si deve chiarire il concetto di memoria di un personal computer.

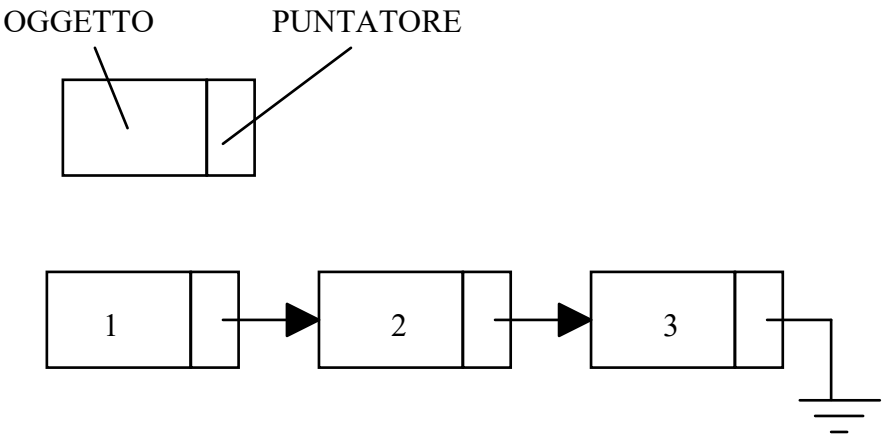
**LA MEMORIA:**

Un personal computer ha più di una memoria, ma in sostanza sono di due tipi: la memoria di massa ("permanente", quella che viene registrata su supporto magnetico) e la memoria di sistema (suddivisa in memoria a sola lettura, ROM, e memoria ad accesso casuale RAM). A noi interessa capire il funzionamento di quest'ultima .

La memoria ram è costituita da un insieme di circuiti integrati in un unico chip, la cui funzione è quella di immagazzinare dati. La ram può essere quindi vista come un insieme ordinato di piccole scatole. Ognuna di queste scatole può contenere una singola informazione, ed è l'utente che decide cosa devono contenere queste scatole.
Al fine avere tutto sotto controllo, si è pensato di disegnare una mappa di queste scatole. Questa mappa altro non è che una serie di numeri progressivi, ognuno dei quali indica il posto dove è situata la scatola, ne indica l' ***indirizzo***.
Quando in Pascal si utilizza una variabile di tipo Integer, Real, String etc., si esegue (viene automaticamente eseguita dal computer) una assegnazione, ovvero si riserva una porzione di memoria per il contenuto della variabile (spazio che varia da tipo a tipo, ma che resta fisso durante l'esecuzione del programma).
Il computer quindi assegna alla variabile da noi dichiarata, non il contenuto, bensì l'indirizzo ove si trova, in gergo si dice che *punta* all'indirizzo, quindi la variabile contiene un puntatore (questo avviene in modo trasparente, l'utente non se ne accorge).
Il puntatore può essere visto quindi come un biglietto dove vi è scritto che la tal cosa si trova nel tal posto.

.... IN PAROLE POVERE ....

In una biblioteca di solito si trova una cassettiera contenente i titoli dei libri autore per autore; la scheda contiene tra le altre cose anche un codice che indica dove è situato (scaffale, posizione etc.) il libro in questione: questo codice è un puntatore.
Ora si sa i libri hanno formati totalmente diversi, hanno un volume ed un ingombro variabile, ma possono sempre essere trovati da un codice; l'utilità di un puntatore è che tramite un unico codice di lunghezza standard, si possono indicare variabili con contenuti diversi.
Una enciclopedia è di solito formata da più volumi, si deve quindi utilizzare un puntatore per ogni volume?
No, ed è questa la comodità e l'utilità dei puntatori: si punta al primo volume (tramite la scheda dell'autore), poi si indica alla fine del primo volume, in uno spazio riservato, che esiste un volume successivo, così sul secondo, sul terzo etc. : questo spazio riservato contiene quindi un altro puntatore.
L'ultimo volume dell'enciclopedia conterrà invece una parola chiave che indica che non vi sono altri volumi.




Nel disegno l'oggetto 1 punta all'oggetto 2 che a sua volta punta all'oggetto 3. L'oggetto 3 è quello finale, quindi il suo puntatore punta a terra (si usa adoperare il simbolo di messa a terra per indicare che il puntatore non punta ad altro e che l'oggetto è quello finale).

## ⇨ I PUNTATORI:

Il puntatore è un tipo di dati semplice (come Integer, Real, etc.) ma non è un identificatore standard. I puntatori si dichiarano così:

```
TYPE
 link =  ^oggetto;
```

Si può notare che per indicare che link è un puntatore che punta ad un oggetto, si è utilizzato il simbolo ' ^ '.
Se si vuole indicare che il puntatore non punta a nulla, simbolo di messa a terra, basta fargli assumere valore **NIL**. NIL è una parola riservata del Pascal che indica appunto che il puntatore è "messo a terra".

La figura rappresenta quella che viene chiamata *lista concatenata*. Per ottenere in Pascal una lista concatenata si deve innanzitutto costruire una struttura di dati che contenga al suo interno un puntatore. Queste strutture dinamiche di dati possono essere costruite come PUNTATORE, come ARRAY o come RECORD.
Si esclude da subito il tipo puntatore: si sa che questo non può contenere informazioni diverse dal valore del puntatore. Rimangono quindi solamente record ed array, ma questi ultimi sono sconsigliati perché le informazioni devono essere tutte dello stesso tipo. Si utilizza quindi il RECORD, ora rimane un unico dubbio: si dichiara prima l'oggetto o prima il puntatore?
Il problema è gia risolto, perché la regola sintattica del Pascal vuole che si definisca prima il puntatore e poi l'oggetto a cui puntare.
Esempio:

```
TYPE
  puntat = ^oggetto
 oggetto = RECORD
      dato : tipodato;
      pun: puntat;
     End;
```

Si nota che ***oggetto*** contiene oltre al ***dato*** il puntatore per il dato seguente ***pun***.
Ulteriori chiarimenti si possono trovare in un qualsiasi libro di Pascal o di programmazione sotto la voce Strutture dinamiche di dati, Liste concatenate, Puntatori, etc.

## ⇨ ESEMPI DI PROGRAMMI IN PASCAL:

Ecco quindi alcuni esempi di procedure, funzioni e programmi completi in Pascal.

**Program pigreco;**

```
Var
   x,y : Longint;
   p   : Real;

Begin
 p:=0;
 x:=1;
 y:=3;
  Repeat
    Begin
      p := p + (4/x) - (4/y);
      writeln(x,p);
      x := x+4;
      y := y+4;
    End;
  Until (x > 1000000);
End.
```

Questo programma calcola il valore di pigreco tramite la serie:
***pigreco = (4/1) - (4/3) + (4/5) - (4/7) .....***

Quando x vale 145653 il valore di pigreco è 3.14157892224102
Quando x vale 938101 il valore di pigreco è 3.14159053561161
Quando x vale 999997 il valore di pigreco è 3.14159064819978

Ricordo che la costante pigreco di una calcolatrice tascabile vale 3.14159265359

**Program potenza(metodo_non_ricorsivo);**

```
Var
 a,b,x,y : Real;

Begin
 write('Base ');
 readln(x);
 write('Esponente ');
 readln(y);
 a := x;
 b := y;
 Repeat
  Begin
   a := a*x;
```

```
   b := b-1;
  End;
 Until (b=1);
 writeln('Risultato ',a);
End.
```

**Program potenz(metodo_ricorsivo);**

```
Var
 z,x,y : Real;
Function potenza(a,n : real) : Real;
 Begin
  if (n > 0) then potenza := a * potenza(a,n-1)
     else potenza :=1;
 End;
Begin
 write('Base ');
 readln(x);
 write('Esponente ');
 readln(y);
 z := potenza(x,y);
 writeln('Risultato ',z);
End.
```

Questi due programmi calcolano la potenza ennesima di un numero, il primo con un ciclo di ripetizione, il secondo con una procedura ricorsiva: si nota infatti che la funzione *potenza* richiama se stessa con dei parametri modificati.

**Program Diario(gestisce_appunti_su_disco);**

```
Var
 fi : text;
 frase : String;
 a : Integer;
 ko : Boolean;
Begin
ko:=true;
assign(fi,'diario.dat');
Repeat
 Begin
  writeln;
  writeln('Agenda su disco - si prega di scegliere:');
  writeln;
  writeln('1. Aggiungere frasi all''agenda');
  writeln('2. Leggere l''agenda');
  writeln('3. Uscire dal programma');
  writeln;
  readln(a);
 case a of
  1:
   Begin
    frase:='';
    append(fi);
```

```
    writeln;
    writeln('Scrivi quello che vuoi.   fine    per uscire');
    while (frase<>'fine') do
     Begin
      readln(frase);
      if (frase<>'fine') then writeln(fi,frase);
     End;
    close(fi);
   End;
  2:
   Begin
    frase:='';
    reset(fi);
    writeln('----------------------------------');
    while not eof(fi) do
      Begin
       readln(fi,frase);
       writeln(frase,'.');
      End;
    close(fi);
    writeln('----------------------------------');
   End;
  3:
   Begin
    ko:=false;
    writeln('FINE');
   End;
  else
    Begin;
    writeln;
      writeln('Dato immesso sbagliato .... digitare un numero da 1 a 3');
    writeln;
      ko:=true;
   End;
 End;
  End;
  Until (ko=false);
End.
```

Questo programma gestisce uno speciale diario su disco. Dopo l'esecuzione si trova sul disco il file DIARIO.DAT, leggibile con un comune text-editor, che contiene le frasi da noi scritte. Questo programma è un esempio dell' utilizzo dei cicli REPEAT UNTIL, WHILE DO, CASE OF, della gestione dei file e delle combinazioni tra questi

Si può notare anche l'indentazione data al programma e l'utilizzo di Begin e End, questo per facilitare la lettura e comprensione del listato.