Programmare in Pascal

Il **Pascal** è forse uno tra i più didattici fra i linguaggi di programmazione. La sua semplicità deriva dal fatto che è molto simile all'inglese. Per questo motivo è utilizzato diffusamente come linguaggio introduttivo alla programmazione. Questo però non significa che abbia poche potenzialità. Anzi, con la sua variante ad oggetti, Delphi, è estremamente diffuso in vari ambiti.

Storicamente, ogni volta che si deve presentare un linguaggio di programmazione, si utilizza il mitico *Hello World*. In pascal è piuttosto semplice da realizzare.

```
program HelloWorld;
begin
     writeln(‘Hello world!’);
end.
```

- La prima riga indica il nome del programma. La parola *program* è una parola riservata del linguaggio. Non può quindi essere usata come nome di variabile o di funzione. Alla fine di ogni istruzione bisogna inserire il carattere *;* per indicare al compilatore che il comando è concluso.
- La seconda riga indica l'inizio del programma vero e proprio. Ogni blocco di codice deve essere inserito fra le parole riservate begin e end.
- La terza riga contiene la nostra prima vera istruzione. I comandi di input-output li ritroveremo in seguito. Per ora basti sapere che questo comando permette di mostrare sullo schermo i caratteri inseriti fra gli apici.
- L'ultima riga rappresenta la conclusione del programma. La parola riservata *end* posta alla fine del programma deve essere sempre seguita dal punto.

Primi concetti sulla sintassi

Da questo primo esempio possiamo ricavare alcune semplici indicazioni sulla sintassi di un programma in pascal.

- esistono delle **parole riservate**, che fanno parte della grammatica del linguaggio Pascal, che possono essere usate solo con lo scopo prefissato.
- ogni programma è composto da **istruzioni**, semplici o composte.
- le istruzioni vengono eseguite dalla prima all'ultima in **sequenza**.
- le istruzioni composte vengono detti **blocchi**. Vengono individuati da una coppia di parole chiave: *begin* indica l'inizio del blocco, *end* la fine.
- il blocco può essere considerato come un'unica istruzione
- ogni istruzione è separata da un carattere speciale: *;* il punto e virgola.
- esistono degli identificatori ovvero nomi arbitrari per variabili, costanti, procedure e funzioni.

Le variabili

Un programma come quello appena fatto può essere divertente e interessante, ma offre ben poche possibilità di sviluppo. Quasi tutti i tipi di programmi richiedono l'uso di calcoli e di “cose da ricordare”. Per questo motivo è necessario introdurre il concetto di variabile. Una variabile, può essere paragonata ad una casella di una certa dimensione nella quale si possono inserire e/o leggere dati mentre il programma è in esecuzione. In pratica le variabili sono delle etichette che noi diamo ad un'area di memoria, che verrà usata per depositare dei dati. La quantità di memoria riservata dipende dal tipo di variabile che andiamo a dichiarare, e dal particolare compilatore che usiamo. Il compilatore pascal ha la necessita di conoscere l'uso e lo scopo di tutte le etichette che incontra durante la fase di compilazione, è indispensabile quindi dichiarare esplicitamente, in particolari punti del programma, le variabili e altre *cose* che vedremo. L'area di dichiarazione delle variabili inizia con la parola chiave *var*.
Il concetto risulta abbastanza chiaro con un esempio pratico.

```
program VariabiliVarie;
var
    n:integer;
    r:real;
begin
     n:=3;
     r:=sqrt(n);
     n:=5;
     writeln(n);
     writeln(r);
end.
```

Il programma in sè è estremamente sciocco, ma ci permette di osservare come le variabili vengano utilizzate in Pascal. Analizzamo come al solito le singole righe.

- La prima riga è, come prima, la dichiarazione che si sta facendo un programma e non altro (più tardi si vedrà cosa altro si può fare)
- Nella seconda e nella terza riga vengono dicharate le variabili. La dichiarazione ha sempre la forma

```
var
    nomevariabile : tipo;
    nuovavriabile : tipo;
```

Come dicevamo il Pascal non permette di dichiarare durante il programma variabili aggiuntive, perciò è necessaria una buona progettazione teorica del programma per non trovarsi a dover correggere molti errori in corso di compilazione.

Altri concetti di sintassi

Abbiamo visto finora due esempi. Possiamo notare che entrambi riflettono anche uno stile di scrittura del codice, che diciamo *stile di formattazione*. Notiamo che il codice presenta varie rientranze. Queste hanno lo scopo di rendere più semplice la vita del programmatore. Consentono di individuare a colpo d'occhio un blocco, semplicemente osservando l'andamento delle rientranze. Per il compilatore tutto ciò è ininfluente, in quanto lui individua i blocchi tramite le parole chiavi e le istruzioni le trova separate da un carattere apposito.
Se provate a compilare le seguenti righe otterrete lo stesso programma visto nell'esempio

precedente. Per il compilatore non ci sono differenze, mentre per il programmatore che dovrà estendere o correggere questo codice le cose si complicano.

```
program VariabiliVarie; var n:integer; r:real; begin
n:=3; r:=sqrt(n); n:=5; writeln(n); writeln(r); end.
```

Ma quali sono i tipi di variabile utilizzabili?

Integer

Il tipo più utilizzato è in genere il tipo *integer*. Corrisponde agli interi, anche se non proprio a tutti. Essendo il computer una macchina limitata, non è possibile fare somme fino all'infinito. Infatti con alcuni semplici programmi si può osservare che esiste un limite superiore per gli interi, oltre al quale si ricade nel limite inferiore, come se fosse una circonferenza. Il fenomeno dello sforamento del limite massimo è detto *Overflow*. Purtroppo Pascal, almeno le versioni più arcaiche, ne soffre spesso.

Ogni tipo di dato ha delle operazioni permesse. In questo caso le operazioni che danno come risultato un integer sono:

- + permette di sommare due valori di tipo integer.
- - permette di calcolare la differenza fra due valori integer.
- * permette di moltiplicare due valori, ottenendo sempre un integer.
- *div* permtte di calcolare la parte intera del rapporto fra due interi.
- *mod* permette di calcolare il resto della divisione fra due interi.

Le funzioni matematiche che danno come risultato un intero sono, invece

- *abs(n)* che calcola il valore assoluto del numero intero *n*
- *round(n)* che arrotonda qualunque numero all'intero più vicino ad *n*
- *trunc(n)* che tronca il numero all'intero minore di *n*
- *sqr(n)* ne calcola infine il quadrato.

Proviamo a fare un programma che utilizzi alcune operazioni. Per ora ci limiteremo a fare semplici assegnamenti di variabili all'interno del programma stesso. Presto troveremo il modo di dinamicizzare le cose...

```
program Pari;
var n:integer;
begin
     n:=5;
     if (n mod 2 = 0) then write (“pari”)
                      else write (“dispari”);
end.
```

Il programma presenta alcuni costrutti che non abbiamo ancora visto, ma la cui intepretazione è semplice. Analizziamo come al solito riga per riga.

- Come al solito la prima riga contiene la dichiarazione del titolo.
- Dichiarazione di una sola variabile *n* di tipo intero.
- Inizia il programma.

- Assegniamo a *n* il valore 5.
- Inizia un costrutto condizionale. Letto “alla lettera” significa “se il resto della divisione fra n e 2 è uguale a zero, allora scrivi *pari* altrimenti scrivi *dispari*”. Da notare, lo rivedremo e faremo alcune considerazioni, che prima di *else* non va **mai** messo il “;”.

Char

Frequentemente si utilizzano invece degli *integer* i *char*, per rappresentare numeri interi piccoli. Infatti i char rappresentano il range di numeri interi che va da 0 a 255 - corrispondente ai codici ASCII - e permettono la visualizzazione nei due formati: intero, ascii. Le operazioni possibili sono le stesse degli integer, ma ci sono due funzioni in più che permettono la trasformazione da intero a carattere e viceversa. Queste funzioni sono

- *chr(x)* che permette di trasformare l'intero *x* nel corrispondente carattere
- *ord(x)* che permette di trasformare il carattere *x* nel corrispondente intero ASCII

Un semplice esempio di uso potrebbe essere la stampa della tabella dei codici ascii.

```
program ASCII;
var n:integer;
begin
     for n:=0 to 255 do
     writeln(n, ' ==> ', chr(n));
end.
```

Come al solito analizziamo riga per riga.

- Dichiarazione del programma,
- Definizione della variabile *n* come char
- Inizio del programma
- Il costrutto *for...to...do* non l'abbiamo ancora visto, ma facendo come prima la traduzione letterale, possiamo intuirne il significato: *Per n che va da 0 a 255 fai...* Questo tipo di costrutto, detto *ciclo*, verrà comunque affrontato in seguito.
- scriviamo su schermo il numero *n* seguito dai caratteri `==>` e poi dal corrispondente carattere ASCII.

Real

Le variabili di tipo real corrispondono ai numeri reali, ovvero i numeri con la virgola. Anche qui, come per gli integer, dobbiamo dare delle limitazioni. Essendo, come già sottolineato un computer una macchina limitata e discreta, inevitabilmente non può superare una certa precisione nel calcolo con i numeri reali. Per questo motivo spesso nelle strutture condizionate si devono cercare di utilizzare stratagemmi per evitare problemi.

Le operazioni di base possibili sembrano meno di quelle possibili con char e integer, ma attraverso l'utilizzo della libreria matematica aumentano in modo incredibile. Comunque le operazioni basilari sono:

- + permette di sommare due valori di tipo real.
- - permette di calcolare la differenza fra due valori real.
- * permette di moltiplicare due valori, ottenendo sempre un real.
- / permette di calcolare il rapporto fra due reali.

Boolean

L'algebra booleana è fondamentale nell'informatica. Questa permette infatti di fare calcoli sulla veridicità o falsità di una affermazione. Le variabili booleane servono proprio a definire concetti quali vero (*true*) e falso (*false*). Le operazioni possibili sono diverse da quelle possibili per gli altri tipi di variabile:

- *and* che corrisponde al simbolo matematico $\wedge$ e al concetto di *e contemporaneamente*
- *or* che rappresenta $\vee$ e *oppure*
- *not* operatore che corrisponde alla negazione
- *xor* che corrisponde matematicamente all'*aut*, ovvero $\dot{\vee}$

L'utilizzo di variabili Boolean è in genere limitato all'analisi di particolari aspetti dell'input da utente, o all'utilizzo come 'flag' nella programmazione più avanzata.

String

Le variabili string sono variabili che possono contenere una stringa alfanumerica di caratteri.

Tabella: Operazioni ed operatori

Operatore	Operandi	Risultato
+	real o integer o char	real o integer o char
-	real o integer o char	real o integer o char
*	real o integer o char	real o integer o char
/	real o integer o char	real
mod	integer o char	integer

div	integer o char	integer
sqrt	real o integer o char	real
abs	real o integer o char	real o integer o char
trunc	real o integer o char	integer

Input e Output

Finalmente ci occupiamo del grande problema dell'input-output. Gran parte delle funzioni di input-output sono contenute nella libreria standard di pascal. Queste funzioni sono tipicamente

- *write()* e *writeln()* che permettono, come già visto in precedenza, di stampare su schermo il contenuto delle parentesi.
- *read()* e *readln()* che permettono di leggere l'input dell'utente da tastiera inserendo fra le parentesi il nome della variabile in cui vogliamo salvare il dato.

La differenza fra la versione con e senza *ln* è che la prima va a capo prima della scrittura o dell'immissione, la seconda versione no.

Un semplice esempio basato sull'uso delle variabili e dell'input-output potrebbe essere un modo per personalizzare un programma, salvando in una stringa il nome dell'utente per inserirlo nelle domande.

```
program username;
var name: String[50];
uses crt;
begin
     clrscr;
     writeln('Inserisci il nome');
     readln(name);
     clrscr;
     writeln('Benvenuto, ', name);
     readkey;
end.
```

Analizziamo come al solito riga per riga.

- Dichiarazione del nome
- Dichiarazione della variabile *name* di tipo Stringa di dimensione 50
- Dichiarazione delle librerie necessarie. Di questo argomento riparleremo a breve, ma sappiate che almeno due dei comandi utilizzati di seguito sono dipendenti da questa libreria.
- Inizio programma
- Pulizia dello schermo. Questa è la prima (e forse la più usata) delle due funzioni che dipendono da *crt*.
- Scrittura della stringa ‘Inserisci il nome’

- Lettura della stringa corrispondente al nome ed inserimento di questa stringa letta da tastiera nella variabile *name*
- nuova pulizia dello schermo
- Scrittura del messaggio di benvenuto.
- Nuova funzione dipendente dalla libreria *crt* che permette di leggere un solo carattere qualunque da tastiera. Vedremo che questa funzione e molto utilizzata per risolvere un problema riguardo all'esecuzione dei programmi compilati finora.

Librerie

Una libreria non è altro che un file contenete molte funzioni e variabili che possono essere utilizzate una volta incluso il file all'interno del programma. Questo da un punto di vista teorico è molto utile perchè permette di avere programmi molto brevi, ma da un punto di vista di sviluppo è piuttosto negativo, poichè una volta compilati anche programmi molto semplici possono essere estremamente "grandi" in dimensioni fisiche (ovvero in kByte).

Il Pascal utilizza come libreria standard *Turbo.tpl*. Questa contiene tutte le funzioni di base. Non viene mai invocata, perchè lo fa in automatico il compilatore alla compilazione. Ma come abbiamo già visto esistono molte altre librerie.

Dall'ultimo programma osservato si nota come deve essere fatta la dichiarazione delle librerie:

```
uses nomelibreria, nomelibreria, ...;
```

uses è una parola riservata, quindi non può essere usata come nome di variabile o funzione.

crt

La libreria di gran lunga più usata è sicuramente *crt*, con tutte le funzioni relative all'estetica in ambito DOS. Ma non abbiamo ancora affrontato il problema dell'utilizzo di librerie all'interno dei nostri programmi.

Probabilmente avrete notato che eseguendo i programmi fatti finora l'esecuzione terminava istantaneamente, non appena il risultato veniva mostrato sullo schermo. Questo può essere abbastanza scomodo. Ma anche per questi problemi aiuta la libreria CRT, con la funzione *readkey();* che, come abbiamo già visto, permette di leggere il primo carattere che viene immesso da tastiera.

funzioni di crt

```
clrscr;
```

Questa funzione permette di pulire lo schermo e di posizionare il cursore in alto a sinistra in posizione (1,1)

```
cursoroff;
cursoron;
```

Queste due funzioni permettono di spegnere o accendere il cursore.

```
gotoXY(posX, posY);
```

Questa procedura permette di posizionare il cursore in una posizione precisa sullo schermo.

```
sound(hz);
nosound;
```

La funzione *sound* fa emettere alla macchina un suono alla frequenza *hz*. Per fermare questo suono bisogna assolutamente utilizzare la procedura *nosound*

```
delay(time);
```

Questa funzione fa sì che il sistema si fermi in pausa per un certo numero di millisecondi definito da *time*. L'uso della procedura <pre>delay</pre> è tipicamente usata quando si fa uso di suoni. La struttura d'uso generalmente è la seguente:

```
sound(440);
delay(1000);
nosound;
```

Il risultato è un LA di durata un secondo. La funzione delay() è però basata sul clock del processore per calcolare il tempo, quando è stato creato il Pascal ovviamente i processori non avevano la potenza di oggi. La funzione delay() è quindi difficile da controllare ma se si vogliono ottenere ritardi definiti con bassa precisione cronometrica si possono inserire più delay() di seguito o sfruttare i cicli.

```
textcolor(numero);
textbackground(numero);
```

Permettono di cambiare colore al testo e allo sfondo rispettivamente. Il codice di colore può anche essere il nome in inglese del colore. Questo perchè nella libreria crt sono definite anche alcune costanti fra cui proprio quelle relative ai colori. Fra l'altro si può utilizzare anche la costante blink, per creare testo intermittente. Questo uso deve essere fatto così:

```
...
textcolor(white+bink);
textbackground(blue);
...
```

Tabella: Le frequenze delle note

Nota	Frequenza	Formula
Do	262	$440 \cdot (2^{\frac{1}{12}})^{-9}$
Do#/Reb	277	$440 \cdot (2^{\frac{1}{12}})^{-8}$
Re	294	$440 \cdot (2^{\frac{1}{12}})^{-7}$

Re#/Mib	311	$440 \cdot (2^{\frac{1}{12}})^{-6}$
Mi	330	$440 \cdot (2^{\frac{1}{12}})^{-5}$
Fa	349	$440 \cdot (2^{\frac{1}{12}})^{-4}$
Fa#/Solb	370	$440 \cdot (2^{\frac{1}{12}})^{-3}$
Sol	392	$440 \cdot (2^{\frac{1}{12}})^{-2}$
Sol#/Lab	415	$440 \cdot (2^{\frac{1}{12}})^{-1}$
La	**440**	$440 \cdot (2^{\frac{1}{12}})^{0}$
La#/Sib	466	$440 \cdot (2^{\frac{1}{12}})^{1}$
Si	494	$440 \cdot (2^{\frac{1}{12}})^{2}$

La frequenza di riferimento è quella del La, ovvero 440 Hz. Per ottenere le altre ottave basta raddoppiare o dimezzare le frequenze della nota corrispondente di questa ottava centrale.

Strutture condizionate

In Pascal esistono due metodi per porre condizioni su certi eventi. Innanzitutto

```
if condizione  then istruzione
       else istruzione;
```

che come abbiamo già visto permette la scelta fra un caso vero e un caso falso e in base a questo esegue il contenuto del *then* o dell'*else*. Spesso il ramo in else non serve.

Il secondo metodo permette di fare scelte più complesse

```
case selettore of
     caso1 : istruzione;
     caso2 : istruzione;
     caso3 : istruzione;
     ...
     else: istruzione;
end;
```

Con il costrutto case, il valore del selettore viene confrontato con il valore di ogni singolo caso, quando viene trovato un caso che eguaglia il selettore l'istruzione che segue viene eseguita, poi il controllo passa alla prima istruzione dopo il costrutto case. Se nessun caso soddisfa il selettore, viene eseguita l'ultima istruzione, quella individuata dalla parola **else**. Occorre fare attenzione che il costrutto case termina con un **end**.

Facciamo un esempio del costrutto case:

```
case Contatore of
  1, 2, 50, 210: scrivi_qualcosa(contatore);
  30, 60..80 :
     begin
       conta_x := conta_x+1;
       fai_qualcosa(z);
       fai_qualcosa(y);
     end;
end;
```

alcune considerazioni sull'esempio proposto:

- abbiamo omesso la parte facoltativa **else**
- sono state scritte delle liste di possibili casi
- è stato usato un range di valori x..y
- è stata usata un'istruzione composta al posto di una semplice

Strutture Iterative

In Pascal abbiamo diversi tipi di struttura iterativa (in genere vengono detti "*cicli*"):

For

Questa struttura, ci permette di ripetere una determinata serie di istruzioni per un numero finito di volte. La sua sintassi e':

```
FOR Contatore:=valore_iniziale TO Valore_finale DO
   Istruzione;
```

Se le istruzioni dovessero essere piu' di una, queste andrebbero messe tra 'begin' e 'end;'.

Traducendo tutto in italiano, si avrebbe:

```
PER Contatore:=Valore_iniziale FINO A Valore_Finale ESEGUI
   Istruzione;
```

Per Contatore s'intende una variabile (di solito vengono usate le lettere I,L,M,N,J,K), per Valore_Iniziale il valore da cui deve cominciare la conta (non per forza 1), e infine Valore_Finale rappresenta il valore per il quale il ciclo terminerà.

While do

Questo ciclo corrisponde ad una struttura decisionale a condizione in testa.

La sua sintassi e':

```
WHILE condizione DO
begin
   ...
end;
```

Tra **begin** ed **and** vanno inserite le istruzioni che canbiano il valore di un qualche cosa affinche' la condizione si avveri. Se vi e' una sola istruzione, **begin** e **end** possono essere omessi.

Qualche esempio

```
...
x:=1;
While x=5 Do
  x:=x+1;
...
```

In questo esempio viene eseguito quanto scritto all'interno del ciclo affinchè non viene verificata la condizione, cioè ad x viene sommato 1 fino a che il suo valore non sia uguale a 5.

Repeat Until

Dopo aver parlato di If...Then...Else, e dei cicli For, While Do, chiudiamo il discorso sui cicli con Repeat...Until.

La sua sintassi e':

```
...
Repeat
  istr1;
  ...
  istrn;
Until Condizione;
...
```

In sostanza questo ciclo ripete le istruzioni comprese tra Repeat ed Until Condizione fino al verificarsi della condizione espressa. Da notare che queste istruzioni verranno eseguite almeno una volta.

Compilatori

- A pagamento
  - Borland Pascal

- Open Source
  - Free Pascal
  - GNU Pascal

IDE

- A pagamento
  - Borland Delphi
  - Borland Turbo Pascal

- Open Source
  - Lazarus
  - Dev-Pas
  - FreePascal

Da notare che Borland Delphi e Lazarus sono IDE di Delphi ovvero Pascal orientato agli oggetti, riescono tuttavia a compilare senza problemi il Pascal normale.