*Prof.ssa De Bernardi*

**MANUALETTO DI TURBO PASCAL**
*(tratto da Emagister e Wikipedia)*

## Capitolo 1: **Introduzione ai linguaggi di programmazione**

"*Il computer è una macchina che elabora delle informazioni; in quanto macchina, però, non è capace di gestirle da sé, ma necessita di un operatore esterno che istruisca l' elaboratore e gli insegni come trattare queste informazioni.*"
Questa definizione racchiude in sé il significato della parola programmazione.
In informatica, un **linguaggio di programmazione** è un linguaggio formale, dotato (al pari di un qualsiasi linguaggio naturale) di un *lessico*, di una *sintassi* e di una *semantica* ben definiti, utilizzabile per il controllo del comportamento di una macchina formale o di una implementazione di essa (tipicamente, un computer) ovvero per la programmazione di questa attraverso la scrittura di un programma sotto forma di codice.

Un computer ovviamente non può risolvere qualunque tipo di problema, ma può risolvere solamente problemi traducibili in **algoritmi**, cioè serie finite di istruzioni semplici che possano condurre ad una soluzione calcolabile o valutabile in modo esatto.

Un algoritmo si può definire come un *procedimento* che consente di *ottenere* un *risultato* atteso eseguendo, in un determinato ordine, un insieme di *passi semplici* corrispondenti ad azioni scelte solitamente da un insieme finito di passi. Il termine deriva dal nome del matematico e filosofo arabo *Abū Ja'far Muhammad ibn Mūsā al-Khwārizmī*, latinizzato in *Algoritmi*, che si ritiene essere uno dei primi autori ad aver fatto riferimento esplicitamente a questo concetto nel libro *GHf - ghtr adouf* (*Libro della matematica orientale*). La parola *algebra* deriva sempre dall'arabo *al-giabr (ridurre le fratture, riunire i pezzi)* probabilmente, in particolare, dal libro *Algiabar wal-muqabala* dello stesso matematico *'l Khuwarizmi* .

Da questa definizione si evincono le due proprietà fondamentali dell'algoritmo:

- la sequenza di istruzioni è finita (*finitezza*);
- essa deve portare ad un risultato (*effettività*).

### Concetti base

Tutti i linguaggi di programmazione esistenti possiedono (almeno) questi due concetti chiave:

- Variabile: un dato o un insieme di dati, noti o ignoti, già memorizzati o da memorizzare; ad una variabile corrisponde sempre, da qualche parte, un certo numero (fisso o variabile) di locazioni di memoria che vengono *allocate*, cioè riservate, per contenere i dati stessi. Molti linguaggi inoltre attribuiscono alle variabili un tipo, con differenti proprietà (stringhe di testo, numeri, liste, ecc.).
- Istruzione: un comando oppure una regola descrittiva: anche il concetto di istruzione è molto variabile fra i vari linguaggi. A prescindere dal particolare linguaggio però, ogni volta che un'istruzione viene eseguita, lo stato interno del calcolatore (che sia lo stato reale della macchina oppure un ambiente virtuale, teorico, creato dal linguaggio) cambia.

Alcuni concetti sono poi presenti nella gran parte dei linguaggi:

- Espressione: una combinazione di variabili e costanti, unite da operatori; le espressioni sono state introdotte inizialmente per rappresentare le espressioni matematiche, ma in seguito la loro funzionalità si è estesa. Una espressione viene valutata per produrre un valore, e la sua valutazione può produrre "effetti collaterali" sul sistema e/o sugli oggetti che vi partecipano.
- Strutture dati, meccanismi che permettono di organizzare e gestire dati complessi.
- Strutture di controllo, che permettono di governare il flusso di esecuzione del programma, alterandolo in base al risultato o valutazione di una espressione (che può ridursi al contenuto di una variabile, o essere anche molto complessa) (cicli iterativi quali ad esempio *for*, *do*, *while* e strutture condizionali quali ad esempio *if*, *switch-case*).
- Sottoprogramma: un blocco di codice che può essere richiamato da qualsiasi altro punto del programma. In tale ambito quasi tutti linguaggi offrono funzionalità di riuso di codice accorpando cioè sequenze di istruzioni all'interno di funzioni richiamabili secondo necessità all'interno di programmi o all'interno di librerie richiamabili in ogni programma.
- Funzionalità di input dati da tastiera e visualizzazione dati in output (stampa a video).

**Il codice sorgente**

Programmare in un dato linguaggio di programmazione significa generalmente scrivere uno o più semplici file di testo ASCII, chiamato **codice sorgente**.

**Compilazione e interpretazione**

Il codice sorgente, contenente le istruzioni da eseguire e (spesso) alcuni dati noti e costanti, può essere poi eseguito passandolo ad un interprete che eseguirà le istruzioni in esso contenute; oppure può venire compilato, cioè tradotto in istruzioni di linguaggio macchina, cioè una modalità di comunicazione con la macchina basta solamente sul codice binario, da un programma compilatore: il risultato è un file binario 'eseguibile' (codice eseguibile) che non ha bisogno di altri programmi per andare in esecuzione, ed è anche molto più veloce di un programma interpretato.

La compilazione è quindi il processo per cui il programma, scritto in un linguaggio di programmazione ad alto livello, cioè in un linguaggio lontano dal linguaggio macchina e più vicino al nostro linguaggio naturale, viene tradotto in un codice eseguibile per mezzo di un altro programma detto appunto compilatore.

La compilazione offre numerosi vantaggi, primo fra tutti il fatto di ottenere eseguibili velocissimi nella fase di run (esecuzione) adattando vari parametri di questa fase all'hardware a disposizione; ma ha lo svantaggio principale nel fatto che è necessario compilare un eseguibile diverso per ogni sistema operativo o hardware (piattaforma) sul quale si desidera rendere disponibile l'esecuzione ovvero viene a mancare la cosiddetta portabilità.

Esistono quindi due principali tipi di linguaggi con cui programmare una macchina:
i **programmi interpreti** e i **compilatori**.

Alla prima categoria appartengono numerosi programmi tra i quali spicca il BASIC ( Beginners All-Purpose Symbolic Instruction Code = Codice simbolico di istruzioni per tutti gli usi, per principianti) grazie alla sua intuitività.
Il suo funzionamento è semplice: il codice sorgente (quello introdotto dall'operatore esterno) viene tradotto in linguaggio macchina (ASCII) riga per riga ogniqualvolta il programma stesso viene eseguito.
Oltre agli interpreti però, come già detto, esistono anche i compilatori.
PASCAL, FORTRAN( FORmula TRAnslate = Traduzione di formule) e C, sono soltanto alcuni dei più diffusi linguaggi compilati.
I programmi compilatori funzionano nel modo opposto degli interpreti: dato il solito codice sorgente, il compilatore lo trasforma in blocco in codice eseguibile.
Questi due metodi di programmazione sono entrambi validi ma ciascuno offre vantaggi e svantaggi.
I compilatori infatti hanno il vantaggio di essere più veloci rispetto agli interpreti ma sono più soggetti ad errori degli interpreti.

## Capitolo 2: **Introduzione al Turbo Pascal**

In questa sede ci occuperemo estesamente di un particolare tipo di linguaggio di programmazione compilato, il Turbo Pascal.
Il Pascal prende il suo nome dal matematico francese Blaise Pascal che fu il primo ad ideare una macchina calcolatrice: la Pascalina.
Questo linguaggio fu messo a punto nel 1960 da Niklaus Wirth, docente all'università di Zurigo. Esso si diffuse dapprima negli ambienti scientifici ma, successivamente, grazie alla sua versatilità ed alla sua semplicità, si diffuse un po' dappertutto.

Passiamo ora alla parte operativa.
Una volta avviato il TP (Turbo Pascal, N.d.A.) si presenterà un' interfaccia a menu abbastanza intuitiva.
Compariranno vari menu, ma quelli principali sono tre: **File, Run,Compile**.
Il primo è formato da:
**New**, per creare un nuovo documento di lavoro;
**Load**, per aprire lavori già salvati;
**Save**, registra un lavoro.
Nel menu Compile l' unico comando che ci riguarda è Compile, che compila il programma (Cfr. Cap.1).
Spesso può risultare utile conoscere il comando Destination (nel menu Compile) per decidere se compilare il programma nella Ram oppure in un file eseguibile (nome_programma.exe).
Una volta compilato il programma, però, dobbiamo eseguirlo, e perciò è necessario selezionare dal menu Run il comando Run.

## Capitolo 3 **: Struttura generale di un programma**

Un programma scritto in TP consta in tre sezioni principali:
**sezione intestazione, sezione dichiarazioni e sezione esecutiva.**
Nonostante questi terribili nomi esse sono semplicissima comprensione: nella prima sezione si inserisce il nome del programma, nella seconda sezione vengono digitate le variabili e la loro tipologia, ed infine viene scritto il programma vero e proprio.
Per intenderci possiamo fare un banalissimo esempio: possiamo considerare il TP come un teatro in cui si inscena un' opera: il titolo dell' opera è dato dalla sezione delle intestazioni, gli attori che prendono parte all' opera vengono presentati nella sezione dichiarativa, mentre lo svolgimento dello spettacolo è rappresentato dal corpo del programma.
Esistono delle parole, dette "**riservate**", a cui non può essere dato il valore di variabile, esse infatti rappresentano dei comandi che il TP può eseguire.
Tornando all' esempio del teatro, non possiamo chiamare un personaggio SIPARIO oppure SCENA, esse infatti sono parti specifiche del teatro e non possono essere perciò dei personaggi.

## Capitolo 4 : **Abbozziamo un programma**

Ora cominceremo con l' abbozzo di un programmino semplice:
-*Calcolare il triplo di un numero dato*-.
D'ora in poi le parole riservate verranno scritte in maiuscolo.
E' da ricordare che alla fine di qualsiasi istruzione bisogna inserire il punto e virgola (;).
Anche se non è obbligatorio andare a capo dopo di esso, è preferibile farlo per rendere più agevole la lettura e la comprensione del listato.
Inoltre per inserire un commento che non interferisca con l' esecuzione del programma bisogna inserire il commento tra asterischi, e mettere il tutto tra parentesi tonde.
Es.(* Questo è un commento *).

Ecco qui il programma:

```
PROGRAM triplo;                        (* Sezione dell'intestazione *)
Uses wincrt;          (* Sezione della dichiarazione della libreria *)
VAR n:INTEGER;      (* Sezione delle dichiarazioni delle variabili *)
 Begin                           (* Inizio del corpo del programma *)
    clrscr                     (* Istruzione per pulire lo schermo *)
    writeln('Questo programma calcola il triplo di un numero dato');
    writeln;                            (* Istruzioni di scrittura *)
    writeln ('Introduci un numero intero');
    readln(n);                            (* Istruzione di lettura *)
    n:=n*3;                         (* Istruzione di assegnazione *)
    writeln;
    writeln('Il triplo è ' ,n);
    writeln;
    writeln('Premi enter per finire');
    readln;
 end.
```

Nella parte dichiarativa abbiamo comunicato all' elaboratore che n è una variabile (VAR) che nel corso dell'esecuzione potrà avere soltanto un valore INTEGER, cioè di numero intero positivo o negativo.

Esaminando successivamente il listato vedremo che la parte esecutiva è introdotta da BEGIN ( = inizio).
Poi troviamo una nuova parola riservata, READLN ( abbreviazione Read Line = leggi riga).
Essa serve a leggere i dati di input.
La sua sintassi è molto semplice: READLN(nome variabile da leggere).
Inoltre l' istruzione READLN non seguito da alcun parametro, crea un ciclo di attesa che dura finché non viene premuto Invio.
E' d'obbligo inserirla alla fine di un listato, prima dell' END, poiché "congela" lo schermo quando dobbiamo leggere il risultato dell' elaborazione.
Ad esempio, se eseguissimo il programma precedente senza READLN finale, non potremmo esaminare i risultati della nostra operazione, perché il programma si arresterebbe troppo in fretta.
Un'altra fondamentale istruzione è WRITELN ( abbreviazione Write Line = scrivi riga); essa serve a stampare i dati di output sullo schermo.
La sua sintassi è: WRITELN('Stringa di testo').
Se consideriamo l'esempio sopra vedremo che il testo tra apici (') verrà stampato sullo schermo; nel caso in cui WRITELN venga usato senza parametri, l'elaboratore lascerà una riga vuota nell'elaborazione.
Un ulteriore esempio sull'uso di WRITELN: WRITELN(a).
In questo caso verrà visualizzato sullo schermo il valore assunto in quel momento dalla variabile a. Dunque, se a vale 3, allora sullo schermo uscirà il numero 3.
E' possibile anche combinare stringhe e variabili, mettendo una virgola tra il secondo apice e la variabile: WRITELN( ' Io sono ', A);
Infine, alla fine del programma (scusate il gioco di parole), troviamo END seguito da un punto. Esso indica la fine del flusso delle informazioni e il conseguente arresto del programma.

**Posizionamento del cursore**

Quando andiamo a creare un programma in Pascal, quello che decidiamo di far vedere sullo schermo viene automaticamente posizionato a sinistra, così

| | |
|---|---|
| **Write ('ciao');** | **ciao** |

Per far sì che il testo sia posizionato al centro o in altri punti dello schermo possiamo ricorrere all' istruzione **GOTOXY (**colonna,riga**);**. Quando utilizziamo Turbo Pascal, lo schermo viene diviso in righe e colonne come un foglio a quadretti. In termini numerici abbiamo 25 righe e 80 colonne per un totale di 2000 caselle.
Noi possiamo sfruttare queste caselle tramite GOTOXY . Ad esempio:

| | |
|---|---|
| **GotoXY (3,5);**<br>**Write ('ciao');** | **ciao** |

Il testo non è più a sinistra ma in un punto diverso dello schermo. Nel caso in cui vogliamo posizionare il testo esattamente al centro dello schermo si può calcolare la x: **x = (80- lunghezza testo)/2.**

## Capitolo 5 : **I tipi di dati**

Come già visto in precedenza, nel settore dichiarativo di un programma bisogna definire il tipo di variabili che intendiamo utilizzare nel corso del programma stesso.
Esistono vari tipi di variabili, sono raggruppate in questo breve specchietto.

| *Nome tipo* | *Intervallo consentito* | *commenti* |
|---|---|---|
| INTEGER | -32768 / +32767 | numeri interi (positivi e negativi) |
| BYTE | 0 /255 | numeri interi |
| WORD | 0 / 65535 | numeri interi |
| REAL | 2.9x10^-36 / 1.5 x 10^35 | numeri reali (positivi e negativi) |

Questo schema dimostra come ci siano vari tipi di dichiarazione dei dati.
Non è detto, però, che i tipi di dati corrispondano soltanto a cifre; potremmo infatti di fronte a un problema che necessita di dati alfanumerici, cioè composti di numeri e lettere.
In questo caso dovremmo introdurre due nuovi tipi di variabile:

1) STRING[x] : Dichiarando così un tipo di dato assumeremo che quel dato è formato da un numero intero di x caratteri alfanumerici (compresi gli spazi).
Usato senza parametri non definisce la lunghezza della stringa.
Inoltre nel caso in cui si dichiari una lunghezza minore del dato inserito, l' elaboratore non considererà i caratteri che eccedono.
Ad esempio, se dichiarassimo la variabile a di tipo STRING[5] e al momento di inserire i dati digitassimo la parola "folletto", essa risulterebbe troncata in "folle", poiché i caratteri in eccesso non verrebbero considerati nell' elaborazione.

2) CHAR : E' meno usata di STRING, ma qualche volta può risultare utile.
Viene usata per definire variabili da un solo carattere ( eccetto numeri ).

## I) Le strutture iterative: cicli

Il termine iterare deriva dal latino e significa "ripetere".
Questa struttura ( che è presente in tutti i linguaggi programmativi, non solo in Pascal) agisce su una operazione che deve essere ripetuta più volte.

Vi sono tre modi per ripetere un'istruzione o un blocco di istruzioni in Pascal:

- **for…to…do**
- **while…do**
- **repeat …until**

### a) Iterazione enumerativa

**Ciclo FOR …TO…DO**

Come dice la parola stessa esegue un'operazione n volte. In Pascal è indicata dal comando:

```
FOR a =: ni TO nf DO ( * Niente  punto e virgola * )
          istruzione 1;
          istruzione ...;
END;
```

Nel nostro caso `ni` sostituisce il numero intero dal quale si parte a enumerare e `nf` sostituisce il numero fino al quale si continua a enumerare. Dopo questa istruzione di iterazione si inseriscono le istruzioni da ripetere racchiudendo il tutto con un END seguito da un punto e virgola (;), poiché il flusso di informazioni non è del tutto terminato.

E' possibile anche procedere a ritroso, contando alla rovescia da un numero n fino ad arrivare ad un numero t:
FOR a:= n DOWNTO t DO

Viene generalmente utilizzato per ripetere un blocco di istruzioni per un numero N di volte, dove N è conosciuto.

**Esempi:**

Si vuole fare scrivere una frase 10 volte:

```
uses crt;
var I: integer;
begin
      clrscr;
   for I:=  1 to 10 do
      writeln ('La matematica mi piace moltissimo');
      readln;
end.
```

Oppure si vuole far scrivere la potenza ennesima di un numero:

```
PROGRAM potenza;
   USES crt;
   VAR a,b,c,i:INTEGER;
   BEGIN
   WRITELN('Inserisci un numero');
   READLN(A)
   WRITELN('Inserisci esponente');
   READLN(B);
   FOR I:= 1 TO b DO
   c:=a*a;
   END;
   WRITELN('Il risultato della potenza è ' ,c);
   READLN;
   END.
```

**Esercizi:**
-scrivi i primi dieci numeri pari
-scrivi il quadrato dei primi 10 numeri interi

---

**b) Iterazione per vero**

**WHILE…DO**

Esegue le informazioni racchiuse tra WHILE e DO e l' END; ( il punto e virgola indica che l'END è parziale) finché la condizione presa in esame risulta vera ed esce quando la condizione risulta falsa.

```
WHILE a = 3 DO
 istruzione 1;
 istruzione 2;
 istruzione...;
END;
```

Questo semplice esempio spiega come il blocco di istruzioni presente tra WHILE DO e END; venga ripetuto finché si rispetti la condizione a = 3. Nel caso in cui il valore di a fosse stato diverso da 3, infatti, il blocco di istruzioni non sarebbe stato mai eseguito.

(Il blocco di istruzioni che segue “do” va sempre racchiuso tra un “begin” e un “end”.)

**Esempio:**

Scrivere la somma dei primi dieci numeri naturali.

```
uses crt;
var I,S: integer;
begin
      clrscr;
I:=0;             (*inizializzazione del contatore I*)
S:=0;             (*inizializzazione della variabile somma*)
     while I < 10 do
```

```
begin
          I:=I+1;            (*incremento il contatore*)
          S:=S+I;            (*calcolo le somme parziali*)
          end;
     writeln (S);
     readln;
end.
```

**Esercizi:**

- Calcola la somma di una serie di numeri interi immessi da tastiera.
- Sommare i numeri pari compresi tra 10 e 30.

---

**c) Iterazione per falso**

**REPEAT …UNTIL**

Si deve ripetere un blocco di istruzioni <u>fino a che la condizione assegnata risulta falsa</u>; quando diventa vera, l'istruzione non viene più eseguita. In questo ciclo prima viene eseguita l'istruzione, poi si incontra la condizione.

```
REPEAT
      istruzione 1;
      istruzione 2;
      istruzione ...;
UNTIL proposizione;
```

N.B. Ricorda di inserire proposizioni possibili dopo UNTIL altrimenti l'uscita dal ciclo ripetitivo non si avrà mai! ( in gergo si dice che il computer si "impianta" e bisogna riavviarlo)

**Esempio:**

Scrivere i numeri pari fino a 30.

```
uses crt;
var I: integer;
begin
      clrscr;
      I:=0;        (*inizializzazione del contatore I*)
      repeat
             I:=I+2;
             writeln (I);
      until I = 30;   (*condizione*)
             readln;
end.
```

**Esercizi**:

- Scrivi la somma dei primi 10 numeri naturali.

- Scrivi il quadrato di un numero immesso da tastiera per un numero a piacere di volte fino a quando il numero immesso è 0.

## II) Le strutture selettive
==========

A questo punto è lecito domandarci se è possibile operare delle scelte con il TP. La risposta è affermativa. Esistono due tipi di selezione: la selezione binaria e la selezione multipla.

### a) Selezione binaria

E' la più semplice: se consideriamo un numero naturale qualsiasi e ci domandiamo se esso è pari, la risposta potrà avere solo due alternative: Si oppure No.
In TP c'è la possibilità di eseguire una scelta binaria tra due condizioni con il seguente comando:

IF <proposizione1> THEN <istruzione1> ELSE <istruzione2>
che tradotto fa più o meno:
SE <proposizione1> ALLORA <istruzione1> ALTRIMENTI <istruzione2> ]

Questo è quanto serve per operare ( o far operare al computer ) una scelta.
Per chiarire ulteriormente questo importante processo vedrò di presentarvi un esempio.
Costruiamo un programma che calcoli l' area di un rettangolo SE E SOLO SE l' area stessa è minore di un numero n ( per esempio 20) e maggiore di un numero m (per esempio 10);

```
PROGRAM area_rettangolo;
USES crt;
VAR a,b,c: INTEGER;
BEGIN
CLRSCR; (*E' l'abbreviazione di CLeaR SCReen= cancella schermo.
                Pulisce lo schermo.* )
WRITELN('Inserisci base');
READLN(a);
WRITELN('Inserisci altezza');
READLN(b);
c:=a*b;
IF (c<20) AND (c>10) THEN
 (*AND, insieme a OR, sono gli operatori booleani più usati in
  TP. Per approfondimenti consultare l' appendice C *)
BEGIN
WRITELN('Area maggiore di 20 oppure minore di 10.');
READLN;
END ( * L' END prima dell' ELSE senza punto e virgola(;) * )
ELSE
BEGIN
WRITELN(' AREA = ',c);
READLN;
END;
READLN;
END.
```

E' un listato semplice e l' unica osservazione da fare è che se dopo IF...THEN c'è solo un istruzione può essere scritta terminando il tutto con il solito punto e virgola(;).
Tuttavia, se ce ne sono più di una è necessario aprire un altro BEGIN e cominciare un sottoprogramma il quale, però, va terminato con l' END; che, se seguito da ELSE non accetta il punto e virgola(;).

**b) Selezione multipla**

Se domandiamo ad una persona quali sono i piatti che preferisce essa sceglierà fra un' ampia gamma di pietanze fino a scegliere quella che soddisfi i suoi gusti. Infatti, al contrario di prima, la selezione multipla ci fornisce illimitate possibilità di risposta.

Il comando che in TP ci permette di operare un selezione multipla è il seguente:

```
CASE n of
              ( * Se n è stata dichiarata INTEGER * )
     valore 1:BEGIN
                 istruzione1;
                 istruzione2;
                 istruzione...
              END;
              ( * Se n è stata dichiarata CHAR * )
     valore x:BEGIN
                 istruzione1;
                 istruzione2;
                 istruzione...;
              END;
             ELSE
               BEGIN
              istruzione1;
              istruzione2;
              istruzione...;
               END;
            END;
```

Nonostante la sua sintassi spaventosamente ricca, questa struttura è di facilissima comprensione.
Innanzitutto dobbiamo decidere se dichiarare la variabile n come tipo INTEGER oppure CHAR.
Se n è CHAR dovremo usare UNA lettera seguita dai due punti (:) e dal rispettivo sottoprogramma delimitato da BEGIN e END; .
Se invece n è INTEGER dovremo usare un numero naturale al posto di n e mettere dopo i due punti (:) il solito sottoprogramma delimitato da BEGIN e END; .

Proviamo ora con un' esempio: un programma che generi un numero a caso tra 1 e 10 e lo comunichi all' operatore solo se esso è uguale a 7, 9 o 3.

```
PROGRAM numeri_casuali;
  USES crt;
  VAR a,b,c:INTEGER;
  BEGIN
  CLRSCR;
  RANDOMIZE;   (*Inizializza, ossia azzera il generatore di numeri
               casuali*)
  c:=RANDOM(10);  (*Questo  comando  permette  all'elaboratore  di
    generare un numero a caso compreso tra 0 e il numero indicato in
    parentesi. Il massimo consentito è 255*)
  CASE c OF
  7:BEGIN
  WRITELN(' Il numero è ',c);
  END;
  9:BEGIN
  WRITELN(' Il numero è ',c);
  END;
  3:BEGIN
  WRITELN(' Il numero è ',c);
  END;
  ELSE
  BEGIN
  WRITELN(' Il numero uscito è diverso da 3  7 e 9 ');
  END;
  END;
```

Anche se con questo ultimo esempio abbiamo terminato il capitolo riguardante le strutture dei dati al lettore potrebbe sembrare che quelle poche strutture analizzate debbano essere affiancate da molte altre per costruire un programma relativamente complesso, e invece secondo il teorema di Bohm-Jacopini :

"*Le tre strutture matematiche che servono per creare un qualsiasi programma sono: iterative, selettive e sequenziali*."

Le strutture sequenziali sono le più lineari, poiché il flusso di dati scorre senza ripetizioni né selezioni che abbiamo utilizzato tranquillamente nel primo programma (quello del triplo di un numero) senza saperlo.

## Capitolo 7: **Le procedure**

Spesso, soprattutto in programmi complessi e piuttosto elaborati, ci troviamo dinanzi al problema di ripetere uno stesso procedimento più volte all'interno dello stesso programma.
La soluzione a questo problema è data dal fatto che possiamo memorizzare una procedura e ripeterla a piacimento durante la stesura del programma.
Questa utilissima funzione ha la seguente sintassi:

```
PROCEDURE nome procedura;
BEGIN
blocco istruzioni
....
....
END;
```

E' importante dire che i precedenti comandi devono essere digitati tra il settore dichiarativo e il corpo del programma (per intenderci, prima del BEGIN iniziale).
Per richiamare una procedura durante il programma bisogna digitare il nome della procedura seguito da un punto e virgola (;).

Esempio: Modificare il programma precedente inserendo una procedura.

```
PROGRAM numeri_casuali;
USES crt;
VAR a,b,c:INTEGER;
PROCEDURE output; ( * Qualsiasi nome va bene * )
BEGIN
WRITELN(' Il numero estratto è ',c);
READLN;
END;
BEGIN
CLRSCR;
RANDOMIZE;
c:= RANDOM(10);
    (* Questo comando permette all'elaboratore di generare un
    numero a caso compreso tra 0 e il numero indicato in parentesi.
    Il massimo consentito è 255 *)
CASE c OF
7:BEGIN
output;
END;
9:BEGIN
output;
END;
3:BEGIN
output;
END;
ELSE
BEGIN
WRITELN(' Il numero uscito è diverso da 3  7 e 9 ' );
END;
END;
```

In questo caso una procedura è addirittura una perdita di tempo poiché le istruzioni da riscrivere sono poche ma in casi più complessi fanno risparmiare tempo e fatica.

# A P P E N D I C E  A

## GLI ERRORI DEL TP
==========

Spesso (fin troppo spesso) nella digitazione del programma è probabile che siano presenti errori: mancato punto e virgola (;) alla fine di un istruzione, valore alfanumerico dato ad una variabile numerica, ecc.
Per correggerli, il TP, al momento della compilazione, mostra in alto dei messaggi di errore che, malauguratamente per chi non lo conosce, sono in inglese!
Qui di seguito c'è una brevissima guida su come correggere gli errori più comuni:
N.B. Nel caso in cui l' errore non sia presente nella tabella, con un buon vocabolario di inglese si può tentare di capire l' origine dell' errore e quindi correggerlo.

1) **Out of memory**: il compilatore non ha abbastanza memoria per eseguire o compilare il programma. Provare a cambiare destinazione del codice eseguibile.

(Vedi Capitolo1)

2) **Unknown identifier**: identificatore sconosciuto. Non abbiamo dichiarato una variabile.

3) **Duplicate identifier:** identificatore duplicato. Due variabili identiche.Cambiarne una.

4) **Syntax error**: è sbagliata la sintassi di un istruzione.

5) **Type mismatch**: si è dichiarata una variabile diversamente da come si è considerata. Ad esempio si è caricato il valore 1 in una variabile di tipo CHAR;

6) **String constant exceeds lines**: si è dimenticato di mettere la stringa di testo fra apici.

7) **Unexpected end of file**: il numero dei BEGIN non coincide col numero degli END. Verificarli.

## A P P E N D I C E  B

Di seguito troverete dei comandi che potrebbero tornare utili in qualche programmino.

1) TEXTCOLOR(n)= Colora il testo che segue di un colore definito da n che è un numero da 0 a 15. I numeri maggiori di quindici producono un effetto lampeggiante. Ecco gli abbinamenti numeri-colori.

2) TEXTBACKGROUND(n)= Colora lo sfondo d un colore definito da n.
Sintassi simile a TEXTCOLOR.

3) ABS(x) = Calcola il valore assoluto del numero x;

4) SQRT(x) = Calcola la radice quadrata di x;

5) DELAY(n) = Crea un ciclo di attesa lungo n ( millisecondi)

## A P P E N D I C E  C

Gli operatori booleani e i segni di operazione:
==========

Come già visto in precedenza spesso in TP ci serviamo di operatori quali **AND** oppure **OR** (letteralmente 'e' ed 'o').
Essi, introdotti dal matematico inglese George Boole, servono a congiungere due proposizioni.
Prendiamo l' esempio fatto nello spiegare l'iterazione per falso: l' operatore AND infatti serve a dettare più di una condizione e farla avverare solo se entrambi sono soddisfatte.
Se inserissimo il numero 7 ad esempio, l' elaboratore ci ripeterà la "domanda", poiché il numero inserito, non soddisfa le due condizioni: " il numero deve essere minore di 30 e maggiore di 10 ".

I segni di **operazione in TP**, sono i seguenti:

' + ' = ADDIZIONE
' - ' = SOTTRAZIONE
' * ' = MOLTIPLICAZIONE
' / ' = DIVISIONE
DIV = QUOZIENTE INTERO DELLA DIVISIONE
MOD = RESTO DELLA DIVISIONE
(DIV e MOD possono essere usati soltanto se i numeri da dividere sono di tipo INTEGER )

Inoltre per indicare il rapporto fra due cifre ci serviamo dei seguenti simboli matematici:

' < ' MINORE
' > ' MAGGIORE
' = ' UGUALE
' => ' MAGGIORE O UGUALE
' <= ' MINORE O UGUALE
' <> ' DIVERSO DA ...

A P P E N D I C E D (TABELLE)

## *Variabili Numeriche*

| *Tipo Variabile* | *Intervallo* | *Spazio in memoria* |
|---|---|---|
| Shortint | -128....127 | 8 bit |
| Integer | -32768....32767 | 16 bit |
| Longint | -2147483648....2147483647 | 32 bit |
| Byte | 0....255 | 8 bit |
| Word | 0....65535 | 16 bit |
| Real | 2.9 E-39....1.7 E38 | 6 bytes |
| Single | 1.5 E-45....3.4 E38 | 4 bytes |
| Double | 5 E-324....1.7 E308 | 8 bytes |
| Extended | 3.4 E-4932....1.1 E4932 | 10 bytes |
| Comp | -9.2 E18....9.2 E18 | 8 bytes |

## *Variabili AlfaNumeriche*

| *Tipo* | *Descrizione* |
|---|---|
| Char | Da usare quando si vuole dare come valore alla variabile un solo carattere ASCII |
| String | Da usare quando si vuole dare come valore alla variabile una parola di max 255 caratteri |
| Boolean | Da usare nel caso di una condizione che può essere vera o falsa |

## *Funzioni*

Ecco una lista di alcune funzioni già implementate nel Turbo Pascal:

| | |
|---|---|
| **Abs** | Restituisce il valore assoluto dell'argomento |
| **Addr** | Restituisce l'indirizzo dell'oggetto specificato |
| **ArcTan** | Restituisce l'arcotangente dell'argomento |
| **Chr** | Restituisce il carattere del numero ordinale specificato |
| **Concat** | Concatena una sequenza di stringhe |
| **Copy** | Restituisce una sottostringa della stringa |
| **Cos** | Restituisce il coseno dell'argomento (x è un angolo in radianti) |
| **CSeg** | Restituisce il valore corrente del registro CS |
| **DSeg** | Restituisce il valore corrente del registro DS |

| | |
|---|---|
| **Eof** | Restituisce lo stato di fine file |
| **Eoln** | Restituisce lo stato di fine linea di un file di testo |
| **Exp** | Restituisce l'esponenziale dell'argomento |
| **FilePos** | Restituisce la posizione corrente nel file |
| **FileSize** | Restituisce la dimensione corrente di un file |
| **Frac** | Restituisce la parte frazionale dell'argomento |
| **Hi** | Restituisce il byte superiore dell'argomento |
| **Int** | Restituisce la parte intera dell'argomento |
| **IOResult** | Restituisce lo stato dell'ultima operazione di I/O effettuata |
| **Length** | Restituisce la lunghezza dinamica di una stringa |
| **Ln** | Restituisce il logaritmo naturale dell'argomento |
| **Lo** | Restituisce il byte inferiore dell'argomento |
| **MaxAvail** | Restituisce la dimensione del blocco contiguo più grande libero nell'heap |
| **MemAvail** | Restituisce la quantità di tutta la memoria libera nell'heap |
| **Odd** | Controlla se l'argomento è un numero dispari |
| **Ofs** | Restituisce l'offset dell'oggetto specificato |
| **Ord** | Restituisce il numero ordinale di un valore di tipo ordinale |
| **ParamCount** | il numero dei parametri passati al programma nella linea di comando |
| **ParamStr** | Restituisce uno specifico parametro della linea di comando |
| **Pi** | Restituisce il valore di Pigreco |
| **Pos** | Ricerca una sottostringa nella stringa |
| **Pred** | Restituisce il predecessore dell'argomento |
| **Ptr** | Converte il segmento base e l'offset dell'indirizzo in un valore di tipo puntatore |
| **Random** | Restituisce un numero casuale |
| **Round** | Arrotonda un valore di tipo real a uno di tipo integer |
| **SeekEof** | Restituisce lo stato di fine file di un file |
| **SeekEoln** | Restituisce lo stato di fine linea di un file |
| **Seg** | Restituisce il segmento dell'oggetto specificato |
| **Sin** | Restituisce il seno dell'argomento |
| **SizeOf** | Restituisce il numero di byte occupati dall'argomento |
| **SPtr** | Restituisce il valore corrente del registro SR |
| **Sqr** | Restituisce il quadrato dell'argomento |
| **Sqrt** | Restituisce la radice quadrata dell'argomento |
| **SSeg** | Restituisce il valore corrente del registro SS |
| **Succ** | Restituisce il successore dell'argomento |
| **Swap** | Scambia il byte superiore e inferiore dell'argomento |
| **Trunc** | Tronca un valore di tipo real a uno di tipo integer |
| **UpCase** | Converte in maiuscolo un carattere |

## *Procedure*

| | |
|---|---|
| **Append** | Apre un file esistente per l'aggiunta |
| **Assign** | Assegna il nome di un file esterno a una variabile file |
| **BlockRead** | Legge uno o più record in una variabile |
| **BlockWrlte** | Scrive uno o più record da una variabile |
| **ChDir** | Cambia la directory corrente |
| **Close** | Chiude un file aperto |
| **Dec** | Decrementa una variabile |
| **Delete** | Cancella una sottostringa da una stringa. |
| **Dispose** | Elimina una variabile dinamica |
| **Erase** | Cancella un file esterno |
| **Exit** | Esce immediatamente dal blocco corrente |
| **FillChar** | Riempie uno specificato numero (contatore) di byte contigui con il valore specificato (può essere di tipo Byte o Char) |
| **Flush** | Svuota il buffer di un file di lesto aperto in scrittura |
| **FreeMem** | Elimina una variabile dinamica di una data dimensione |
| **GetDir** | Restituisce la directory corrente del drive specificato |
| **GetMem** | Crea una variabile dinamica della dimensione specificata e inserisce l'indirizzo del blocco in una variabile puntatore |
| **Halt** | Ferma l'esecuzione del programma e ritorna al sistema operativo |
| **Inc** | Incrementa una variabile |
| **Insert** | Inserisce una sottostringa in una stringa |
| **MkDir** | Crea una sottodirectory |
| **Move** | Copia i byte dalla sorgente alla destinazione |
| **New** | Crea una nuova variabile dinamica e imposta una variabile puntatore con l'indirizzo |
| **Randomize** | Inizializza il generatore interno di numeri casuali con un valore casuale (ricavato dall'orologio di sistema) |
| **Read** | Legge un componente del file in una variabile per i file tipizzati; per i file di testo legge uno o più valori in una o più variabili |
| **Readln** | Esegue la procedura Read, poi passa alla linea seguente del file |
| **Rename** | Cambia il nome di un file esterno |
| **Reset** | Apre un file esistente |
| **Rewrite** | Crea e apre un nuovo file |
| **RmDir** | Elimina una sottodirectory vuota |
| **RunError** | Ferma l'esecuzione del programma |
| **ScrollTo** | Scorre la finestra dello schermo per mostrare una locazione |
| **Seek** | Sposta la posizione corrente di un file in quella specificata |
| **SetTextBuf** | Assegna un buffer di I/O a un file di testo |

| | |
|---|---|
| **Str** | Converte un valore numerico in una stringa |
| **Truncate** | Tronca il file alla posizione corrente |
| **Val** | Converte un valore stringa nella sua rappresentazione numerica |
| **Write** | Scrive una variabile in un componente del file per i file tipizzati; per i file di testo scrive uno o più valori al file |
| **Writeln** | Esegue la procedura Write, poi inserisce il segnalatore di fine della linea nel file |

# CODICE ASCII

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | 32 | | 64 | @ | 96 | ` | 128 | Ç | 160 | á | 192 | └ | 224 | Ó |
| 1 | ☺ | 33 | ! | 65 | A | 97 | a | 129 | ü | 161 | í | 193 | ┴ | 225 | ß |
| 2 | ☻ | 34 | " | 66 | B | 98 | b | 130 | é | 162 | ó | 194 | ┬ | 226 | Ô |
| 3 | ♥ | 35 | # | 67 | C | 99 | c | 131 | â | 163 | ú | 195 | ├ | 227 | Ò |
| 4 | ♦ | 36 | $ | 68 | D | 100 | d | 132 | ä | 164 | ñ | 196 | ─ | 228 | õ |
| 5 | ♣ | 37 | % | 69 | E | 101 | e | 133 | à | 165 | Ñ | 197 | ┼ | 229 | Õ |
| 6 | ♠ | 38 | & | 70 | F | 102 | f | 134 | å | 166 | ª | 198 | ã | 230 | µ |
| 7 | • | 39 | ' | 71 | G | 103 | g | 135 | ç | 167 | º | 199 | Ã | 231 | þ |
| 8 | ◘ | 40 | ( | 72 | H | 104 | h | 136 | ê | 168 | ¿ | 200 | ╚ | 232 | Þ |
| 9 | ○ | 41 | ) | 73 | I | 105 | i | 137 | ë | 169 | ® | 201 | ╔ | 233 | Ú |
| 10 | ◙ | 42 | * | 74 | J | 106 | j | 138 | è | 170 | ¬ | 202 | ╩ | 234 | Û |
| 11 | ♂ | 43 | + | 75 | K | 107 | k | 139 | ï | 171 | ½ | 203 | ╦ | 235 | Ù |
| 12 | ♀ | 44 | , | 76 | L | 108 | l | 140 | î | 172 | ¼ | 204 | ╠ | 236 | ý |
| 13 | ♪ | 45 | - | 77 | M | 109 | m | 141 | ì | 173 | ¡ | 205 | ═ | 237 | Ý |
| 14 | ♫ | 46 | . | 78 | N | 110 | n | 142 | Ä | 174 | « | 206 | ╬ | 238 | ¯ |
| 15 | ☼ | 47 | / | 79 | O | 111 | o | 143 | Å | 175 | » | 207 | ¤ | 239 | ´ |
| 16 | ► | 48 | 0 | 80 | P | 112 | p | 144 | É | 176 | ░ | 208 | ð | 240 | - |
| 17 | ◄ | 49 | 1 | 81 | Q | 113 | q | 145 | æ | 177 | ▒ | 209 | Ð | 241 | ± |
| 18 | ↕ | 50 | 2 | 82 | R | 114 | r | 146 | Æ | 178 | ▓ | 210 | Ê | 242 | ‗ |
| 19 | ‼ | 51 | 3 | 83 | S | 115 | s | 147 | ô | 179 | │ | 211 | Ë | 243 | ¾ |
| 20 | ¶ | 52 | 4 | 84 | T | 116 | t | 148 | ö | 180 | ┤ | 212 | È | 244 | ¶ |
| 21 | § | 53 | 5 | 85 | U | 117 | u | 149 | ò | 181 | Á | 213 | ı | 245 | § |
| 22 | ▬ | 54 | 6 | 86 | V | 118 | v | 150 | û | 182 | Â | 214 | Í | 246 | ÷ |
| 23 | ↨ | 55 | 7 | 87 | W | 119 | w | 151 | ù | 183 | À | 215 | Î | 247 | ¸ |
| 24 | ↑ | 56 | 8 | 88 | X | 120 | x | 152 | ÿ | 184 | © | 216 | Ï | 248 | ° |
| 25 | ↓ | 57 | 9 | 89 | Y | 121 | y | 153 | Ö | 185 | ╣ | 217 | ┘ | 249 | ¨ |
| 26 | → | 58 | : | 90 | Z | 122 | z | 154 | Ü | 186 | ║ | 218 | ┌ | 250 | · |
| 27 | ← | 59 | ; | 91 | [ | 123 | { | 155 | ø | 187 | ╗ | 219 | █ | 251 | ¹ |
| 28 | ∟ | 60 | < | 92 | \ | 124 | \| | 156 | £ | 188 | ╝ | 220 | ▄ | 252 | ³ |
| 29 | ↔ | 61 | = | 93 | ] | 125 | } | 157 | Ø | 189 | ¢ | 221 | ¦ | 253 | ² |
| 30 | ▲ | 62 | > | 94 | ^ | 126 | ~ | 158 | × | 190 | ¥ | 222 | Ì | 254 | ■ |
| 31 | ▼ | 63 | ? | 95 | _ | 127 | ⌂ | 159 | ƒ | 191 | ┐ | 223 | ▀ | 255 | |

www.oppes.it