9 - LE ALTRE STRUTTURE ITERATIVE: REPEAT … UNTIL … E WHILE … DO …

Una FOR ... DO ...
comanda la ripetizione di un'istruzione (o di un blocco di istruzioni)
PER UN NUMERO PREFISSATO DI VOLTE.
Pertanto

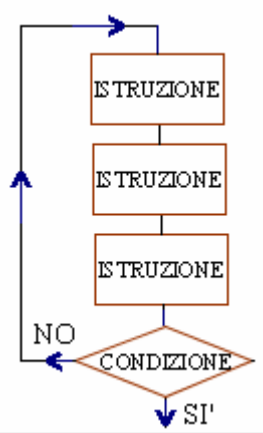
la FOR … DO …
è utilizzabile solo quando è noto fin dall'inizio
***quante volte* l'istruzione (o il blocco) andrà ripetuto.**

In caso contrario,
si utilizza la
REPEAT … UNTIL … (RIPETI … FINCHE')

oppure la
WHILE … DO … (FINTANTOCHE' … ESEGUI)

Esempio:

```
program somma_di_un_numero_non_prefissato_di_addendi;
var somma, a: longint;
begin
        writeln ('Introduci i numeri da addizionare');
        writeln ('Quando la sequenza sarà finita digita il numero 0');
        somma:=0;
        repeat
                readln(a);
                somma:=somma+a;
        until a=0;
        writeln ('Somma = ', somma);
        readln;
end.
```



La struttura REPEAT ... UNTIL ...
ordina alla macchina di:

1. **ESEGUIRE il blocco di istruzioni che sta fra la parola *REPEAT* e la parola *UNTIL*** (NOTA)**;**
2. **CONTROLLARE se è verificata o no LA CONDIZIONE che sta dopo la parola *UNTIL*;**
   - ♪ **SE *tale condizione* NON E' VERIFICATA, RIPETERE *il ciclo* (= *ritornare al punto* 1),**
   - ♫ **SE *la condizione* E' VERIFICATA, USCIRE *dal ciclo.***

Osserviamo che:

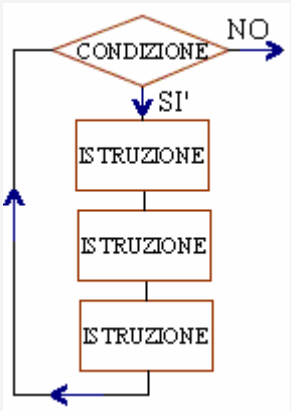
a) **il blocco di istruzioni viene certamente eseguito almeno una volta**;
b) il **controllo** se la condizione sia verificata o meno avviene **alla fine** del ciclo;
c) l' **uscita** dal ciclo si ha **quando la condizione E' verificata**

NOTA: *il blocco si può eventualmente ridurre ad una sola istruzione*

Poiché con una struttura REPEAT ... UNTIL ... il ciclo viene sempre eseguito almeno una volta, tale struttura NON può essere utilizzata in quei casi in cui è necessario fare in modo che, se una data variabile assume determinati valori, il ciclo ... non venga mai eseguito!
In tali casi, si utilizza la WHILE ... DO ... (FINTANTOCHE' … ESEGUI).

Esempio:

```
program indovina;
var animale: string [20];
begin
        writeln ('Caro amico che stai davanti a me,');
        writeln ('io, il computer, sto pensando ad un animale.');
        writeln ('Prova ad indovinare di quale animale si tratta ... ');
        writeln ('(scrivi tutto minuscolo e senza articolo)');
        readln (animale);
        while animale <> 'orso' do
                                begin
                                    writeln ('Sbagliato. Ritenta!');
                                    readln (animale);
                                end;
        writeln ('OK, indovinato!');
        readln;
end.
```



La struttura WHILE ... DO ... ordina alla macchina di:

1. **CONTROLLARE** se la **CONDIZIONE** che sta dopo la parola *WHILE* è verificata oppure no;
2.
   - ♪ **SE la *condizione* E' VERIFICATA,**
     ESEGUIRE *il blocco di istruzioni compreso fra le parole BEGIN e END* (NOTA)**,**
     POI, RIPETERE IL CICLO *ritornando al punto 1;*
   - ♫ **SE *la condizione* NON E' VERIFICATA, USCIRE *dal ciclo***

Osserviamo che:

a) **il blocco di istruzioni può anche non essere eseguito neppure una volta**;
b) il **controllo** se la condizione sia verificata o meno avviene **all'inizio**;
c) l' **uscita** dal ciclo si ha **quando la condizione NON è verificata**

NOTA: *il blocco si può eventualmente ridurre ad una sola istruzione; in tal caso sono inutili i BEGIN, END*

Spesso accade che una stessa iterazione possa essere realizzata, indifferentemente, sia con una repeat ... until ... che con una while ... do ...

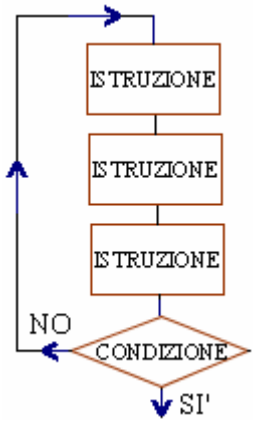
Esempio: *L'utente sceglie un numero intero n compreso fra 1 e 100.*
La macchina "estrae a sorte", ripetutamente, un intero x (1≤ x ≤ 100);
si contano i "tentativi" che farà la macchina prima di estrarre proprio n.

```
program computerindovinatucolrepeat;
var n, tent, x: integer;
begin
        write ('n= ');  readln (n);
        tent:=0;
        randomize;
        repeat
           x:=random(100)+1;  writeln (x);
           tent:=tent+1;
        until x=n;
        write ('numero tentativi = ', tent);
        readln;
end.
```

```
program computerindovinatucolwhile;
var n, tent, x: integer;
begin
        write ('n= ');  readln (n);
        randomize;
        x:=random(100)+1;  writeln (x);
        tent:=1;
        while x<>n do
        begin
             x:=random(100)+1;  writeln (x);
             tent:=tent+1;
        end;
        write ('numero tentativi = ', tent);
        readln;
end.
```

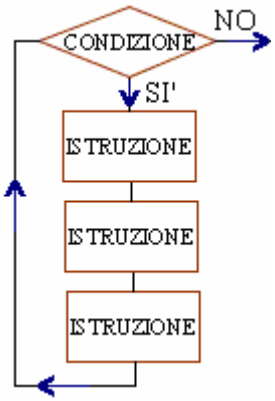
SCHEMA RIASSUNTIVO

REPEAT
istruzione 1;
istruzione 2;
istruzione 3;
…
UNTIL condizione



a) il blocco delle istruzioni viene certamente eseguito almeno una volta
b) controllo condizione **alla fine**
c) uscita dal ciclo **quando la condizione E' verificata**

WHILE condizione DO
BEGIN
istruzione 1;
istruzione 2;
istruzione 3;
…
END;



a) il blocco delle istruzioni potrebbe anche non essere eseguito nemmeno una volta
b) controllo condizione **all'inizio**
c) uscita dal ciclo **quando la condizione NON E' verificata**

Ecco, infine, qui di seguito un esempio in cui uno stesso problema di programmazione (calcolo della somma dei quadrati dei primi n interi positivi, con n letto da tastiera) può essere risolto, indifferentemente, con TUTTE E TRE le strutture iterative studiate.

```
program alfa;  var n, i, sommaquadrati: longint;
begin
      write ('n= '); readln(n); sommaquadrati:=0;
      for i:=1 to n do
                        sommaquadrati:=sommaquadrati+i*i;
      write (sommaquadrati);    readln;
end.
```

```
program beta;  var n, i, sommaquadrati: longint;
begin
      write ('n= '); readln (n); sommaquadrati:=0;
      i:=1;
      repeat
          sommaquadrati:=sommaquadrati+i*i;
          i:=i+1;
      until i>n;
      write (sommaquadrati);    readln;
end.
```

```
program gamma;  var n, i, sommaquadrati: longint;
begin
      write ('n= '); readln (n); sommaquadrati:=0;
      i:=1;
      while i<=n do
                begin
                     sommaquadrati:=sommaquadrati+i*i;
                     i:=i+1;
                end;
      write (sommaquadrati);    readln;
end.
```

In questo spazio, per esercizio, puoi fare la "TRACCIA" del programma, ossia scrivere come muta il valore delle variabili, man mano che le istruzioni vengono eseguite

n
i
sommaquadrati

n
i
sommaquadrati

n
i
sommaquadrati

LE “SCATOLETTE DI MEMORIA” E LA “TRACCIA”

Una “**variabile**”, in un linguaggio di programmazione, è un *nome* il cui ruolo è di **identificare una locazione di memoria nella quale viene immagazzinato un valore** (numerico, o anche non numerico) **che può cambiare man mano che le diverse istruzioni del programma vengono eseguite.**

Una variabile si può pensare dunque come una **“scatoletta” di memoria**,
il cui contenuto può, nel corso dell’esecuzione del programma, mutare.

Seguire l’evoluzione delle varie “scatolette di memoria” durante l’esecuzione
(= fare la cosiddetta “TRACCIA” del programma)
è essenziale per comprendere cosa il programma fa veramente, e quindi anche per capire se davvero realizza l’obiettivo che ci eravamo prefissi scrivendo la sequenza delle istruzioni: insomma **per riconoscere se, scrivendolo, abbiamo fatto degli errori “di sostanza”.**

AD ESEMPIO, facciamo la “traccia”, per un dato input, del precedente “program alfa”:

```
program alfa;  var n, i, sommaquadrati: longint;
begin
      write (’n= ’); readln (n);
      sommaquadrati:=0;
      FOR i:=1 to n DO sommaquadrati:=sommaquadrati+i*i;
      write (sommaquadrati);    readln;
end.
```

MONITOR $n = 4$

RAM

n	4			
i		1	~~1~~ 2	~~1~~ ~~2~~ 3	~~1~~ ~~2~~ ~~3~~ 4
sommaquadrati	0	~~0~~ 1	~~0~~ ~~1~~ 5	~~0~~ ~~1~~ ~~5~~ 14	~~0~~ ~~1~~ ~~5~~ ~~14~~ 30

MONITOR
$n = 4$
30

ALTRO ESEMPIO di “traccia” di un programma:

programma per il calcolo del **Massimo Comun Divisore con l’Algoritmo di Euclide**.
(puoi andare a pag. 26 per una spiegazione dettagliata delle basi teoriche del procedimento).

Siano a, b i due interi di cui vogliamo determinare il M.C.D.
Calcoliamo il resto r della divisione intera $a : b$,
e a questo punto sostituiamo la coppia (a, b) con la coppia (b, r).
Iteriamo (=ripetiamo) il procedimento per la nuova coppia;
prima o poi, si arriverà al punto in cui il secondo elemento della coppia si annullerà.
Bene, il valore che avrà in quel momento il primo elemento della coppia sarà il M.C.D. cercato.

```
program euclide;  uses crt;
var a, b, r: longint;
BEGIN
  clrscr;
  write (’a = ’); readln(a);
  write (’b = ’); readln(b);
  REPEAT
        r:=a mod b;
        a:=b;
        b:=r;
        writeln (a, ’ ’, b);
  UNTIL b=0;
  writeln (’Il MCD è ’, a);
  readln;
END.
```

Lascio a te il compito
di vedere come varia,
istruzione dopo istruzione,
il contenuto delle tre scatolette

a	
b	
r	

per un dato input scelto dall’utente.

DIAGRAMMA DI FLUSSO → dell’algoritmo;
ricorda che i blocchi devono aver forma di

OVALE	**INIZIO, FINE**
PARALLELOGR.	**INPUT, OUTPUT**
RETTANGOLO	**ELABORAZIONE**
ROMBO	**SELEZIONE**