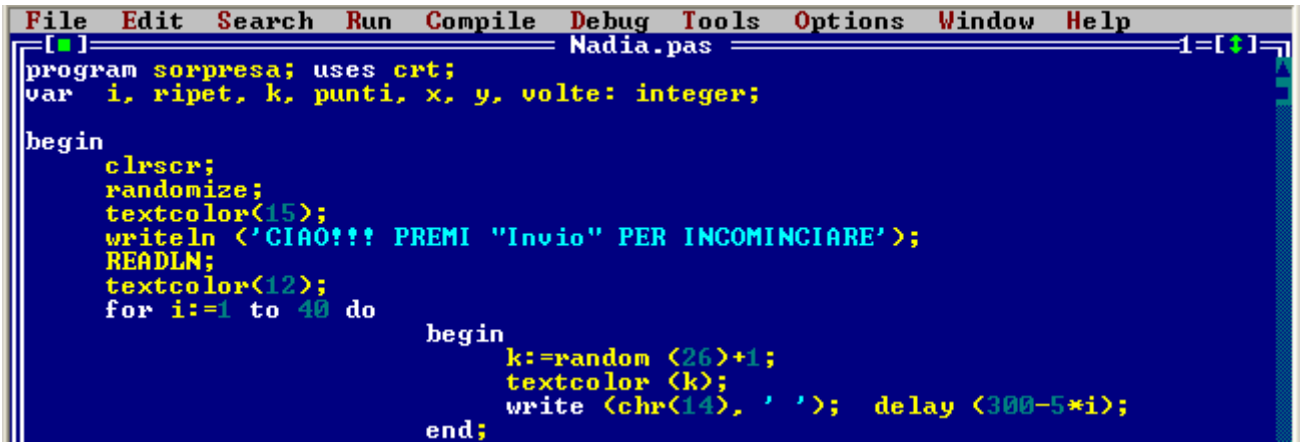
PROGRAMMAZIONE IN LINGUAGGIO PASCAL - INDICE



PROGRAMMAZIONE IN LINGUAGGIO PASCAL

♫ INTRODUZIONE ALLA PROGRAMMAZIONE IN LINGUAGGIO PASCAL

- Le pagine che seguono insegnano a scrivere un programma in linguaggio *Pascal*.
  In realtà, il Pascal (creato nel 1970) NON è, fra i linguaggi, proprio l'ultima novità.
  Tuttavia, te lo propongo caldamente come
  linguaggio davvero ideale per iniziare a familiarizzare con la programmazione.
  E perché mai?

  Perché **il Pascal fu creato apposta per facilitare il compito di chi voglia imparare a programmare**: esso infatti **concilia l'essenzialità con l'eleganza e la completezza.**

  Dopo aver appreso – divertendoti – gli elementi fondamentali del Pascal,
  potrai facilmente, se i tuoi interessi o i tuoi studi o il tuo lavoro te lo suggeriranno, passare
  a occuparti di *qualsiasi altro linguaggio*. Ma nel frattempo avrai piacevolmente assimilato
  un *metodo* e una *mentalità* che ti renderanno *molto più agevole* affrontare cose nuove.

  Inoltre **realizzare programmi con Pascal è completamente gratuito nella legalità**, come vedremo.

- Ti è già noto che una volta scritto un programma in un dato linguaggio,
  per comunicarlo al computer occorre *un altro* programma,
  che sia capace di prendere il programma da noi scritto e tradurlo in "*linguaggio macchina*",
  quello a base di 0 e di 1, l'unico che il computer possa effettivamente "comprendere".

  Nel caso del *Pascal* tale programma si chiama **compilatore Pascal**
  ("compilatore" perché, a differenza di un "interprete", non traduce le istruzioni una per una,
  ma traduce *tutto il complesso* delle istruzioni, generando un codice binario che viene detto
  "il programma oggetto", può essere salvato in un file, e può essere eseguito da *qualsiasi* computer).

- Bene! **Un compilatore Pascal, chiamato Free Pascal, è scaricabile comodamente, gratuitamente e legalmente** (è *free software*!) **dal sito**
  www.freepascal.org/

  Collegati, scegli l'opzione Download,
  poi, dopo la parola "Binari", clicca sulla scritta blu
  corrispondente al microprocessore e al sistema operativo del TUO computer.

  Nella nuova finestra che si apre
  clicca sul sito dal quale vuoi scaricare, ad es.
  ftp.freepascal.org;

  ora clicca sulla scritta blu
  che segue le parole "Scarica come installer".

  Uscirà una finestrella grigia;
  in essa, clicca su "Salva"
  e scegli poi il desktop come destinazione per il file.

  Dovrai attendere qualche minuto per il trasferimento del file
  dal server remoto al tuo computer.

  Terminato il download,
  sul tuo desktop comparirà la seguente icona:

  Fai doppio clic su di essa e, nella successiva finestrella grigia, clicca con fiducia su "**Esegui**".

  Segui le istruzioni,
  che porteranno alla vera e propria installazione di Free Pascal sul computer.
  Alla fine della procedura guidata, ti ritroverai sul desktop un'altra icona, la seguente:

  Se ci fai *doppio clic* sopra, ecco che sei pronto per lavorare!!!

… L'hai fatto, vero, il doppio clic?
Bene: sul tuo monitor è ora comparsa una finestra con questo bordo superiore:

Clicca su **File**, poi su **New**.

Ecco che ti compare uno spazio di color blu: **scrivi il tuo programma**
(per rompere il ghiaccio, potresti copiarne uno dalle pagine successive)
poi, quando avrai finito, clicca sul menu **Compile**.

Fra le opzioni della finestrella che si apre, troverai ancora **Compile**: clicca su questa opzione.
Apparirà una finestra nella quale sarà richiesto di specificare il nome che intendi assegnare al file
(il nome del file non dovrà obbligatoriamente coincidere con quello del programma).

Scegli il nome, poi clicca su OK per confermarlo, ma …
... tieni presente che il compilatore,
prima di procedere alla traduzione del tuo programma in linguaggio macchina,
andrà a valutare la **correttezza sintattica** del programma stesso,
e, nel caso rilevi un errore formale, farà comparire sul monitor una finestrella con la scritta:
Compile Failed (Compilazione fallita).

Starà allora a te andare alla caccia degli errori:
occhio, questo sarà forse il compito più noioso,
perché anche una semplice virgola, o punto e virgola, o apice, dimenticato può essere un errore.
Comunque, se hai sbagliato e quindi è spuntata la **Compile Failed**,
nella **parte bassa della finestra** ti saranno comparse
anche delle **indicazioni**, che certamente ti aiuteranno, **sulla natura e la posizione dell'errore**.

Ad esempio,
";" expected but "WRITE" found
è il messaggio che ti dice: hai omesso un "punto e virgola" prima di un "WRITE".
E accanto **troverai scritta** anche **LA RIGA E LA COLONNA** in cui l'errore è stato riscontrato!!!

Premi "Invio", mettiti con pazienza a correggere l'errore, dai di nuovo il comando Compile;
se sono rimasti ancora degli errori, rivedrai il **Compile Failed** e le indicazioni a fondo pagina,
altrimenti leggerai la scritta:
Compile successful (Compilazione riuscita), **Press any key** (Premi un tasto qualsiasi).
Tu premi dunque un tasto, ad esempio "Invio".

A questo punto nella "directory" (=cartella) corrente,
che sarà poi una delle sottocartelle della cartella principale (su hard disk)
nella quale è stato collocato Free Pascal dalla procedura di installazione,
saranno stati creati DUE file, entrambi col nome da te scelto:

- ♪ uno, di estensione **.pas** (ma comunque apribile con un elaboratore di testi, ad esempio con Word)
  conterrà il **testo del programma in linguaggio**
  (proprio la sequenza di istruzioni da te scritta, il cosiddetto "**listato**" del programma);
- ♫ l'altro conterrà invece il **programma "compilato", scritto in linguaggio macchina**,
  e sarà dunque un file **"eseguibile"**, di estensione **.exe**.

Ma nel momento presente tali file non ti interessano molto:
infatti una copia del programma oggetto risiede a questo punto anche nella RAM,
e quando vuoi che il computer lo esegua,
clicca sul menu RUN e in questo scegli l'opzione "Run".
Il programma va in esecuzione.

Ricorda che, se il programma ti richiede di inserire un dato, ad esempio un numero,
tu dopo averlo digitato sulla tastiera dovrai sempre premere "Invio" per confermare.
Al termine dell'esecuzione dovrai ulteriormente premere "Invio" per tornare al listato delle istruzioni.

Il file **.exe** creato dal compilatore, invece,
potrebbe essere anche ricopiato su qualsiasi altro computer:
doppiocliccandoci sopra, ecco che il programma si metterebbe a funzionare.
Quindi se hai realizzato un programma interessante, lo puoi anche regalare a qualche tua amica/amico.

Buon divertimento con la programmazione!!!

1 - ESEMPIO INTRODUTTIVO

ESEMPIO

```
program pitagora;
uses crt;
var a, b, c: longint;
begin
  clrscr;
  writeln (’Inserisci le misure dei lati del triangolo’);
  writeln (’Devono essere numeri interi in ordine crescente’);
  readln (a); readln (b); readln (c);
  if a*a+b*b=c*c then write (’Il triangolo è rettangolo’)
                 else write (’Il triangolo non è rettangolo’);
  readln;
end.
```

Vuoi vedere il programma in esecuzione?

(*dopo aver inserito ciascun numero, dovrai premere “Invio” per confermare*)

⇨

OSSERVAZIONI

program = parola chiave con cui deve iniziare ogni programma (1)

uses crt = è la chiamata di un programma “di servizio” (2)

var = precede l’**elenco delle variabili** col rispettivo *tipo* (*longint* = lungo intero)

begin = parola chiave che deve precedere il “corpo” del programma

clrscr = istruzione di “**pulizia dello schermo**”: **CL**ea**R SCR**een

write = scrivi, **writeln** = scrivi-poi-vai-a-capo (si pronuncia di solito *wraitlain*)

read = leggi, **readln** = leggi-poi-dopo-aver-letto-vai-a-capo (pron. *ridlain*)

if … then … else … = se … allora … altrimenti …

La moltiplicazione, in Pascal, si indica con un asterisco che non si può sottintendere: *

“readln” prima dell’ “end” finale = inserisce una pausa (3)

end = parola chiave che deve chiudere ogni programma

(1) **Per il nome di un programma PASCAL si possono utilizzare caratteri alfanumerici** (=alfabetici/numerici), **ma sono vietati caratteri di altro tipo, eccetto quello “di sottolineatura”**, che può essere utile quando si vogliono usare più parole, dato che il nome di un programma non può contenere spazi vuoti:
es. program mese1_e_mese2.
Le stesse regole valgono anche per i nomi di variabili.
Al posto di dire “nome” (di un programma, di una variabile) si preferisce dire “identificatore”.

(2) **“uses crt” è la chiamata di un programma “di servizio” senza il quale il nostro programma non sarebbe in grado di eseguire la successiva istruzione “clrscr” per la pulizia dello schermo**.

(3) **“readln” prima dell’ “end” finale inserisce una pausa senza la quale il computer eseguirebbe immediatamente l’ “end” finale** e ritornerebbe quindi subito a mostrare sullo schermo la videata delle istruzioni del programma, **senza che l’utente possa avere il tempo di osservare l’output.**
In fase di esecuzione, quando si vorrà che la pausa termini, si premerà “Invio”.

(4) **OGNI ISTRUZIONE DEVE SEMPRE TERMINARE COL “PUNTO E VIRGOLA”, con l’eccezione del “begin” iniziale (per cui il “punto e virgola” è facoltativo) e dell’ “end” finale, che deve invece essere seguito da un “punto”**.

(5) **La rientranza verso destra, o “INDENTAZIONE”, di alcune righe del programma rispetto ad altre, non è obbligatoria** ma è *utilissima* per migliorare la “leggibilità” del programma.

(6) **Le istruzioni, le dichiarazioni, gli identificatori di un programma PASCAL possono essere scritti indifferentemente in minuscolo o usando le maiuscole a piacere**: PASCAL interpreta, ad esempio, “prodotto”, “ProDOTto” e “PRODOTTO” considerandoli come *uno stesso* identificatore.
Invece quando utilizziamo un’istruzione di scrittura “write” o “writeln”, e inseriamo una “stringa” (=sequenza di caratteri) entro la coppia di apici, questa scritta (che potrà contenere caratteri di tipo qualsiasi) verrà mandata in output esattamente come si presenta, eventuali maiuscole comprese.

2 - ISTRUZIONI DI INPUT-OUTPUT E DI ASSEGNAZIONE

- **INPUT-OUTPUT**

write (’Carissima ti bacio ’, x, ’ volte’)

a) Scrivi la **stringa** “Carissima ti bacio ”
b) poi il **valore della variabile** x (NOTA)
c) infine la **stringa** “ volte”

NOTA: ossia, il contenuto che ha in quel momento la locazione di memoria (“scatoletta”) x

Ogni coppia **’ ’** di **APICI** serve a indicare che il computer dovrà scrivere sul monitor **ESATTAMENTE quella SEQUENZA DI CARATTERI (=STRINGA) che è contenuta entro gli apici stessi.**

In **ASSENZA DI APICI,** va invece scritto il **VALORE di una variabile (o espressione)**

writeln (’a+b = ’, a+b)

a) Scrivi la stringa “a+b = ”
b) poi il valore dell’espressione a+b (il valore della somma dei contenuti delle “scatolette” a, b)
c) infine, **vai a capo** col cursore (writeLN e non write)

read (num) **Leggi** ciò che l’utente digita sulla tastiera e **mettilo nella “scatoletta”** num
NOTA: **l’utente, dopo aver digitato, dovrà premere il tasto “Invio” sulla tastiera**

readln (num) **Come prima**; **alla fine**, però, **vai a capo** col cursore

NOTA: sovente ci permetteremo di scrivere, alla buona, “scatoletta” anziché “locazione di memoria”

- **ASSEGNAZIONE**

Il simbolo dell' "assegnazione" in PASCAL *NON* è =, bensì è **:=**

num:=5 **Assegna alla variabile num (cioè: metti nella scatoletta num) il valore 5**

y:=a+b **Assegna alla variabile y (cioè: metti nella scatoletta y) il valore dato dalla somma dei valori che in quel momento hanno le variabili a, b**
(cioè: che in quel momento stanno nelle scatolette a, b)

c:=c+1

Assegna alla variabile c (ossia: metti nella scatoletta c) il valore che c aveva PRECEDENTEMENTE, aumentato di 1
(quindi, in pratica: **incrementa di una unità il valore della variabile c**)
:= non indica uguaglianza, indica assegnazione!

ESEMPI

```
program moltiplicazioni_1;
uses crt;
var a, b: longint;
begin
        clrscr;
        readln (a);
        readln (b);
        writeln (a*b);
        readln;
end.
```

OSSERVAZIONI

a*b

Il simbolo di moltiplicazione non si può sottintendere all’interno di una espressione matematica in Pascal, e si realizza con un asterisco

readln

Istruzione indispensabile se si vuole avere tempo di osservare l’output: prova a toglierla, fai eseguire (=dai il “Run”) e vedrai!

clrscr

Prova a toglierlo, dai il Run per 2 esecuzioni consecutive e vedi che succede …

```
program moltiplicazioni_2;
uses crt;
(*questo programma è simile
  al precedente, è solo più ricco*)
var a, b, prod: longint;
begin
        clrscr;
        write (’a = ’);
        readln (a);
        write (’b = ’);
        readln (b);
        prod:=a*b;
        write (’a*b = ’);
        writeln (prod);
readln;
end.
```

OSSERVAZIONI

(* *)

Le “parentesi asteriscate” consentono di inserire un commento in un punto qualsiasi del programma

write (’a = ’)

ha qui la funzione di mandare in output una scritta che aiuti l’utente a capire cosa deve fare.

```
program moltiplicazioni_3;
uses crt;
var a, b: longint;


begin
        clrscr;
        write (’a = ’);
        readln (a);
        write (’b = ’);
        readln (b);

        write (’a*b = ’, a*b);

readln;
end.
```

OSSERVAZIONI

Organizzando il nostro programma in questo modo, possiamo evitare di introdurre la variabile “prod”

Domanda: se anziché scrivere

write (’a*b = ’, a*b)

scrivessimo

write (a, ’*’, b, ’=’, a*b)

come cambierebbe l’output?

3 - a) I PRINCIPALI TIPI DI VARIABILI NUMERICHE b) LE VARIABILI "STRINGA"

a) Alcuni fra i principali tipi di variabili numeriche

FREE PASCAL mette a disposizione parecchi tipi di variabili *intere* e *non intere*.
Ma lasciando i tanti possibili approfondimenti all'eventuale iniziativa del lettore, che potrà trovarli ad es. sul sito www.freepascal.org, diciamo che, avendo necessità di utilizzare una variabile di tipo intero, la si dichiarerà in genere come **VAR integer** oppure come **VAR longint**, tenendo presente che

- ♪ una variabile di tipo **integer** può assumere i suoi valori nell'intervallo da **–32768** a **+32767**
- ♫ e una variabile di tipo **longint** da **–2147483648** a **+2147483647** .

♥ **E' di estrema importanza tener conto dell'intervallo di variabilità (range).**
Se ad esempio una variabile n in un dato programma PASCAL è stata dichiarata di tipo integer, e in quel programma compare l'istruzione n:=40000,
oppure la coppia di istruzioni successive n:=32767; n:=n+1, allora sarà un bel guaio!!!
In fase di compilazione o di esecuzione verrà segnalato un errore di "traboccamento" (*overflow*).

Una variabile numerica a valori **non necessariamente interi** in generale è dichiarata come **VAR real.** Le variabili di tipo *real* hanno un *range* che copre, perlomeno, l'intervallo da $1.5 \cdot 10^{-45}$ a $3.4 \cdot 10^{38}$ (nel caso del sottotipo *single*); gli altri sottotipi *double* e *extended* hanno un *range* molto più ampio. Il numero di byte occupati in memoria può essere di 4, di 8, o di 10. Il programmatore potrà specificare il sottotipo con una dichiarazione come *VAR x: double*, oppure non specificarlo scrivendo *VAR x: real*; in tal caso, la variabile in gioco è destinata a diventare una *single* o una *double* a seconda del processore. Consulta www.freepascal.org per informazioni più dettagliate; vedi anche il paragrafo 7b a pag. 11.

♥ OCCHIO! - In PASCAL il **separatore** della parte intera dalla decimale è il **PUNTO** e non la virgola.

- Si ha la possibilità, quando una *write* o una *writeln* è riferita ad un numero di tipo intero, di **far sì che il numero occupi, sul monitor, tanti spazi quanti esattamente vogliamo noi**.
  - Ad esempio, se *prodotto* è una variabile *integer* o *longint*, l'istruzione **write (prodotto:8)** fa sì che sul monitor compaia il valore che in quel momento la variabile *prodotto* possiede, scritto in modo da occupare **esattamente 8 posizioni di carattere, e allineato a destra** in tale spazio. In questo contesto, il numero 8 viene detto "**ampiezza**".

- **Se noi mandiamo in output, tramite una write o una writeln, un numero di tipo *real*, esso ci apparirà sul monitor in "notazione esponenziale".** Vale a dire, vedremo il numero scritto come **prodotto di un fattore compreso fra 1** (incluso) **e 10** (escluso)**, per una opportuna potenza di 10**.
  - Esempio. Supponiamo che in un dato istante la variabile x, di tipo *real*, abbia il valore **123.45**. Allora l'istruzione **write (x)** farebbe comparire sul monitor **1.23450000000000E+002** (che significa 1.2345 moltiplicato per 10^2 ; "**E" sta per "esponente di 10"**).

Se vogliamo invece **che l'output appaia scritto in notazione *non* esponenziale**, dovremo integrare l'istruzione *write* come nell'esempio che segue: **write (x:20:12)**. Tale istruzione avrebbe l'effetto di far scrivere sul monitor il valore della variabile *real x*, scritto in notazione NON esponenziale, in modo da occupare un campo di esattamente **20 caratteri, di cui 12 riservati alle cifre dopo il punto decimale**.

- Ad esempio, nel caso **x =123.45**, si avrebbe
  123.450000000000 (4 spazi vuoti all'inizio: numero *allineato a destra* in un campo di 20 caratteri)

NOTA 1 - Lo ribadiamo: i numeri non interi hanno sempre, in PASCAL, **il PUNTO e non la virgola come separatore per le cifre decimali.** Anche quando un non-intero viene inserito in *input*, in fase di esecuzione di un programma scritto in PASCAL, occorre regolarsi così.

NOTA 2 - L'aggettivo *reale* è un po' "sprecato" in questo contesto. Sì, è vero, si tratta di numeri reali, però, per il fatto di non poter avere infinite cifre decimali, sono senz'altro addirittura *razionali*.

b) Cenni alle variabili "stringa"

STRINGA = SEQUENZA DI CARATTERI

```
program esempiosullevariabilistringa; uses crt;
var nome: string [20];
begin
   clrscr;
   writeln ('Come ti chiami?');
   readln (nome);
   if nome='Mario' then writeln ('Ti chiami come me!')
      else writeln ('Buona giornata ', nome);
   readln;
end.
```

var nome: string [20]

"**nome**" è una **variabile stringa**.
La "scatoletta" chiamata "nome" non è destinata a contenere un numero, bensì una "**stringa**", ossia una *sequenza di caratteri*.
Il numero [20] *entro parentesi quadre* indica che la stringa potrà avere una *lunghezza massima* di 20 caratteri.

Il caso particolare string [1] può essere rimpiazzato dal tipo "**carattere**" (*char*).
Esempio: **var consonante: char**

4 - IF … THEN … ELSE … (= LA STRUTTURA DI SELEZIONE)

- Esempio 1: **if a*a+b*b=c*c then write (’rettangolo’) else write (’non rettangolo’)**

In generale, la struttura (*struttura* = istruzione che governa altre istruzioni)
IF *condizione* THEN *istruzione 1* (ELSE *istruzione 2*)
ordina al computer di:

i) controllare se è verificata la *condizione*;

ii) in caso affermativo, eseguire l’*istruzione 1*,
in caso negativo eseguire l’*istruzione 2*
(questa parte con l’ELSE può anche mancare)

- Esempio 2:

```
if a*a+b*b=c*c then
                    begin
                       writeln (’Il triangolo è rettangolo, perché la somma’);
                       writeln (’dei quadrati di due lati uguaglia il quadrato del terzo’);
                    end
               else write (’Il triangolo non è rettangolo’)
```

Notare in questo es. l’uso degli **indicatori di INIZIO BLOCCO (begin) e FINE BLOCCO (end)**.

- Esempio 3:

```
if media<4 then media:=4
```

Questa istruzione potrebbe far parte di un programma nel quale un insegnante indulgente decida di "alzare" al 4, per non infierire, la media di un suo alunno nel caso questa risulti < 4; in caso contrario, cioè se la media è maggiore o uguale a 4, essa non verrà modificata).

L’esempio ribadisce che **l’ELSE non è sempre obbligatorio nell’ambito di una IF**.

ATTENZIONE: **prima dell’ ELSE non ci vuole mai il “punto-e-virgola”** (errore frequente!)
Il compilatore si arresterebbe fornendo il “Syntax error” che segue: *“;” expected but “ELSE” found*.

```
program equazioni_strane_1;  uses crt;
var a, b, x: real;
begin
    clrscr;
    writeln (’Risolviamo l’’equazione ax=b’);
    write (’a = ’); readln (a); write (’b = ’); readln (b);
    IF a<>0 THEN
                  begin
                     x:=b/a;
                     write (’x = ’, x);
                  end
               ELSE
               IF b<>0 THEN write (’Eq. Impossibile’)
                         ELSE write (’Eq. Indeterminata’);
    readln;
end.
```

Questo programma ci dà un esempio di due IF … THEN … ELSE … contenute una dentro l’altra, cioè “ANNIDATE”

```
program equazioni_strane_2;  uses crt;
var a, b, x: real;
begin
    clrscr;
    writeln (’Risolviamo l’’equazione ax=b’);    (*NOTA*)
    write (’a = ’); readln (a); write (’b = ’); readln (b);
    if a<>0 then write (’x = ’, b/a);
    if (a=0) AND (b<>0) then write (’Eq. Impossibile’);
    if (a=0) AND (b=0)  then write (’Eq. Indeterminata’);
    readln;
end.
```

L’uso dell’operatore logico AND ci ha permesso di scrivere il programma in modo più semplice.

Gli operatori logici in Pascal sono:
AND, OR, NOT, XOR
(XOR = disgiunzione esclusiva:
***p* XOR *q* è vera se e solo se**
è vera una e una sola delle due condizioni *p, q*)

Esercizio 1)

Nel 1969 l’uomo è sbarcato per la prima volta sulla Luna.
Scrivi un programma PASCAL che domandi all’utente il suo anno di nascita e fornisca, a seconda dei casi, l’output:

i) “Quando l’uomo sbarcò sulla Luna tu avevi … anni” (NOTA)
ii) “Quando l’uomo sbarcò sulla Luna tu non eri ancora nato: saresti nato … anni dopo”
iii) “Caspita, ma sei nato proprio l’anno in cui l’uomo sbarcò sulla Luna!!!”

NOTA
Gli apostrofi nelle stringhe vengono indicati tramite un doppio apice:
vedi pag. 9

5 - VARIE

a)  ERRORI FREQUENTI E LORO CORREZIONE

OCCHIO! Fra i più frequenti, "tipici" errori formali nella redazione di un programma citiamo i seguenti:

- **dimenticare il ";" alla fine di una istruzione**.
  OGNI ISTRUZIONE O DICHIARAZIONE (NOTA)
  DEVE OBBLIGATORIAMENTE TERMINARE COL ";" !!!
  Uniche eccezioni l'*end* finale (seguito da un ".") e, volendo, i vari *begin* (";" facoltativo)

  Il **messaggio di errore** che compare è in questo caso:
  ";" expected ossia **ci si aspettava un ";"**
  Nel messaggio troviamo pure l'**indicazione di "cosa è stato invece trovato (found)".**

  NOTA: le "dichiarazioni" sono, ad esempio:
  il "program …", la "uses crt", la specificazione del *tipo* per ogni variabile.

  QUANDO IL COMPILATORE SEGNALA UN ERRORE,
  simultaneamente indica pure
  LA RIGA E LA COLONNA IN CUI E' STATO RISCONTRATO L'ERRORE !!!
  … E QUESTO TI AIUTA TANTISSIMO, PER LA CORREZIONE !!!

- **mettere il ";" prima dell'ELSE** di una *if … then … else …* (in questo caso, non ci vuole!)
  Qui il **messaggio di errore** è:
  ";" expected" but "ELSE" found

- **dimenticare il "." dopo l' "end" conclusivo**
  Messaggio di errore:
  "." expected but "end of file" found

b) OPERAZIONI E SIMBOLI MATEMATICI

somma, sottrazione	**+, –**		
moltiplicazione	*		
divisione	/	OCCHIO! Il risultato di una "/" va sempre incasellato in una variabile di tipo *reale*! … Altrimenti la compilazione si arresterebbe, con un messaggio di errore.	
divisione intera	**div**	Può operare solo con variabili di tipo *intero*: *integer* o *longint*.	
resto della divisione intera	**mod**	(può operare solo con variabili di tipo *intero*!)	
radice quadrata	**sqrt**	**Esempio: y:=sqrt(x)**	
Altre operazioni (es. potenza …)	Volutamente non ne parliamo a questo livello. Richiedono una conoscenza più avanzata di Free Pascal. A questo proposito, i vari "HELP" si possono scaricare da www.freepascal.org/down/docs/docs.html Fra l'altro, una potenza ad esponente intero si può ottenere servendosi della moltiplicazione; se l'esponente è una *variabile* intera possiamo ottenere lo scopo con un programmino, che sarà richiesto a suo tempo come esercizio.		
diverso da	**<>**		
minore o uguale, maggiore o ug.	**<=, >=**		
parte intera di	**int**	**Esempi: int(4.72) dà come risultato 4; int(–3.8) dà –3**	

Esercizio 2) Realizza un programma PASCAL che, letto in ingresso un numero x, fornisca in output:

- ♪ la radice quadrata di x, in notazione NON ESPONENZIALE, se x>=0;
- ♫ la scritta "Mi spiace, ma non esiste la radice quadrata di un numero negativo" se x<0.

c) L'OVERFLOW (= traboccamento)

Si chiama così la **fuoriuscita di una variabile dal suo "range" (=campo di variabilità)**.
Ad esempio, se abbiamo dichiarato una variabile di tipo *integer* (*range* da –32768 a +32767)
e a questa variabile viene attribuito, o direttamente da una istruzione di assegnazione,
o in input, o per l'effetto di un calcolo, un valore fuori dal *range* (poniamo: il valore 50000),
allora in fase di compilazione o di esecuzione verrà segnalato un errore: il programma non funzionerà.

d) COME SALTARE UNA RIGA SUL MONITOR, IN FASE DI ESECUZIONE

L'istruzione
writeln
provoca l'effetto di **spostare il cursore all'inizio della riga successiva**;
se il cursore SI TROVA GIA' all'inizio di una riga vuota,
l'effetto sarà di passare all'inizio della riga successiva a questa, e quindi di **saltare una riga**.

e) APOSTROFO NELLE STRINGHE: CHE GUAIO! IL COMPUTER LO CONFONDEREBBE CON IL SIMBOLO DI "FINE STRINGA"!

Come fare, allora? I creatori del PASCAL hanno previsto che si operi come nell'esempio seguente:

write ('L''amicizia è una cosa stupenda')

Per realizzare l'**apostrofo all'interno di una stringa** basterà dunque digitare il **doppio apice**.

f) TESTO COLORATO IN OUTPUT

L'istruzione per ottenere in output un testo colorato è
textcolor (n),
dove **n è il codice del colore desiderato**
(**0 ≤ n ≤ 255, ma oltre il 15 si ripete la sequenza di colori precedente**).

Ad esempio:
9 = blu, 10 = verde, 11 = azzurro,
12 = rosso, 14 = giallo, 15 = bianco.

Per conoscere le altre corrispondenze codice-colore, possiamo scrivere un programmino apposito, ad es. un programma che, letto in input un intero, lo restituisca in output, colorato del colore che gli corrisponde →

```
program codici_dei_vari_colori;   uses crt;
var n: integer;
begin
     clrscr;
     writeln ('Scrivi un intero fra 0 e 15.');
     write ('n = ');  readln (n);
     writeln ('Colore corrispondente:');
     textcolor (n);  write (n);  readln;
end.
```

g) L' EFFETTO RITARDO

L'istruzione **delay (n)**, dove n è un numero di tipo intero,
ordina al computer di attendere n "unità di tempo" prima di eseguire l'istruzione successiva.
L' "unità di tempo" dovrebbe essere in teoria di 1 millesimo di secondo, ma in realtà differisce un po' da un computer all'altro. Si faranno delle prove, poi si sceglierà n a seconda dell'effetto che si vuol realizzare.

h) I "COMMENTI"

Nel "listato" (=sequenza delle istruzioni) di un programma si possono scrivere dei "**commenti**", **che verranno ignorati nella fase di "compilazione" (=trascrizione in linguaggio macchina).**

I commenti devono essere inseriti

- **fra "parentesi asteriscate": (* questo è un commento *)**
- **oppure fra parentesi graffe: { questo è un commento }**

OSSERVAZIONE - Nel linguaggio C le graffe hanno invece un ruolo completamente diverso: stanno a indicare inizio blocco e fine blocco (hanno cioè la funzione che in PASCAL compete a BEGIN e END)

i) LA SELEZIONE MULTIPLA: CASE … OF …

Quando le alternative sono più di due, potrà essere utile la CASE … OF … ,
una struttura di **SELEZIONE MULTIPLA** illustrata dal seguente esempio:

```
CASE punteggio OF
        0, 1, 2: writeln ('Vai a zappare');
        3, 4: writeln ('Gravemente insufficiente');
        5: writeln ('Insufficiente');
        6, 7, 8, 9, 10: writeln ('Esame superato');
END;
```

6 - NUMERI CASUALI (O MEGLIO, "PSEUDOCASUALI")

Nell'istruzione

bigliettino:=random (25)

bigliettino è una variabile di tipo *intero*;
l'istruzione le assegna un valore casuale,
o meglio "*pseudo*casuale" (vedi il riquadro qui a destra)
che potrà essere 0, 1, 2, 3, … , 23, 24.

In generale, **in PASCAL, la funzione RANDOM (n)**
genera un intero pseudocasuale che potrà valere: 0, 1, 2, … , n–1.

Due esempi significativi:

- **esito:=random(2)**
  assegnerà alla variabile *esito* o il valore 0 o il valore 1
  perciò si presta a **simulare il lancio di una moneta**
  (0 si potrà interpretare come "Testa" e 1 come "Croce", o viceversa)
- **x:=random(6)+1**
  assegnerà alla variabile x uno dei valori 1, 2, 3, 4, 5 o 6
  perciò si presta a **simulare il lancio di un dado**.

Invece la funzione RANDOM, usata senza alcun argomento,
genera un numero pseudocasuale di tipo *reale*,
compreso fra 0 (incluso) e 1 (escluso).

Esempio: l'istruzione **y:=random** assegna alla variabile y,
che deve essere stata dichiarata di tipo *reale*,
un valore pseudocasuale ≥ 0 e < 1.

COSA VUOL DIRE
"PSEUDOCASUALE"
?

L'aggettivo "*pseudo*casuale"
esprime il fatto che in realtà,
quando il computer produce
tutta una successione
di numeri di questo tipo
uno dopo l'altro,
quello *davvero casuale…*
… è *solo il primo*
(*detto "il seme"*
della sequenza),
perché si basa su di un valore
preso dal *clock*
nel preciso istante in cui
la procedura viene avviata,
mentre *i successivi*
vengono *calcolati*
mediante appositi algoritmi
(scelti in modo tale
da assicurare "l'apparenza",
ma non dunque la "sostanza",
della casualità).

IMPORTANTE

Occorre ricordarsi, tutte le volte che in un programma Pascal si utilizza una funzione *random*,
di **premettere**, dopo il "begin" e **prima dell'istruzione che contiene la RANDOM**,
l'istruzione

RANDOMIZE;

essa **ordina al computer di**
"**rendere casuale il seme per la generazione dei numeri pseudocasuali**"
(a tale scopo viene impiegato un valore fornito in quell'istante dal *clock* del sistema)

E' un po' come scuotere preventivamente l'urna da cui si estrarranno le palline;
se non lo si fa, la pallina estratta, quando viene posata,
resterà sempre in superficie e continuerà ad essere ripescata.

Esercizio 3)

Il fratellino deve ripassare le operazioni aritmetiche? Scrivi un programma Pascal che operi nel seguente modo:

a) sul monitor deve comparire la stringa: BAMBINO, CHE OPERAZIONE VUOI FARE? + OPPURE * ?

b) a questo punto, evidentemente, il bambino digiterà il simbolo + o in alternativa il simbolo *
e il computer "leggerà" il carattere scelto (NOTA: dovrai allora prevedere una variabile
di tipo STRING[1], oppure di tipo CHAR, la quale renda possibile la lettura del simbolo);

c) poi il computer deve "estrarre" due numeri interi pseudocasuali x, y, ciascuno compreso fra 0 e 10,
e mandare sul monitor i numeri stessi, separati dall'operazione che il bambino aveva scelto;
quindi, ad esempio,
9*7 =

d) Il bambino scrive la sua risposta, il computer ne valuta la correttezza e manda in output la stringa
GIUSTO! oppure SBAGLIATO a seconda dei casi.

Insomma, al termine dell'esecuzione il monitor deve apparire (ad esempio) così:

```
BAMBINO, CHE OPERAZIONE VUOI FARE? + OPPURE * ?
+
6+8=15
SBAGLIATO
```

dove, evidentemente, il bambino ha digitato esclusivamente il simbolo + della seconda riga e il numero 15,
mentre tutto il resto lo ha scritto il computer.

7a - GLI OPERATORI "DIV" E "MOD"; PARI? DISPARI? DIVISIBILE PER …? DIVISORE DI …?

GLI OPERATORI "DIV" E "MOD" forniscono il quoziente intero e il resto della divisione intera.
a DIV b dà il QUOZIENTE INTERO, a MOD b dà il RESTO.
a, b devono essere numeri, o variabili, di tipo INTERO (con b diverso da 0). Esempi:

30 DIV 7 = 4 ("30 diviso 7" dà 4; poi c'è anche un resto, ma in questo momento non ci interessa)
30 MOD 7 = 2 (il *resto* della divisione "30 diviso 7" è 2)
15 DIV 3 = 5; 15 MOD 3 = 0; 4 DIV 7 = 0; 4 MOD 7 = 4

Per fare esercizi, trascrivi e manda in esecuzione per alcune volte il seguente programmino:

```
program esercizi_sul_div_e_sul_mod;   uses crt;
var a, b, x, y, i : integer;
begin
    clrscr;
    randomize;
    a:=random (100);  b:=random (10)+1;
    write ( a, ' div ', b, ' = ' ); readln (x);   write ( a, ' mod ', b, ' = ' ); readln (y);
    if (x = a div b) and (y = a mod b) then writeln ('OK')
                                       else
                                           begin
                                                writeln ('NO. Risultati esatti: ');
                                                writeln ( a, ' div ', b, ' = ', a div b);
                                                writeln ( a, ' mod ', b, ' = ' , a mod b);
                                           end;
    readln;
end.
```

PARI? DISPARI? DIVISIBILE PER …? DIVISORE DI …?

if a mod b = 0 … significa (come preferisci):

□ **"se a è divisibile per b …"** □ **"se a è multiplo di b …"** □ **"se b è divisore di a …"**

e perciò:

♪ **if x mod 2 = 0 …** significa: **"se x è pari …"**
♫ **if x mod 2 = 1 … (oppure: if x mod 2 <>0)** significa: **"se x è dispari …"**

NOTA: "diverso da" in PASCAL ha come simbolo <>

7b - ANCORA SULLE VARIABILI REAL: CIFRE SIGNIFICATIVE, UNDERFLOW …

Per le variabili di tipo *reale*, destinate ad assumere valori numerici non necessariamente interi, oltre che il discorso sul *range* (= intervallo di variabilità) è importante anche quello relativo al *numero massimo di cifre significative* supportate.
Lo specchietto seguente è tratto dalla documentazione on-line presente su www.freepascal.org:

Type	Range	Significant digits	Size (*byte occupati*)
Real	platform dependant	???	4 or 8
Single	1.5E – 45 .. 3.4E38	7-8	4
Double	5.0E – 324 .. 1.7E308	15-16	8
Extended	1.9E – 4932 .. 1.1E4932	19-20	10

E' evidente che il numero limitato di cifre significative utilizzabili può costringere il computer ad effettuare delle approssimazioni, a seguito delle quali è possibile che si determinino risultati imperfetti.

Tornando poi ad occuparci del *range*, osserviamo che nel caso in cui il valore di una variabile numerica di un determinato "tipo" andasse a superare il massimo del *range* che compete a quel tipo, si avrebbe un *overflow* (traboccamento) e il programma, di norma, si arresterebbe con una segnalazione di errore.

Se al contrario tale valore diventasse (in valore assoluto) troppo piccolo, l'errore sarebbe di *underflow* e purtroppo non verrebbe segnalato: il destino dei numeretti "troppo piccoli" è semplicemente di essere approssimati a 0. E anche da ciò possono derivare alterazioni per i risultati forniti dal programma.
I quali dunque, in determinate situazioni, devono essere interpretati con senso critico, alla luce di quanto detto.
Questo discorso può riguardare ad esempio i programmi per l'approssimazione di π proposti a pagina 23.

8 - LA STRUTTURA ITERATIVA (= di “iterazione”, cioè “ripetizione”) FOR … DO …

La struttura

FOR i:= n1 TO n2 DO istruzione

oppure:

```
FOR i:= n1 TO n2 DO
                    BEGIN
                            istruzione 1;
                            istruzione 2;
                            istruzione 3;
                            …
                    END;
```

ordina al computer di:

- **assegnare il valore n1 alla variabile i** (NOTA);
- **eseguire** l'istruzione che segue il DO (oppure, il blocco di istruzioni compreso fra il BEGIN che segue il DO e l' END successivo);
- **incrementare di una unità il valore di i**;
- **eseguire** nuovamente l'istruzione (o il blocco);
- **incrementare di un'altra unità i**;
- **eseguire** nuovamente l'istruzione (o il blocco);
- ...

Quando, a forza di incrementi di un'unità, la variabile i assume il valore n2, viene eseguita per l'ultima volta l'istruzione (o il blocco), poi si esce dal ciclo.

NOTA: ho usato la lettera i per fissare le idee.
Al posto di i si può mettere un qualunque identificatore di variabile *intera*.
La variabile del ciclo FOR viene detta “**variabile di controllo**”, e dev’essere sempre di tipo *intero*.

ESEMPI di programmi con l’uso della FOR … DO …

```
program somma_dei_numeri_naturali_da_1_fino_a_n;     uses crt;
var k, n, somma: longint;
(*  in questo programma la variabile k funge da "CONTATORE"
    e la variabile somma da "ACCUMULATORE"  *)
begin
        clrscr;
        writeln (’Dammi un numero intero positivo n;’);
        writeln (’ti scriverò la somma dei numeri interi da 1 fino ad n.’);
        readln (n);
        somma:= 0;
        for k:= 1 to n do somma:=somma+k;
        write (’La somma che ti avevo promesso vale ’, somma);
        readln;
end.
```

```
program massimo;     uses crt;
var a, n, k, max: longint;
(*  questo programma riceve in input dei numeri e stabilisce quale è stato il più grande fra i numeri introdotti.
    La variabile max svolge il ruolo di "MASSIMO PROVVISORIO";
    il suo valore finale corrisponderà al massimo cercato  *)
begin
        clrscr; writeln (’Quanti numeri vuoi introdurre?’); readln (n);
        writeln (’Dammi pure questi  ’, n, ’  numeri: ti dirò qual è il massimo’);
        max:=0;
        for k:= 1 to n do
                        begin
                                readln (a);
                                if a>max then max:=a;
                        end;
        writeln (’Massimo = ’, max);
        readln;
end.
```

ESERCIZI

Esercizio 4)

Scrivi un programma che, letto da tastiera un numero intero positivo n,
fornisca in output, su tre colonne:

a) **i numeri interi da 1 fino a n;**
b) **i rispettivi quadrati;**
c) **le rispettive radici quadrate (in notazione non esponenziale)**.

Supponendo, per fissare le idee, n = 4, l'output desiderato è:

NUMERO	QUADRATO	RADICE QUADRATA
1	1	1.00000
2	4	1.41421
3	9	1.73205
4	16	2.00000

Esercizio 5)

Letto in input un intero positivo n, si vuole in output il valore della somma 1+1/2+1/3+1/4+ ... +1/n.

Esercizio 6)

**Letti in ingresso un numero qualunque a ed un numero intero positivo n,
si vuole in output l' n-esima potenza di a**, ossia il numero a^n .

*E' indispensabile, in questo programma,
una variabile accumulatore,
il cui valore FINALE sarà la potenza desiderata.*

Esercizio 7)

**Letti in ingresso n numeri (n è fornito in input dall'utente)
in parte positivi e in parte negativi,
che rappresentano gli esiti (in euro) di altrettante partite di poker,**

- **si conta quanti sono quelli negativi e se ne fa la media aritmetica;**
- **si conta quanti sono quelli positivi e se ne fa la media aritmetica.**

L'output dovrà essere:

Hai perso … volte, perdendo in media … euro ogni volta;
e hai vinto … volte, vincendo in media … euro ogni volta.
In totale, hai (perso oppure vinto) … euro.

*Non è evidentemente necessario utilizzare tante variabili diverse
per ciascuno dei numeri in ingresso;
basterà*

- *una sola variabile per la lettura;*
- *una variabile accumulatore per la somma dei negativi
e una variabile accumulatore per la somma dei positivi;*
- *una variabile contatore per i negativi e una variabile contatore per i positivi;*
- *oltre, naturalmente, alla variabile di controllo della struttura FOR ... DO ...*

Esercizio 8)

Scrivi un programma che mandi **in output tutti gli interi da 0 a 30,
ciascuno colorato del colore di cui l'intero considerato rappresenta il codice**
(vedi paragrafo 5f, "Testo colorato in output").

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

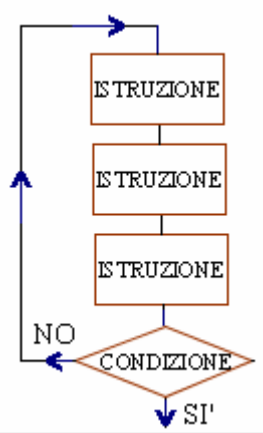
la FOR … DO …
è utilizzabile solo quando è noto fin dall'inizio
***quante volte* l'istruzione (o il blocco) andrà ripetuto.**

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:

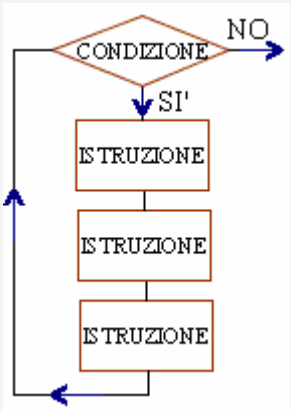
a) **il blocco di istruzioni viene certamente eseguito almeno una volta**;
b) il **controllo** se la condizione sia verificata o meno avviene **alla fine** del ciclo;
c) l' **uscita** dal ciclo si ha **quando la condizione E' verificata**

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 ...

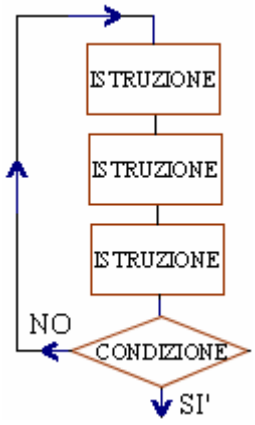
Esempio: *L'utente sceglie un numero intero n compreso fra 1 e 100.*
La macchina "estrae a sorte", ripetutamente, un intero x (1≤ x ≤ 100);
si contano i "tentativi" che farà la macchina prima di estrarre proprio n.

```
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.
```

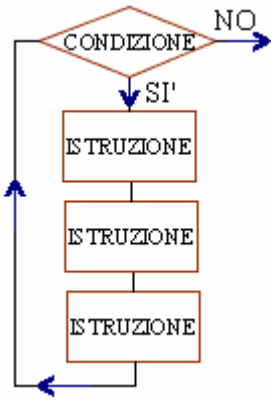
SCHEMA RIASSUNTIVO

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.
```

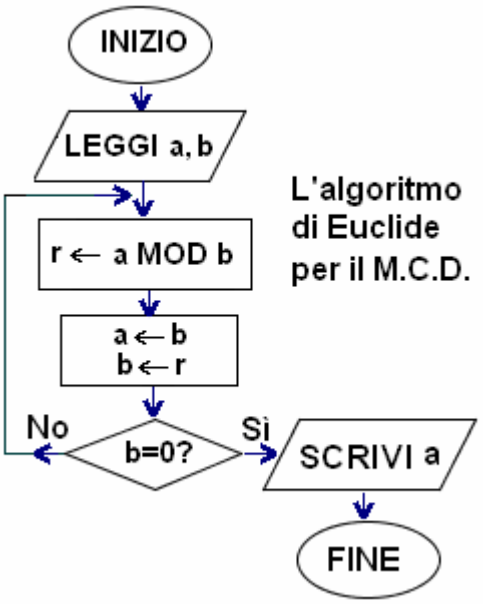
Lascio a te il compito
di vedere come varia,
istruzione dopo istruzione,
il contenuto delle tre scatolette

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**



10 - ESERCIZI SULLE STRUTTURE ITERATIVE

Esercizio 9)

Riprendi il programma

program **somma_di_un_numero_non_prefissato_di_addendi**

(vedi paragrafo precedente)

e ribattezzalo

program **prodotto_di_un_numero_non_prefissato_di_fattori**

modificandolo affinché esegua, non più la somma, bensì il *prodotto* dei numeri letti da tastiera.

Esercizio 10)

Realizza un programma (program **margherita**), che faccia comparire sullo schermo il seguente output:

1 M'AMA
2 NON M'AMA
3 M'AMA
4 NON M'AMA
……
n ...

essendo n un numero (pseudo)casuale compreso fra 1 e 20.

INDICAZIONI

Puoi usare la struttura iterativa che ritieni più opportuna,
ma successivamente ti invito a rifare il programma altre due volte,
utilizzando, come utile esercizio, le due strutture iterative rimanenti.

Ricordiamo che il "test di parità"
si effettua mediante l'operatore MOD (che dà il resto della divisione intera):

if a mod b = 0 significa: **"se a è divisibile per b"** e perciò:
if x mod 2 = 0 significa: **"se x è pari"**
if x mod 2 <> 0 oppure **if x mod 2 =1** significa: **"se x è dispari"**

Sarà opportuno inserire un "**ritardo**" fra un "petalo" e l'altro.
Vedi a questo proposito, al paragrafo 5g, "L' effetto ritardo".

Esercizio 11)

Scrivi **un programma che, letta da tastiera una sequenza di numeri qualsiasi,**
scriva sullo schermo il minimo ed il massimo fra i numeri introdotti.
Per indicare che la sequenza dei numeri in input è terminata, l'utente digiterà il numero 0 ("sentinella").

Attenzione!
La "sentinella" 0 segnala la fine della sequenza,
ma *non fa parte* dell'insieme dei numeri di cui si desidera conoscere il massimo e il minimo.
Occhio perciò a metter giù il programma correttamente.
Converrà preliminarmente far leggere il primo numero della sequenza,
per assegnare come valore iniziale a ciascuna delle due variabili
massimoprovvisorio e *minimoprovvisorio*,
il valore del numero letto.

Esercizio 12)

Scrivi **un programma che si occupi di far fare al fratellino piccolo 10 esercizi sulle tabelline**
(un po' come nell'esercizio 3 del paragrafo 6).
Il computer, di fronte ad una risposta sbagliata,
deve continuare a riproporre la stessa moltiplicazione fino a quando il bambino risponde esattamente.
I fattori per ciascun esercizio dovranno essere (pseudo)casuali
(ciascun fattore potrà andare da 0 a 10).
Il programma dovrà anche contare il numero complessivo di risposte sbagliate.

Esercizio 13)

Letto in ingresso un intero n, il programma ne elenca i divisori e li conta,
includendo anche i divisori "impropri" (che sono il numero stesso e l'unità).

Esercizio 13')

Completare il programma precedente, in modo che fornisca in output anche la scritta
IL NUMERO E' PRIMO oppure IL NUMERO NON E' PRIMO, a seconda dei casi.
E' chiaro che la prima circostanza si verifica quando i divisori sono soltanto 2 (l'unità e il numero).

Esercizio 14)

Letto in ingresso un intero positivo n, stabilire se si tratta di un numero primo.
Questa volta, però, ti chiedo di **RIDURRE il numero di passi**
che la macchina dovrà compiere prima di fornire la risposta.
A tale scopo, non si stanno a individuare e contare *tutti* i divisori di *n*;
si prova invece a vedere se *n* è divisibile per 2, poi per 3, ecc., fermandosi quando

a) si trova un divisore
b) oppure (operatore logico OR) il numero che è "candidato" ad essere un divisore di *n*
ha già superato sqrt(*n*), ossia la radice quadrata di *n*
(infatti si dimostra che, se in tal caso non sono ancora stati trovati divisori propri, *n* non potrà avere alcun divisore proprio; i divisori "propri" sono quelli diversi dall'unità e dal numero stesso).

Esercizio 14')

Integra il programma precedente per far sì che in output venga fornita
la **sequenza di tutti i numeri primi non superiori a MAX, essendo MAX un intero letto in ingresso**.
Considera la possibilità di mandare i numeri ordinatamente in output senza andare a capo: ad es., se utilizzi **write(x:8)** l'intero x verrà scritto sul monitor allineandolo a destra su di un campo di 8 caratteri.
Tieni conto del fatto che **ogni riga della finestra FREE PASCAL contiene esattamente 80 caratteri.**
Variante: un programma che scriva la sequenza degli interi, coi soli numeri primi colorati in rosso.

Esercizio 15)

PREMESSA Si dice "**fattoriale**" di un intero n>0, e lo si indica con **n! (leggi: "n fattoriale")**,
il prodotto dei fattori interi decrescenti da n fino a 1:
$n! = n \cdot (n-1) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$. Ad esempio: $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$

Scrivi **un programma che, letto in input n, calcoli n!**

Esercizio 16)

Gara di testa-e-croce.
Ricordiamo che un'istruzione del tipo moneta:=random(2) ha l'effetto di assegnare
alla variabile "moneta" (di tipo intero) il valore 0 oppure il valore 1, in modo (pseudo)casuale.
Se interpretiamo, ad esempio, il valore 0 come "testa" e il valore 1 come "croce",
avremo così "simulato" il lancio di una moneta.
Si lancia dunque **per n volte una moneta (con n letto in ingresso)**
e si vuole stabilire quante volte è uscita "testa" (cioè, 0) e quante volte "croce" (cioè, 1).
E' opportuno far sì che la sequenza di "colpi" compaia in output.

Esercizio 17)

Si lancia un dado più volte; ci si ferma dopo che è uscito per esattamente 100 volte il numero 6.
Si vuole sapere quanti lanci sono stati effettuati in totale.

Esercizio 18)

Scrivi un programma che, letti in ingresso due numeri interi a, b, ne calcoli il **minimo comune multiplo**.
Ci sono diversi modi alternativi per impostare l'algoritmo;
ad esempio, si può calcolare la successione dei multipli di a ($a, 2a, 3a, 4a, \ldots$), arrestandosi quando si perviene ad un numero che risulta multiplo anche di b (cioè, che risulta divisibile per b).
Ricorda che **in PASCAL l'asterisco di moltiplicazione NON può essere lasciato sottinteso.**

Esercizio 19)

Scrivi un programma che, letti in ingresso due numeri interi a, b, ne calcoli il **massimo comun divisore**.
Puoi utilizzare **diversi metodi alternativi**, fra cui l' "algoritmo di Euclide". Scegli quello che desideri.

Esercizio 20)

Dato un intero n>1, scomporlo in fattori primi.

Esercizio 21)

Letti in input i due termini a, b di una frazione, ridurre la frazione data ai minimi termini.

Ad esempio, se l'input è	180 450	l'output dovrà essere	2 5

11 - ESERCIZI VARI

Può darsi che in certi casi tu trovi opportuno tracciare
il diagramma di flusso dell'algoritmo prima di metterti a scrivere il programma;
in tal caso, ricorda la forma che convenzionalmente si assegna ai vari blocchi:
OVALE per INIZIO/FINE; PARALLELOGRAMMO per INPUT/OUTPUT;
RETTANGOLO per ELABORAZIONE; ROMBO per SELEZIONE.

Iniziare o meno col diagramma di flusso
dipende dalla tua volontà, oltre che dal tipo di esercizio.

Esercizio 22)

Si simula ripetutamente il lancio di un dado, mediante l'istruzione x:=random(6)+1,
e ci si arresta non appena esce il numero 6. Quanti "colpi" sono stati effettuati in totale?

Esercizio 23)

PREMESSA
Un intero si dice "perfetto" se è uguale alla somma dei suoi divisori,
inclusa l'unità ma escluso il numero stesso:
ad esempio 28 è un numero perfetto,
perché ha come divisori (a parte sé stesso) 1, 2, 4, 7, 14, e $1+2+4+7+14=28$.

Letto in input un intero n, stabilire se n è un numero "perfetto".

Esercizio 24)

Elenco **dei numeri perfetti $\leq$ m**, con m letto in ingresso.

Esercizio 25)

Elenco degli interi da a fino a b, con a, b letti in input,
coi numeri PRIMI scritti in rosso, i PERFETTI in verde (textcolor).
Utilizza una opportuna "ampiezza" (vedi paragrafo 3) per collocare i numeri ordinatamente sul monitor:
ad es., write(n:10) fa scrivere l'intero n allineato a destra, in un campo la cui *ampiezza* è di 10 caratteri.

Esercizio 26)

Il computer "inventa" un intero pseudocasuale non superiore a 100
e invita l'utente a indovinarlo, dandogli 7 tentativi al massimo.
Ogni volta che l'utente "tenta", il computer gli dice
"più alto…" oppure "più basso…" o "giusto!!!" a seconda dei casi.
Il gioco si deve arrestare quando l'utente indovina,
oppure quando sono stati effettuati i 7 tentativi concessi.

Esercizio 27)

Si lancia un dado finché escano due risultati consecutivi uguali,
e si conta il numero di colpi effettuato.

Esercizio 28)

Programma "**fuochi_di_artificio**". Output desiderato:

……..BOOM!
…BOOM!
……BOOM!
…

con 10 "esplosioni"
e un numero pseudocasuale di puntini prima di ogni esplosione.
Ci deve essere un ritardo opportuno fra la comparsa di ciascun puntino e il successivo,
nonché fra una riga e la successiva.
Vogliamo far comparire la scritta BOOM! in colore, anch'esso (pseudo)casuale.
Ricordiamo che a tale scopo si ricorre all'istruzione textcolor (n) dove n è il codice del colore desiderato.

Esercizio 29)

Scrivi un programma che, letti in ingresso due interi positivi y, k, calcoli la **somma algebrica**

$$\mathbf{1 - y + y^2 - y^3 + \ldots \pm y^k}$$

Ad esempio, se y=2 e k=5, l'output dovrà essere il numero –21.

Esercizio 30)

PREMESSA: le equazioni di 2° grado

$x_{1,2} = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ è la FORMULA RISOLUTIVA DELL'EQUAZIONE $ax^2 + bx + c = 0$

Esempio di applicazione della formula:

$$3x^2 + x - 2 = 0 \qquad (a = 3,\ \ b = 1,\ \ c = -2)$$

$$x_{1,2} = \frac{-1 \pm \sqrt{1^2 - 4 \cdot 3 \cdot (-2)}}{2 \cdot 3} = \frac{-1 \pm \sqrt{1 + 24}}{6} = \frac{-1 \pm \sqrt{25}}{6} = \frac{-1 \pm 5}{6} = \begin{cases} \dfrac{-1-5}{6} = -\dfrac{6}{6} = -1 \\ \dfrac{-1+5}{6} = \dfrac{4}{6} = \dfrac{2}{3} \end{cases}$$

La quantità $\mathbf{b^2 - 4ac}$ che sta sotto radice quadrata nella formula risolutiva, è chiamata "delta" o "discriminante" e indicata col simbolo Δ (la lettera greca "*delta*" maiuscola).

Sono possibili tre casi:

- se $\Delta > 0$, l'equazione ha due soluzioni distinte
- se $\Delta = 0$, l'equazione ha una sola soluzione (si può anche dire che ha "due soluzioni coincidenti")
- se $\Delta < 0$, l'equazione non ha nessuna soluzione (= è impossibile)

Scrivi un programma che,
letti in input i 3 coefficienti a, b, c di un'equazione di 2° grado, ne fornisca le soluzioni.

Converrà far calcolare innanzitutto il delta; poi:

```
if delta > 0 then … ;
if delta = 0 then … ;
if delta < 0 then …
```

Le variabili a, b, c, delta dovranno essere di tipo *real* e si vuole l'output in notazione non esponenziale.

Esercizio 31)

Scrivi un programma per la risoluzione di un sistema di 1° grado **a*x+b*y=c, a1*x+b1*y=c1** **col metodo di Cramer**.

L'utente verrà invitato a inserire semplicemente i 6 numeri a, b, c, a1, b1, c1.

Esercizio 32)

Si chiama "**successione di Fibonacci**" la sequenza 0 1 1 2 3 5 8 13 21 …
costruita nel modo seguente:

- i primi due numeri della sequenza sono 0 e 1 rispettivamente;
- a partire dal terzo, ciascun termine della sequenza è ottenuto sommando i due termini che lo precedono.

Scrivi **un programma che, letto in input n, scriva i primi n termini della successione di Fibonacci.**

Esercizio 33)

Un metodo per calcolare la radice cubica di un numero, approssimata per difetto a meno di 0.1 (cioè, con la prima cifra decimale esatta), è il seguente:

i) dato il radicando x, si parte da un valore intero y che sia una approssimazione per difetto di $\sqrt[3]{x}$;
ii) poi si passa a considerare, via via, i numeri y; y+0.1; y+0.2; y+0.3; …
ciascuno di questi numeri viene elevato al cubo, finché il risultato superi x.

Allora il valore cercato sarà il *penultimo* fra i numeri considerati.

Ad esempio, data l'operazione $\sqrt[3]{70}$, si partirà da $y = 4$ e si farà:

$4^3 = 64 < 70$; poi $4.1^3 = 68.921 < 70$; poi $4.2^3 = 74.088 > 70$; quindi il valore cercato è 4.1.

Scrivi un programma che, letti in input il radicando x (la variabile x dovrà essere di tipo *real*)
e un numero intero, fornito dall'utente, che sia approssimazione per difetto di $\sqrt[3]{x}$,
determini e mandi in output il valore $y = \sqrt[3]{x}$ approssimato per difetto a meno di 0.1.

AVVERTENZA (INTERESSANTE):
può darsi che a volte il valore in output non sia quello corretto.
Ad esempio, con x = 125, l'output anziché essere 5.0 potrà essere inaspettatamente 5.1.
Per comprendere il motivo di questo strano fenomeno, vai a rileggere il paragrafo 7b a pag. 11.

Esercizio 34)

Un intero n letto in input è sottoposto alla seguente procedura:

- lo si fa diventare $3n+1$ se è dispari;
- lo si fa diventare la metà, ossia $n \text{ div } 2$, se è pari.

Il procedimento viene poi iterato sul nuovo numero ottenuto,
fino ad arrestarsi quando si ottenga il valore 1.
Ad esempio, se $n = 7$, in questo modo si genera la sequenza:

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

I numeri con cui si è provato finora ad innescare questa procedura iterativa si sono tutti "abbattuti a 1"
dopo un numero più o meno alto di passi;
la comunità matematica sta tentando di provare che questo "abbattersi a 1"
deve necessariamente aver luogo per *ciascun* numero intero,
ma fino ad oggi questa congettura non è ancora stata dimostrata.

**Scrivi un programma PASCAL che, letto in input un intero n fornito dall'utente,
mandi in output la successione dei numeri che si ottengono sottoponendo n alla procedura descritta.
L'algoritmo si deve arrestare quando n diventa uguale a 1,
e deve contare il numero di passi che sono stati necessari.**

Esercizio 35)

Se tre interi a, b, c sono tali che $a^2 + b^2 = c^2$,
allora si dice che a, b, c costituiscono una "terna pitagorica".

**Utilizzando delle strutture FOR … DO … "annidate",
dovresti riuscire a scrivere un programma che fornisca in output un elenco di terne pitagoriche;
ad es., tutte le terne pitagoriche nelle quali c sia non superiore ad un numero N scelto dall'utente.**

Esercizio 36)

Una terna pitagorica si dice "fondamentale"
se i suoi 3 elementi sono numeri primi fra loro (= privi di divisori comuni).
Ad esempio, la terna (5, 12, 13) è fondamentale, mentre la (6, 8, 10) non lo è.

**Modifica il programma precedente in modo che le terne fondamentali
compaiano, nell'elenco, scritte in rosso: textcolor (12).**

Tieni conto del fatto che, in una terna (a, b, c) tale che $a^2 + b^2 = c^2$,
i tre numeri a, b, c sono primi fra loro se e solo se sono primi fra loro due qualsiasi di essi
(sapresti giustificare questa affermazione?)

Esercizio 37)

Si lancia n volte una moneta, con n letto in ingresso.
Si vuole stabilire quale è stato il **numero massimo di risultati consecutivi uguali**.
Ad esempio, se la successione dei lanci è stata

1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 1,

la risposta è "4".

Esercizio 38)

Per noi esseri umani è facile, **letto un numero intero a, calcolare quanto vale la somma delle sue cifre**.
Ad esempio, se $a = 30227$, la somma delle cifre di a è 14.
Non è invece altrettanto immediato far sì che *il computer*, dopo aver letto da tastiera un intero a
per effetto di un'istruzione read (a) contenuta in un programma PASCAL,
calcoli il valore della somma delle cifre di a.
Voglio dire:
se si comunicano al computer le cifre del numero UNA PER UNA, allora il problema è banale;
ma se invece si digita alla tastiera IL NUMERO a,
e il computer, eseguendo una read (a), lo acquisisce in memoria "tutto in una volta",
allora far calcolare dal computer la somma delle cifre di a è un po' più complicato.
Ci vuoi provare?

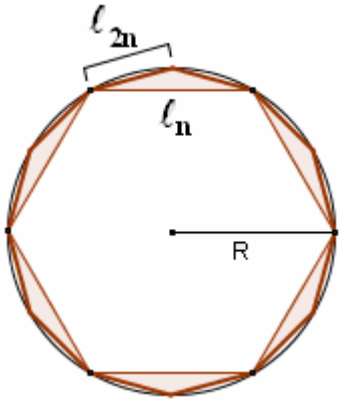
Esercizio 38')

**Esistono dei numeri naturali, non superiori a 10000, che siano uguali al
CUBO DELLA SOMMA DELLE PROPRIE CIFRE**?
In caso affermativo, quali e quanti sono?
Scrivi un programma PASCAL che possa fornire la risposta.

12 - APPROSSIMAZIONI DI PI GRECO

Se indichiamo con ℓ_n la misura del lato del poligono regolare di n lati, inscritto nella circonferenza di raggio R,
allora si dimostra che la misura del lato del poligono regolare, inscritto nella stessa circonferenza, ma avente numero di lati DOPPIO, è dato da

$$\ell_{2n} = \sqrt{2R^2 - R\sqrt{4R^2 - \ell_n^2}} \qquad \textbf{FORMULA } \ell_n \to \ell_{2n}$$



Poniamo ora per semplicità R=1; la formula diventa $\ell_{2n} = \sqrt{2 - \sqrt{4 - \ell_n^2}}$

Se partiamo dall'esagono regolare inscritto ($\ell_6 = 1$: è noto che il lato dell'esagono regolare inscritto è uguale al raggio della circonferenza), applicando ripetutamente questa formula potremo calcolare $\ell_{12}, \ell_{24}, \ell_{48}$ ...

Osserviamo ora che i perimetri dei corrispondenti poligoni regolari inscritti costituiscono approssimazioni per difetto, sempre più precise, della lunghezza della circonferenza.

Ma tale lunghezza è data da $2\pi \cdot raggio = 2\pi$ (ricordiamo che abbiamo preso R=1),
per cui i SEMIperimetri dei poligoni regolari considerati costituiranno approssimazioni per difetto, sempre più precise, del numero π.

Esercizio 39)

Scrivi un programma Pascal che, a partire dal lato dell'esagono regolare inscritto nella circonferenza di raggio R=1 ($\ell_6 = R = 1$), calcoli successivamente la misura del lato del poligono regolare inscritto avente: 12, 24, 48, 96, 192 … lati, e mandi in output tale misura insieme con la misura del SEMIperimetro del poligono corrispondente. Tali semiperimetri forniranno una successione di approssimazioni via via più precise, del numero π. Fai in modo che sia l'utente a stabilire il numero di iterazioni.

ALCUNE FAMOSE E BELLE FORMULE PER IL CALCOLO DEL VALORE DI π

- $\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \ldots = \frac{\pi}{4}$ (Leibniz)
- $\frac{1}{1\cdot 3} + \frac{1}{5\cdot 7} + \frac{1}{9\cdot 11} + \frac{1}{13\cdot 15} + \frac{1}{17\cdot 19} + \ldots = \frac{\pi}{8}$ (conseguenza della precedente)
- $\frac{2}{1}\cdot\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdot\frac{6}{7}\cdot\frac{8}{7}\cdot\frac{8}{9}\cdot\ldots = \frac{\pi}{2}$ (Wallis)
- $\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \ldots = \frac{\pi^2}{6}$ (Eulero)
- $\frac{1}{1\cdot 2\cdot 3} - \frac{1}{2\cdot 3\cdot 5} + \frac{1}{3\cdot 4\cdot 7} - \frac{1}{4\cdot 5\cdot 9} + \ldots = \pi - 3$ (Nilakantha)
- $\sqrt{12}\left(1 - \frac{1}{3\cdot 3} + \frac{1}{3^2\cdot 5} - \frac{1}{3^3\cdot 7} + \ldots\right) = \pi$ (Madhava)

In questa famiglia di formule, si hanno

- delle "somme infinite" (nel senso di: "somme di infiniti addendi")
- o dei "prodotti infiniti" (nel senso di: "prodotti di infiniti fattori")

e prendendo un certo numero di termini a partire da quello iniziale (es.: i primi 5, i primi 100 …) si ottiene un *valore approssimato* di π, con l'approssimazione che diventa via via più precisa quanto più si fa alto il numero dei termini considerati.

Esercizio 40) - Vedi NOTA qui a fianco

Scrivi un programma Pascal che fornisca una approssimazione di π tramite una a tua scelta delle formule sopra riportate. Fai in modo che sia l'utente a stabilire quanti termini utilizzare.

NOTA (il computer è COSTRETTO ad approssimare)

Un programma Pascal che calcoli il valore dei primi *n* termini in queste formule non potrà comunque, se *n* è molto alto, fornire il risultato *esatto* di quel calcolo, ma soltanto una sua *approssimazione*. La ragione è che una variabile numerica occupa in memoria un numero prefissato di byte (max 8-10) da cui tutta una serie di possibili errori di *overflow*, *underflow* (arrotondamento a 0 se il valore è troppo piccolo), "*cancellazione*".
C'è poi anche il fatto che il computer lavora in sistema binario, e nella conversione fra il decimale e il binario può essere costretto ad altre approssimazioni, dato il numero limitato di bit utilizzabili. Basti pensare che, ad esempio, il numero che in base dieci si scrive come 0.1 se viene portato in base due diventa periodico:

0.0001100110011001100110...

Questi problemi sono inerenti alla natura stessa del computer, quindi vanno valutati anche da chi programmi in un linguaggio diverso dal Pascal.
Vedi ⇨ per approfondimenti.

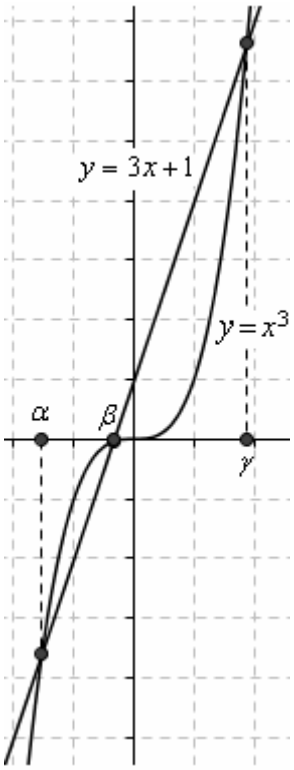
13 - APPROSSIMAZIONE DELLE SOLUZIONI DI UN'EQUAZIONE COL METODO DI BISEZIONE

Per descrivere questo bellissimo metodo, partiamo da un esempio.
Sia data l'equazione

$$\underbrace{x^3 - 3x - 1}_{f(x)} = 0$$

Innanzitutto, localizziamo approssimativamente le radici (NOTA)
col "metodo grafico".

*NOTA: quando si parla di un'equazione, la parola "**radici**" è sinonimo di "**soluzioni**"*



L'obiettivo è di "*separare*" le radici,
ossia di determinare, *per ciascuna* radice,
un *intervallo* che contenga *quella* radice e nessun'altra.
Per fare il grafico, sarà conveniente, nel nostro caso,
trasportare qualche termine a secondo membro,
in modo da aver a che fare con funzioni il più possibile facili da disegnare:

$$\underbrace{x^3}_{f_1(x)} = \underbrace{3x + 1}_{f_2(x)}$$

Vediamo così (figura qui a fianco) che l'equazione assegnata ha 3 radici:

$$\boxed{-2 < \alpha < -1}, \quad \boxed{-1 < \beta < 0}, \quad \boxed{1 < \gamma < 2}.$$

Consideriamo ora una radice, ad esempio γ, e l'intervallo in cui è stata "separata":

$$[a,\ b] = [1,\ 2].$$

Per la risoluzione grafica, abbiamo portato l'equazione sotto la forma

$$f_1(x) = f_2(x)$$

ma ora dobbiamo tornare a pensare alla forma iniziale

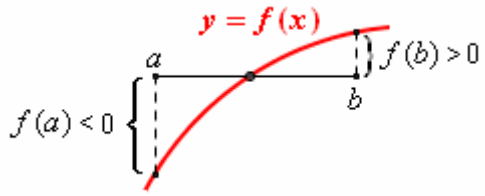
$$f(x) = 0 \quad \text{ossia} \quad f_1(x) - f_2(x) = 0$$

Della funzione $f(x) = x^3 - 3x - 1$ noi *non* abbiamo tracciato il grafico;
tuttavia, il fatto che le due funzioni $f_1(x),\ f_2(x)$
si sono "scavalcate" nell'intervallo $[a, b] = [1, 2]$
ci dice che la loro differenza $f_1(x) - f_2(x) = f(x)$
passa, al variare di x da $a = 1$ fino a $b = 2$,
dalla positività alla negatività o viceversa.

In effetti, se calcoliamo $f(a)$ e $f(b)$, troviamo valori discordi:

$$f(a) = f(1) = \left[x^3 - 3x - 1\right]_{x=1} = 1 - 3 - 1 = -3 < 0$$

$$f(b) = f(2) = \left[x^3 - 3x - 1\right]_{x=2} = 8 - 6 - 1 = 1 > 0$$



La situazione è perciò quella della figura qui a fianco →

E risolvere l'equazione $f(x) = 0$ equivale a chiedersi in quale ascissa avviene l'attraversamento dell'asse orizzontale, da parte del grafico della $f(x)$.

Cominciamo a chiederci se questo attraversamento dell'asse orizzontale avviene
nella *metà sinistra* dell'intervallo, oppure nella *metà destra*.

A questo scopo, troviamo il punto di mezzo dell'intervallo:

$$m = \frac{a+b}{2} = \frac{1+2}{2} = \frac{3}{2} = 1.5$$

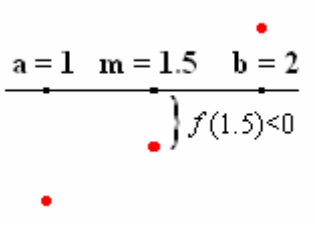
e calcoliamo $f(m)$ ossia $f(1.5)$.

Avremo $f(m) = f(1.5) = f\left(\frac{3}{2}\right) = \left(\frac{3}{2}\right)^3 - 3 \cdot \frac{3}{2} - 1 = \frac{27}{8} - \frac{9}{2} - 1 = \frac{27 - 36 - 8}{8} = -\frac{17}{8}$

Ora, essendo $-\frac{17}{8} < 0$,

la situazione sarà quella illustrata dalla figura qui a destra:
poiché $f(m)$ è **concorde** con $f(a)$,
il grafico della $f(x)$ attraverserà l'asse orizzontale
NON nell'intervallo $[a, m]$ bensì nell'ALTRO intervallo $[m, b]$.
Insomma, semplicemente confrontando il segno di $f(m)$ con quello di $f(a)$,
abbiamo stabilito che la soluzione cercata deve trovarsi nell'intervallo

$$[m, b] = [1.5; 2].$$



Schematizzando: **dal "vecchio" intervallo $[a, b]$ si passa al NUOVO intervallo**

$$\begin{cases} [a, m] \text{ se } f(m) \text{ è DISCORDE con } f(a), \text{ cioè se } f(m)\cdot f(a) < 0 \\ [m, b] \text{ se } f(m) \text{ è CONCORDE con } f(a), \text{ cioè se } f(m)\cdot f(a) > 0 \end{cases}$$

Ora iteriamo (= ripetiamo) il procedimento su questo nuovo intervallo …
… il nuovo intervallo prende il posto del vecchio!

Otterremo così, per **dimezzamenti successivi** (= bisezioni) dell'intervallo iniziale,
nuovi intervalli sempre più piccoli i cui estremi forniranno un'approssimazione per difetto e una per eccesso,
della soluzione cercata, via via sempre più precise.
Eccezionalmente (rarissimo), se dovesse capitare di trovare $f(m) = 0$,
ci imbatteremmo proprio nella soluzione esatta.

Esercizio 41)

Scriviamo un programma Pascal per risolvere, col metodo di bisezione, l'equazione P(x)=0, essendo P(x) un polinomio di grado non superiore a 5:

$$\mathbf{P(x) = c_0x^5 + c_1x^4 + c_2x^3 + c_3x^2 + c_4x + c_5}\ .$$

Innanzitutto stabiliremo, col metodo grafico, quante sono le soluzioni dell'equazione

$$c_0x^5 + c_1x^4 + c_2x^3 + c_3x^2 + c_4x + c_5 = 0$$

e le localizzeremo approssimativamente.

Dopodiché, individuato un intervallo [a, b] in cui siamo sicuri che cada una e una sola soluzione
della nostra equazione, affideremo ad un programma Pascal il compito di approssimare tale soluzione
con la precisione da noi desiderata (ad esempio, a meno di 0.0001), applicando, appunto,
il metodo di bisezione (= dimezzamenti successivi dell'intervallo in cui è localizzata la soluzione).

Il programma dovrà:

I) LEGGERE in input
- i 6 coefficienti c0, c1, c2, c3, c4, c5 del polinomio $P(x) = c_0x^5 + c_1x^4 + c_2x^3 + c_3x^2 + c_4x + c_5$
- gli estremi a, b dell'intervallo in cui l'utente ha "separato" una e una sola radice (= soluzione) dell'equazione $c_0x^5 + c_1x^4 + c_2x^3 + c_3x^2 + c_4x + c_5 = 0$
- e la precisione p con la quale l'utente desidera sia approssimata la soluzione in questione

II) CALCOLARE m = (a+b)/2 e poi SOSTITUIRE l'intervallo [a, b]
- ♪ con l'intervallo [a, m] se P(m) è discorde rispetto a P(a): P(m)*P(a)<0
- ♫ con l'intervallo [m, b] in caso contrario;

III) ITERARE il procedimento (calcolo di m = (a+b)/2 e sostituzione di [a, b] con [a, m] oppure [m, b]) **FINO A QUANDO**
- ci si imbatta nella soluzione esatta (caso rarissimo)
- OPPURE l'intervallo sia diventato tale che la sua ampiezza sia minore o uguale a p.

IV) Nel primo (eccezionale) caso, l'output dovrà essere

LA SOLUZIONE CERCATA E' …

mentre nel secondo caso dovrà essere

LA SOLUZIONE CERCATA E' COMPRESA FRA …

Beh, ho detto "scriviamo" … ma il programmatore sei tu.
Buon lavoro!!!

14 - LE BASI TEORICHE DELL'ALGORITMO DI EUCLIDE PER IL CALCOLO DEL M.C.D.

Siano a, b due interi, e sia d un loro divisore comune.
Dico che d è pure divisore del resto che si ottiene effettuando la divisione intera $a : b$.

Infatti:
la divisione intera $a : b$ produce
un "quoziente intero" q e un resto r $(r < b)$,
legati dalla relazione $a = q \cdot b + r$.
Ma se d è un divisore comune per a e b,
allora d è contenuto
un "numero esatto" di volte sia in a che in b,
e ciò implica che sia contenuto
un numero esatto di volte pure in r, perché

$$a = md \quad (m \text{ intero})$$
$$b = nd \quad (n \text{ intero})$$
$$\text{IMPLICA}$$
$$r = a - q \cdot b = md - qnd = \underbrace{(m - qn)}_{\text{intero}} d$$

Per capire meglio, vediamo un esempio specifico.
$a = 186,\ b = 24,\ d = 6.$
6 *è divisore comune per* 186 *e* 24.
Mi aspetto dunque che 6 *sia pure divisore del resto che si ottiene facendo la divisione intera* $186 : 24$. *Vediamo.*
$186 : 24 = 7$ *col resto di* $\boxed{18}$ $(186 = 7 \cdot 24 + 18)$.
In effetti, 6 *è divisore di* 18!
E d'altronde: dato che 6 *è contenuto*
– un numero esatto di volte nel 186 (31 *volte*),
– e un numero esatto di volte nel 24 (4 *volte*)
PER FORZA 6 *doveva essere contenuto un numero esatto di volte nel resto della divisione* $186 : 24$, *come spiega la catena*

$$18 = \underbrace{186}_{31 \cdot 6} - 7 \cdot \underbrace{24}_{4 \cdot 6} = 31 \cdot 6 - 28 \cdot 6 = \underbrace{3}_{\text{intero}} \cdot 6$$

Siano a, b due numeri interi, e sia d un intero, che risulti divisore comune a uno di questi due numeri (ad esempio, b) e al resto della divisione intera $a : b$.
Dico che d è pure divisore dell'altro numero.

La giustificazione generale
è analoga alla precedente.

Anche qui, un esempio gioverà alla comprensione.
$a = 68,\ b = 20$
$68 : 20 = 3$ *col resto* $r = 8$ $(68 = 3 \cdot 20 + 8)$
$d = 4$ *è divisore comune per* b *ed* r:
è contenuto un numero esatto di volte in $b = 20$ (5 *volte*),
e un numero esatto di volte in $r = 8$ (2 *volte*)

$$68 = 3 \cdot \underbrace{20}_{5 \cdot 4} + \underbrace{8}_{2 \cdot 4} = 15 \cdot 4 + 2 \cdot 4 = \underbrace{17}_{\text{intero}} \cdot 4$$

$d = 4$ *è contenuto un numero esatto di volte in* $a = 68$

Ricapitoliamo.
Ci sono tre interi in gioco, che sono a, b, ed r (*resto di* $a : b$).
Abbiamo scoperto che se d è divisore di DUE fra i tre interi considerati, allora sarà necessariamente divisore anche del terzo.

Dal discorso fatto, ci interessa trarre quanto segue.

Detti a, b due interi, e detto r il resto della divisione intera $a : b$, allora:
- **se d è divisore comune per a, b, allora d è pure divisore comune per b, r**
- **se d è divisore comune per b, r, allora d è pure divisore comune per a, b**

per cui

l'insieme dei divisori comuni della coppia a, b
coincide con l'insieme dei divisori comuni della coppia b, r
quindi si ha pure
$$\text{M.C.D.}(a, b) = \text{M.C.D.}(b, r)$$

Il vantaggio di tutto ciò sta nel fatto che
il calcolo di $\text{M.C.D.}(a, b)$
può essere sostituito da quello di $\text{M.C.D.}(b, r)$
che è più facile perché più piccoli sono i numeri in gioco.

E' sulla base di queste considerazioni che "funziona"
l' ALGORITMO DI EUCLIDE per la ricerca del M.C.D.
di cui ci siamo occupati alle pagine precedenti.