L'ITERAZIONE (I CICLI)

Iterare significa "ripetere", quindi "iterazione" significa ripetizione. In informatica la struttura di controllo iterativa è detta quindi anche "ciclo" poiché prevede una sequenza di istruzioni che si ripetono.

Cosa sono i cicli? Sono una blocco di istruzioni che viene eseguito dall'esecutore fintanto che una condizione è vera. Quando la condizione è falsa il blocco non viene più eseguito. Non confondiamolo con la selezione (if).

Il ciclo "if" non esiste, "if" non è un ciclo.

Perché servono i cicli? Perché solo con con la sequenza e la selezione alcune cose non si riescono a fare.

Esempio di una cosa che si può fare sia con ciclo sia senza ciclo:

- Algoritmo che chiede di calcolare la media fra tre voti

Esempio di una cosa che si può fare solo con ciclo:

- Algoritmo che chiede di sommare tutti i voti inseriti dall'utente fino a quando l'utente inserisce un numero negativo (valore sentinella)

```
inizio
        somma=0
        esegui
                leggi n
                somma=somma+n
        mentre (n!=0)
        scrivi somma
fine
```

Senza ciclo non puoi farlo perché non sai quante volte far leggere n. 3 volte? 4 volte? non puoi saperlo!

Nel linguaggio C esistono tre tipologie di cicli: il ciclo precondizionato (ciclo while) il ciclo post-condizionato (ciclo do – while) e il ciclo enumerativo (ciclo for)

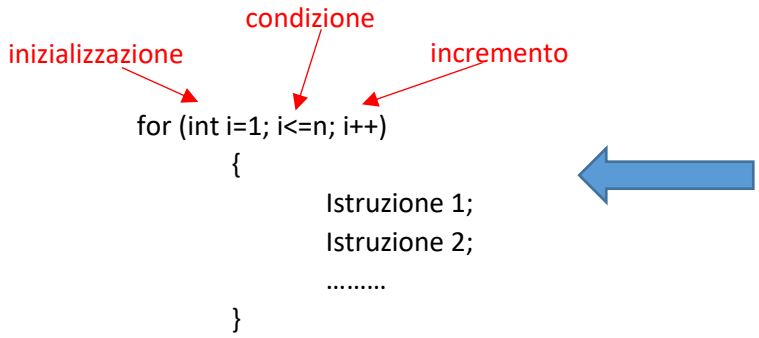
1. Iterazione indeterminata con controllo in testa (ciclo while precondizionato):

```
while (condizione)
        {
                Istruzione 1;
                Istruzione 2;
                ……
        }
```

2. Iterazione indeterminata con controllo in coda (ciclo do while postcondizionato):

```
do
        {
                Istruzione 1;
                Istruzione 2;
                ……
        } while (condizione) ;
```

3. Iterazione determinata (ciclo for – iterazione enumerativa):



Ad ogni ripetizione i viene incrementato di 1. Fintanto che la condizione i<=n è vera, l'iterazione viene eseguita, quando la condizione non è più vera, l'esecutore esce dal ciclo. Si può incrementare i di valori diversi da 1. Esempio: for (float i=0;i<=1;i+=0.1).

Il ciclo for è particolare in C, rispetto agli altri linguaggi, perché **la condizione può esser espressa come una condizione qualsiasi** che non ha nulla a che fare con il contatore dei cicli.

Generazione di numeri casuali in C.

NOTA: per la generazione di numeri casuali è necessario richiamare le librerie:

```
#include <stdlib.h>  //per utilizzare srand e rand
#include <time.h>    //per utilizzare la funzione time
```

Ed utilizzare le funzioni:

time (NULL)	**// restituisce il secondi trascorsi dalla mezzanotte del 1 gennaio 1970**
srand(time(NULL))	**//invocata una volta sola per inizializzare il generatore di numeri casuali**
rand()	**//invocata ogni volta che si vuole generare un numero casuale**
	genera un numero casuale fra 0 e 32767 (questo valore dipende dal compilatore, può variare)

Spiegazione:

```
int main()
{
        int x, valoreDado;
        x=rand();
        valoreDado =(x%6)+1;
        printf("%i", valoreDado);
        return 0;
}
```

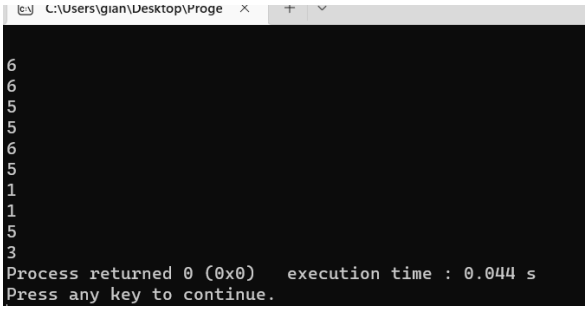
La funzione rand genera un numero pseudocasuale (x) compreso fra 0 e RAND_MAX, costante dell'ANSI C che vale 32767

Del numero casuale generato viene preso il resto della divisione per 6 (che sarà un numero compreso fra 0 e 5) e gli viene sommato 1. In questo modo si ottiene sempre un numero casuale (valoreDado) compreso fra 1 e 6

Eseguendo il codice precedente per 10 volte (con ciclo for ad esempio) si ottengono 10 numeri casuali (x) compresi fra 1 e 6. Ecco il codice:

```
int main()
{
   int x;
   for(int i=0;i<10;i++)
   {
      x=rand();
      x=(x%6)+1;
      printf("\n%i",x);
   }
   return 0;
}
```

Il problema è che ripetendo l'esecuzione del codice precedente si ottengono sì 10 numeri casuali, ma ogni volta gli stessi 10 numeri casuali



Questo accade perché la funzione **rand()** genera i numeri in maniera non proprio casuale ma pseudo casuale, ossia applicando un algoritmo ad un numero di partenza chiamato seme (seed). Ogni volta che viene eseguito il codice dall'inizio il seed è sempre lo stesso quindi la sequenza di numeri casuali è sempre la stessa.

Per generare, ad ogni esecuzione, sequenze casuali è necessario che ad ogni esecuzione venga cambiato il seed. Per fare questo si utilizza la funzione **srand()** che accetta come argomento un numero intero che rappresenta il seed. Affinchè ogni volta che viene eseguito il programma, il seed sia diverso dalla volta precedente si utilizza come seed un numero che rappresenta il numero di secondi trascorsi dalla mezzanotte del 1 gennaio 1970 (anno della nascita del SO UNIX, istante chiamato **EPOCH**), che sarà dunque un numero sempre diverso ogni secondo.

Per ottenere questo numero di secondi si utilizza la funzione time() della libreria <time.h> con argomento il valore NULL. Quindi

time(NULL) = numero di secondi trascorsi dal 1 gennaio 1970

quindi il seguente codice genera una sequenza di 10 numeri casuali compresi fra 1 e 6, diversa per ogni esecuzione.

```
int main()
{
   int x;
   srand(time(NULL));
   for(int i=0;i<10;i++)
   {
      x=rand();
      x=(x%6)+1;
      printf("\n%i",x);
   }
   return 0;
}
```

srand() inizializza il seed ad ogni nuova esecuzione del programma.

Il vaore del seed è, ad ogni esecuzione, diverso perché è dato dal numero di secondi trascorsi da **epoch.**

il codice per generare (e visualizzare) un numero casuale compreso fra 1 e 6, sempre diverso, è dunque il seguente:

```
int main()
{
   int x;
   srand(time(NULL));
   x=rand();
   x=(x%6)+1;
   printf("\n%i",x)
   return 0;
}
```

srand() inizializza il seed ad ogni nuova esecuzione del programma.

Il vaore del seed è, ad ogni esecuzione, diverso perché è dato dal numero di secondi trascorsi da **epoch.**

Se si volesse generare un numero casuale compreso fra 1 e un qualsiasi altro numero intero MAX (minore di 32767) basta sostituire il valore 6 con il valore MAX

Esempio che genera un numero casuale compreso fra 1 e 100:

```
int main()
        {
                int x;
                srand(time(NULL));
                x=rand();
                x =(x%100)+1;
                printf("%i", x);
                return 0;
        }
```

LE ISTRUZIONI BREAK E CONTINUE NEI CICLI

Sono istruzioni che vengono inserite all'interno di un ciclo per "gestire" casi particolari, infatti si utilizzano generalmente in una selezione interna al ciclo che "seleziona" il caso particolare.

Break causa la fuoriuscita dal ciclo, l'esecuzione riprende dalla prima istruzione successiva al ciclo.

Continue riprende dalla prima istruzione del ciclo (non arriva al termine di quella iterazione, riparte dalla successiva).

Esempio p. 115, facciamolo insieme insieme (break e continue): realizzare un programma che acquisisce numeri interi e mostra la somma dei numeri interi acquisiti applicando però le seguenti regole:

- Se il numero inserito è superiore a 100 viene mostrato un messaggio di errore ed il numero NON viene conteggiato nella somma.
- Se il numero inserito è <0 viene mostrato un messaggio di errore e termina l'acquisizione.
- Ogni volta che viene acquisito un numero compreso fra 1 e 100, viene mostrato il numero acquisito e viene incrementata la somma.
- Il programma termina inserendo un valore =0 oppure quando si verifica un errore (valore inserito <0), al termine del programma viene mostrata la somma ed il numero di numeri validi inseriti (escluso lo 0).

```
4   int main()
5   {
6       int n,contatore=0,somma=0;
7       do
8       {
9           printf("\nInserisci il valore n positivo compreso fra 1 e 100: !\n0 terminera' l'inserimento\n");
10          scanf("%i",&n);
11          if (n>100)
12          {
13              printf("\nErrore! Valore >100, non valido!");
14              continue;
15          }
16          else if(n<0)
17          {
18              printf("\nErrore! Il valore inserito è <0. L'acquisizione verra' terminata");
19              break;
20          }
21          else if (n==0)
22              break;
23          printf("Valore acquisito %i. Valore valido ",n);
24          somma=somma+n;
25          contatore++;
26      } while(1);
27      printf("\n numero di valori inseriti: %i ",contatore);
28      printf("\n somma: %i ",somma);
29      return 0;
30  }
```

Ciclo infinito dal quale si esce con break.

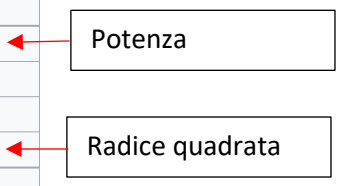
Continue fa ripartire l'iterazione

LA LIBRERIA MATEMATICA E LE ALTRE LIBRERIE IN C

Per utilizzare particolari funzioni matematiche (ad esempio radice quadrata, potenza, …) è necessario includere la libreria matematica <math.h>.

Nella seguente tabella si riportano le funzioni presenti nella libreria math.h

Membro	Descrizione
`acos`	arcocoseno
`asin`	arcoseno
`atan`	arcotangente
`atan2`	arcotangente di due parametri
`ceil`	il più piccolo intero non minore del parametro
`cos`	coseno
`cosh`	coseno iperbolico
`exp(double x)`	funzione esponenziale, calcola e^x
`fabs`	valore assoluto
`floor`	il più grande intero non maggiore del parametro
`fmod`	resto del numero in virgola mobile
`frexp`	frazione e potenza di due.
`ldexp`	operazione in virgola mobile
`log`	logaritmo naturale
`log10`	logaritmo in base 10
`pow(x,y)`	eleva un valore dato ad esponente, x^y
`sin`	seno
`sinh`	seno iperbolico
`sqrt`	radice quadrata
`tan`	tangente
`tanh`	tangente iperbolica



Le principali librerie del linguaggio C che utilizzeremo sono contenute nei seguenti header file. Ricordiamoci di includere, all'occorrenza, nei progetti.

<stdlib.h>: libreria standard, contiene funzioni di uso comune per allocazione della memoria, funzioni di ricerca e ordinamento (vedremo quando ci serviranno)

<stdio.h>: Funzioni per la gestione degli I/O standard (tastiera, monitor) e su file (ad esempio printf e scanf)

<string.h>: funzioni per la gestione delle stringhe di caratteri

<math.h>: funzioni matematiche

<time.h> funzioni per la gestione delle date e degli orari

<stdbool.h> per utilizzare variabili booleane (true e false)

Link a tutte le librerie standard del C: https://en.cppreference.com/w/c/header

Esercizio sulla libreria matematica: distanza fra due punti (p121): scrivere il programma che, date come input le coordinate di due punti sul piano cartesiano (xa, ya, xb,yb) fornisce la distanza fra i due punti utilizzando la formula: d= radice $((xa-xb)^2+(ya-yb)^2)$

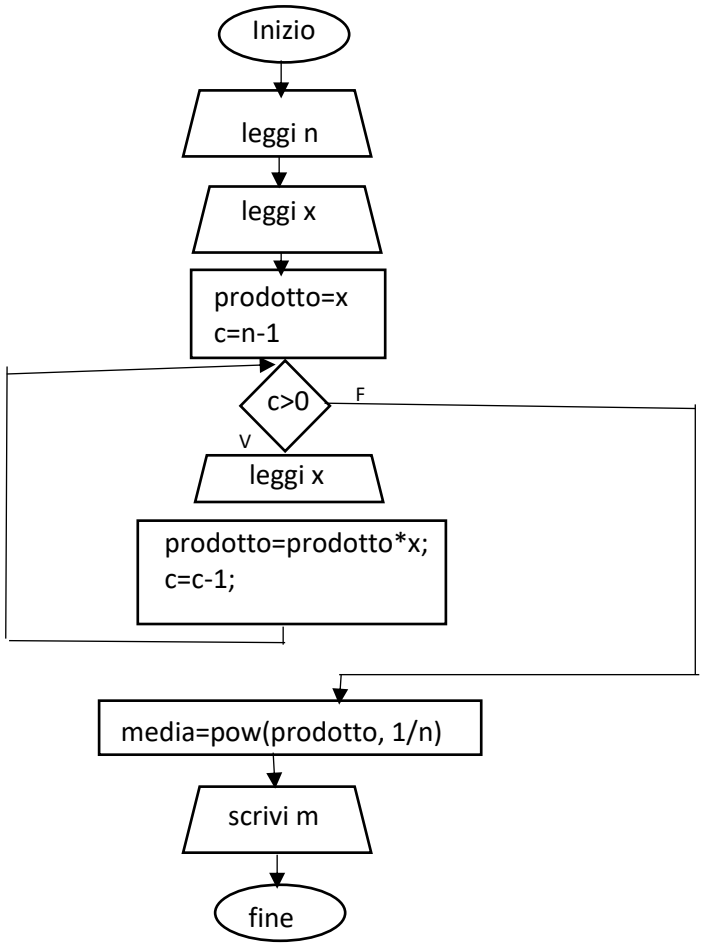
ESERCIZIO: uso del debugger (Media Geometrica)

Dati n numeri, la media geometrica (m) di questi numeri si calcola con la seguente formula:

$$m=(x1*x2*\ldots*xn)^{1/n}$$

Scrivere un programma che calcoli la media geometrica di n numeri inseriti dall'utente.

Soluzione (algoritmo):

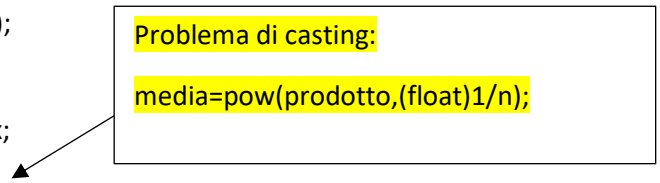


Codice sintatticamente corretto ma che contiene un errore da individuare con debug:

provare a calcolare con la calcolatrice quale dovrebbe essere il risultato corretto (ad esempio con 3 numeri 10,20,30) e poi verificare che il risultato fornito dal codice non è corretto. Eseguire passo passo il codice ed individuare l'errore e correggerlo

```
int main()
{
   int n,x,c,prodotto;
```

```
   float media;
   printf("Inserisci n (n>1)\n");
   scanf("%i",&n);
   printf("Inserisci x \n");
   scanf("%i",&x);
   c=n-1;  //contatore
   prodotto=x;
   while(c>0)
   {
      printf("Inserisci x \n");
      scanf("%i",&x);
      c=c-1;
      prodotto=prodotto*x;
   }
   media=pow(prodotto,1/n);
   printf("\nMedia geometrica=%f",media);
   return 0;
}
```



Esercizio importante per uso del debugger: scrivere un programma che mostri la posizione di un corpo che cade in caduta libera (in assenza di attrito). Nell'istante iniziale (t=0) il corpo si trova nella posizione 0. Il programma deve mostrare, ogni 0.1 s, la posizione del corpo per un tempo che va da 0 ad 1 s. La formula del moto della caduta di un corpo è

$$posizione=(1/2)*g*t^2$$

dove g=9.81 m/s^2 è l'accelerazione di gravità (costante).

ESERCIZI RIASSUNTIVI SU CICLI E NUMERI CASUALI

Esercizio 1 (gioco dei dadi)

Si realizzi un programma che simuli il seguente gioco dei dadi:

1. **Viene chiesto al giocatore un budget iniziale in denaro (compreso fra 1 e 1000 Euro, controllare l'input)**
2. **A questo punto il giocatore può iniziare a giocare.**
3. **Per ogni giocata viene chiesto al giocatore una "puntata" in denaro che deve essere inferiore o uguale al budget disponibile.**
4. **Il giocatore "lancia" due dadi (ossia vengono generati due numeri casuali fra 1 e 6)**
5. **Il giocatore vince quando la somma dei due dadi è 7 oppure quando è uscito lo stesso numero su entrambi i dadi.**
6. **Se il giocatore vince, la "puntata" effettuata viene sommata al budget, se perde viene sottratta al budget.**
7. **Dopo ogni giocata viene mostrato il budget e il giocatore può decidere se ritirarsi oppure effettuare un'altra giocata.**
8. **La partita termina quando il giocatore decide di ritirarsi oppure quando il giocatore ha terminato i soldi (budget=0)**

Esercizio 2 (Distributore sigarette con resto)

Scrivere un programma per un distributore di sigarette.

Il programma deve acquisire l'età dell'utente ed abilitarlo all'acquisto solo se l'età è >=18.

Il programma deve acquisire il denaro inserito da parte dell'utente.

Una volta inserito il denaro, il programma deve mostrare all'utente il tipo di sigarette ed il relativo costo affinchè l'utente possa selezionare il prodotto desiderato (deve esserci anche la possibilità per l'utente di non selezionare alcun prodotto e recuperare il denaro). Si prevedano almeno 5 tipologie di sigarette di costo diverso.

Il distributore deve erogare il prodotto ed il resto (si simulino tali attività con delle comunicazioni sul monitor)

Se il denaro introdotto non è sufficiente il distributore deve chiedere l'inserimento di altro denaro.

Eventuale miglioramento: verificare se l'utente è maggiorenne facendo inserire all'utente la data di nascita anziché l'età.

Esercizio 3(Estrazione del lotto con calcolo vincita)

Programma "ESTRAZIONI_LOTTO"

Realizzare un programma che generi 5 numeri casuali compresi fra 1 e 90 come se fossero le estrazioni del lotto su di una ruota e comunichi quanti numeri sono stati azzeccati da un utente che gioca una "schedina".

Il programma deve inizialmente chiedere all'utente quanti numeri intende giocare sulla schedina (minimo 1 numero, massimo 10 numeri), I numeri devono essere compresi fra 1 e 90. Non deve essere possibile inserire due volte lo stesso numero per una puntata (i valori verranno assegnati alle variabili Puntata1, Puntata2,….., Puntata10).

Il programma chiede all'utente quanto denaro intende puntare sull'ambata (significa 1 numero azzeccato), quanto sull'ambo (la richiesta va effettuata solo se l'utente ha giocato almeno due numeri), quanto sul terno (la richiesta va effettuata solo se l'utente ha giocato almeno 3 numeri), quanto sulla quaterna (la richiesta va effettuata solo se l'utente ha giocato almeno 4 numeri), quanto sulla cinquina (la richiesta va effettuata solo se l'utente ha giocato almeno 5 numeri).

Il programma dovrà generare 5 numeri casuali compresi fra 1 e 90 (i valori verranno assegnati alle variabili Estratto1, Estratto2,…, Estratto5).

I 5 valori estratti dovranno essere diversi fra loro.

Il programma confronterà i numeri puntati con i 5 numeri estratti e poi comunicherà a quanto ammonta la vincita in euro in base alla seguente tabella. La tabella indica il moltiplicatore di vincita, ossia in caso di vittoria viene moltiplicata la puntata per il moltiplicatore di vincita:

N. giocati	Ambata	Ambo	Terno	Quaterna	Cinquina
1	11,23	-	-	-	-
2	5,61	250	-	-	-
3	3,74	83,33	4.500	-	-
4	2,80	41,66	1.125	120.000	-
5	2,24	25	450	24.000	6.000.000
6	1,87	16,66	225	8.000	1.000.000
7	1,60	11,90	128,57	3.428,57	285.714,28
8	1,40	8,92	80,35	1.714,28	107.142,85
9	1,24	6,94	53,57	952,38	47.619,04
10	1,12	5,55	37,50	571,42	23.809,52

Alla fine il programma consentirà di scegliere se effettuare una nuova estrazione (con una nuova schedina) o terminare l'esecuzione.

La vincita va arrotondata all'euro più vicino. Per arrotondare un numero float all'intero più vicino si usa la funzione round() della libreria math.h

Esempio x=round(5.3) → x vale 5

Esempio x=round(5.7) → x vale 6

Esempio 1:

- **giocati 2 numeri puntando 1€ sull'ambo:**
  azzeccati 2 numeri

  vincita per l'ambo = 250*1 = 250 €

Esempio 2:

- **giocati 2 numeri puntando 1€ sull'ambo e 1 € sull'ambata:**
  azzeccati 2 numeri

  vincita per l'ambo = 250*1 = 250 € +

  vincita per le due ambate = 5,61*1€ *2 = 11,22 €

  Totale: 261 € (la vincita viene arrotondata all'intero più prossimo)

Esempio 3:

- **giocati 3 numeri puntando 1€ sull'ambo e 1 € sul terno**
  azzeccati 2 numeri

  vincita per l'ambo = 83,33*1€ = 83 € (la vincita viene arrotondata all'intero più prossimo)

Esempio 4:

- **giocati 3 numeri puntando 1€ sull'ambo e 1 € sul terno**
  azzeccati 3 numeri

  vincita per il terno = 4500*1€ = 4500 € +

  vincita per i 3 ambi = 83,33*3*1€=249,99 €

  Totale 4750 € (la vincita viene arrotondata all'intero più prossimo)

La formula per calcolare il numero di combinazioni di "n" numeri a gruppi di "k" è il coefficiente binomiale:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

(il punto esclamativo indica il fattoriale) dove n sono i numeri azzeccati e k vale 2 per l'ambo, 3 per il terno, 4 per la quaterna.

Infatti nell'ultimo esempio, oltre al terno sono stati azzeccati anche 3 ambi perché le combinazioni di 3 numeri (n=3) a gruppi di 2 (ambo, quindi k=2) sono: 3! / (2! * 1!) = 3

Al seguente link vi è un calcolatore del coefficiente binomiale che può essere utile: https://miniwebtool.com/it/binomial-coefficient-calculator/?n=10&k=2

Esempio 5:

- **giocati 4 numeri puntando 1€ sull'ambo, 1 € sul terno e 1 € sulla quaterna,**
  azzeccati 4 numeri)

le vincite sono:

- **quaterne: 1**
- **terni: 4! / 3! *1! = 4**
- **ambi: 4! / (2! * 2!) = 6**

vincita per la quaterna= 120000*1€ = 120000 € **+**

vincita per i 4 terni (con 4 numeri) = 1125*1€ *4= 4500 € **+**

vincita per i 6 ambi = 41.66 *6*1€=249.96 €

Totale 12750 € (la vincita viene arrotondata all'intero più prossimo)

Esempio 6:

- **giocati 7 numeri puntando 1 € sull'ambata, 1€ sull'ambo, 1 € sul terno e 1 € sulla quaterna, 1 € sulla cinquina**
  azzeccati 4 numeri)

le vincite sono:

- **quaterne: 1**
- **terni: 4! / (3! *1!) = 4**
- **ambi: 4! / (2! * 2!) = 6**
- **ambate: 4 (sono i 4 numeri azzeccati)**

vincita per la quaterna (con 7 numeri) = 3428,57*1€ = 3428,57 € **+**

vincita per i 4 terni (con 7 numeri) = 128,67*4*1€ = 514.28 € **+**

vincita per i 6 ambi (con 7 numeri) =11,9*6*1€=71,4 €

vincita per le 4 ambate (con 7 numeri) =1,6*4*1€=6.4 €

Totale 4021 € (la vincita viene arrotondata all'intero più prossimo)

Puoi verificare la correttezza del calcolo delle vincite su questo sito (Verificare la vincita lorda): https://camporotondo.altervista.org/mioweb3/calcolo_vincite_lotto_.htm

I FILE E IL LORO UTILIZZO PER L'INPUT DEI DATI

Perché servono i file?

Abbiamo visto che durante l'esecuzione di programmi in C è sempre necessario inserire dati. Per evitare di dover inserire i dati all'avvio ogni esecuzione e di perdere i risultati al termine di ogni esecuzione, sarebbe utile memorizzarli su una memoria di massa. Per questo si usano i file. I file consentono di rendere **persistenti** i dati e i risultati, quindi di averli sempre a disposizione.

1. CLASSIFICAZIONE DEI FILE

I file si possono classificare in base a diverse caratteristiche.

1. **Classificazione In base alla codifica utilizzata** per memorizzare i dati, i file si distinguono in due tipologie: **file binari e file di testo**.

   I **file di testo** memorizzano i dati come una sequenza di caratteri, quindi per ogni carattere viene memorizzata la sua codifica ASCII (o altra codifica)

   I **file binari** memorizzano i dati con una sequenza di byte che sono interpretabili solamente dalle applicazioni con i quali sono stati creati. Nei file binari vengono salvati dei dati strutturati, quindi ogni dato salvato corrisponde ad un esemplare della struttura memorizzata. Ad esempio in un videogame di automobili verranno memorizzati in un file binario tutte le automobili del videogioco, quindi ogni esemplare di dato memorizzato, che viene chiamato record, rappresenta un personaggio, ad esempio i primi 7 byte rappresentano la targa, i successivi 100 byte la marca dell'automobile ecc.

   Quindi in un file binario vengono memorizzati dei record. Ad esempio in un file automobili.bin ogni record rappresenta un'automobile, in un file studenti.bin ogni record rappresenta in binario uno studente (ad esempio con matricola, cognome, nome) e così via..

   Se memorizziamo il valore 1 su un file binario esso verrà memorizzato in un byte della memoria di massa con il suo valore numerico 00000001, se memorizziamo 1 come file di testo esso sarà memorizzato nella memoria di massa come 00110001 (codice ASCII del simbolo 1, corrispondente al numero 49 in base dieci).

   Naturalmente se il carattere "1" è memorizzato in un file di testo, il suo valore in binario non è 1 quindi non è possibile utilizzarlo per svolgere delle operazioni aritmetiche nell'applicazione.

2. **Classificazione In base in base alla organizzazione dei dati**, ossia a come le applicazioni possono accedere ai dati: i file possono essere: in file **sequenziali** o ad **accesso diretto**.
   I file sequenziali consentono alle applicazioni di accedere ai dati **solo in modo sequenziale**, ossia partendo dal primo dato memorizzato, passando al secondo e così via,
   I file ad accesso diretto (o random) consentono alle applicazioni di accedere ad un dato in qualsiasi posizione, ad esempio direttamente al dato in 98esima posizione.

I file che gestiremo ora in questo modulo sono file **di tipo testuale ad accesso sequenziale**. I file testuali sono sempre ad accesso sequenziale. Sono quelli la cui gestione è la più semplice (per questo impariamo inizialmente a gestire questi), la loro gestione è spesso poco efficiente in termini di tempo di accesso, ma comunque, oltre al vantaggio di essere facilmente gestibili, sono ancora oggi spesso utilizzati per lo scambio di informazioni fra applicazioni diverse, quindi è utile imparare a gestirli.

2. OPERAZIONI CHE UN'APPLICAZIONE PUO' SVOLGERE SUI FILE

Un'applicazione può svolgere le seguenti operazioni su un file:

- **apertura** → nella memoria centrale vien riservato "uno spazio" in cui memorizzare il file
- **lettura** → i dati del file vengono copiati dalla memoria di massa alla memoria centrale affinché l'applicazione li possa utilizzare e modificare.
- **scrittura** → i dati del file vengono copiati dalla memoria di centrale alla memoria di massa per essere resi persistenti
- **chiusura** → la memoria centrale viene "liberata" dallo spazio utilizzato per ospitare i dati del file

Prima di svolgere una lettura o una scrittura un file va sempre aperto.

Dopo aver svolto un'operazione di lettura o scrittura un file va sempre chiuso.

3. UTILIZZO DEI FILE IN LINGUAGGIO C

Per l'accesso ai file introduciamo un tipo di variabile particolare che si chiama "**puntatore a file**", tale variabile va dichiarata premettendo al suo identificatore l'asterisco e per ora ci basta sapere che **contiene l'indirizzo della prima cella di memoria centrale in cui si trova il file.**

Per eseguire sui file di testo le operazioni menzionate, la libreria standard del linguaggio C (<stdio.h>) mette a disposizione le seguenti funzioni:

apertura → funzione **fopen (puntatore a file, modalità di apertura): in caso di impossibilità di aprire il file restituisce il valore NULL, ad esempio se si cerca di aprire in lettura un file con non esiste**

TABELLA **2**

Modalità	Descrizione
`"r"`	Apertura del file in lettura (*read*). Genera un errore se il file è inesistente.
`"w"`	Apertura del file in scrittura (*write*). Se il file non esiste viene creato, se esiste viene sovrascritto.
`"a"`	Apertura del file in estensione (*append*). Se il file non esiste viene creato, se esiste viene esteso.

lettura → funzione **fscanf (puntatore a file, specificatore di formato, variabile). Restituisce un intero con il numero di dati letti.**

Vediamo come si usa questa funzione per leggere da un file di testo una sequenza di numeri interi:

numeroDatiLetti=fscanf(file,"%i",&numeroLetto);

1. **Se lettura OK →, viene posto il numero letto in numeroLetto, viene restituito 1, viene incrementato il puntatore ai dati**
2. **Se lettura non OK perché il dato letto non è un numero intero → viene restituito 0 (nessun dato letto), il puntatore ai dati non viene incrementato**
3. **Se il file è finito oppure si verifica un errore di lettura (ad esempio file fisicamente non raggiungibile, cambio permessi di accesso al file) → viene restituito -1;**

Per verificare se il file è finito o se si è verificato un errore, si possono utilizzare le funzioni feof() oppure ferror().

feof() vale 1 se è stata raggiunta la fine del file, 0 altrimenti

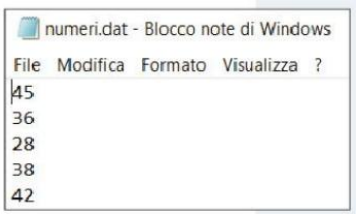
ferror() vale 1 se ci sono errori, 0 se si è verificato l'errore di I/O

In C è esiste una costante predefinita chiamata EOF (End Of File) che vale -1 che e' possibile utilzzare per verificare se il file è terminato o no (if (fscanf(….)==EOF)

chiusura → funzione **fclose (puntatore e file)**

Esempio:

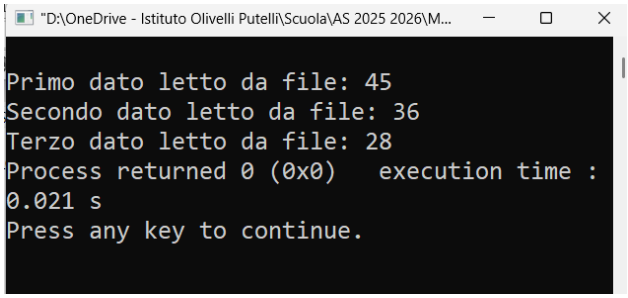
Supponiamo che un file di testo "dati.dat" contenga dei dati numerici:



Per leggere i primi tre numeri (45,36,28) dal file e stamparli sul monitor si potrebbe scrivere il seguente codice:

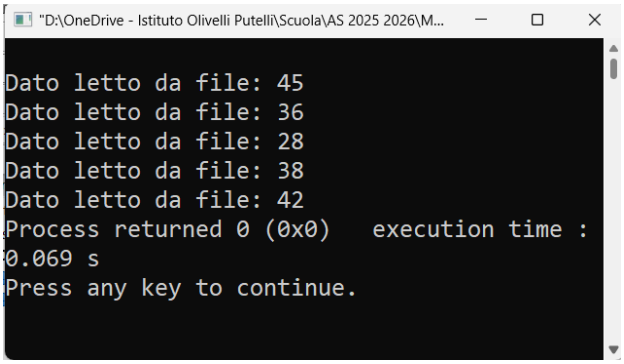
```
int main()
{
   FILE *file1; //variabile puntatore a file
   int num;

   file1=fopen("dati.dat","r");
   if (file1==NULL)
      printf("Impossibile aprire il file");
   else
   {
      fscanf(file1,"%i",&num);
      printf("\nPrimo dato letto da file: %i",num);
      fscanf(file1,"%i",&num);
      printf("\nSecondo dato letto da file: %i",num);
      fscanf(file1,"%i",&num);
      printf("\nTerzo dato letto da file: %i",num);
      fclose(file1);
   }
   return 0;
}
```



Per poter leggere (e stampare a video) uno alla volta tutti i dati presenti nel file senza sapere quanti sono questi dati si realizza un ciclo che legge il dato fintanto la funzione fscanf() che, ricordo, restituisce il numero di dati letti, restituisce il valore 1. Il codice è il seguente:

```
int main()
{
   FILE *file1;                    //variabile puntatore a file
   int num;
   file1=fopen("dati.dat","r");
   if (file1==NULL)
      printf("Impossibile aprire il file");
   else
   {
      while(fscanf(file1,"%i",&num)==1)          //finchè viene letto un dato
      {
         printf("\nDato letto da file: %i",num);
      }
      fclose(file1);
   }
   return 0;
}
```



Come si può vedere, l'esecutore esce dal ciclo di lettura quando la funzione fscanf() restituisce un valore diverso da 1. Questo può accadere per tre motivi:

1. Il dato letto non è un numero intero, in tal caso fscanf() restituisce il valore 0;
2. Il file è terminato, in tal caso la funzione fscanf() restituisce -1 e la funzione **feof(file1)** restituisce true.
3. Si è verificato un errore di I/O, in tal caso la funzione **ferr(file1)** restituisce true.

per verificare se tutti i dati presenti sono stati letti oppure il ciclo è terminato a causa della presenza di dati non validi (per esempio stringhe), si possono aggiungere al codice le seguenti istruzioni in rosso:

```
int main()
{
   FILE *file1;                    //variabile puntatore a file
   int num;
   file1=fopen("dati.dat","r");
   if (file1==NULL)
      printf("Impossibile aprire il file");
   else
   {
            while(fscanf(file1,"%i",&num)==1)          //finchè viene letto un dato
            {
                  printf("\nDato letto da file: %i",num);
            }
    if (fscanf(file1,"%i",&num)==0)
printf("\nErrore! Sono presenti dati non numerici nel file!");
     else if (fscanf(file1,"%i",&num)==EOF)
            printf(\nTutti i dati sono stati letti correttamente);
            fclose(file1);
   }
   return 0;
}
```

//significa: è stato letto un dato non corrispondente allo specificatore di formato indicato

//il file è terminato

//per ora trascuriamo il controllo sul caso in cui i verifichi un errore di I/O

Verificare il corretto funzionamento del codice sia con un file che contiene i dati corretti (tutti numerici) sia con un file che contiene dati non corretti (ad esempio una stringa).

Ora facciamo gli esercizi delle lezioni 7,8,9,10 del libro ma utilizzando il nostro ciclo di lettura (da copiare nella libreria personale dello studente), non quello del libro che non funziona bene.