# Esercizi di programmazione in linguaggio C – File

Nota: memorizzare i programmi in cartelle separate. Ogni cartella deve contenere, oltre al codice sorgente e al file eseguibile, i file di input indicati nel testo e gli eventuali file di output prodotti dal programma. I file di input devono essere generati con un apposito editor prima della scrittura del codice.

## Input/Output carattere per carattere

1. **stampavideo1** – Scrivere un programma che stampi a video il contenuto del file `in.txt`.
2. **stampavideo2** ★ Scrivere una funzione che riceva in ingresso il nome di un file e ne stampi a video il contenuto. Utilizzare la funzione per mostrare il contenuto dei file `in.txt` e `in2.txt`.
3. **contacaratteri** – Scrivere una funzione che, ricevuto in ingresso il nome di un file, restituisca il conteggio dei caratteri presenti al suo interno oppure -1 nel caso in cui non sia possibile accedere al file. Utilizzare la funzione per mostrare il numero di caratteri dei file `in.txt`, `in2.txt` e `nonesiste.txt` (si supponga che quest'ultimo file non sia presente nel disco), emettendo un apposito messaggio in presenza di errori di apertura dei file.
4. **contaparole** – Un file di testo contiene un certo numero di parole. Ogni parola (a eccezione dell'ultima) è separata da quella successiva da un solo carattere di *whitespace*. Scrivere un programma che, richiesta l'immissione da tastiera del nome del file da elaborare, visualizzi il conteggio delle parole contenute (suggerimento: contare i *whitespace*...).
5. **copiafile1** – Scrivere una funzione che, ricevuti in ingresso i nomi di due file, effettui la copia carattere per carattere del primo file nel secondo file. La funzione restituisce `true` se la copia si è conclusa con successo, `false` in caso contrario. Utilizzare la funzione per copiare il file `in.txt` nel file `out.txt`.
6. **copiafile2** – Scrivere un programma che effettui la copia un file di testo in un altro file trasformando le lettere minuscole in maiuscole e le maiuscole in minuscole.
7. **copialettere** – Scrivere un programma che, richiesto da tastiera il nome di un file testo, memorizzi i caratteri dell'alfabeto nel file `lettere.txt` e tutti gli altri caratteri in `altri_car.txt`.
8. **accodaparole** ★ Dopo aver creato due file di testo `parole1.txt` e `parole2.txt`, ciascuno contenente un elenco di parole separate da uno spazio, scrivere un programma che accodi il contenuto del primo file al secondo file. Alla fine dell'esecuzione, osservare il contenuto del secondo file con un editor testuale.

## Input/Output per linee di testo

9. **inverti** – Scrivere un programma che inverta ogni riga contenuta nel file `righe.txt` e riporti il risultato sullo schermo (per esempio, la riga `Prova di stampa` diventa `apmats id avorP`).
10. **filtralinee** ★ Scrivere una funzione che accetti come parametri d'ingresso due nomi di file e una parola P e memorizzi nel secondo file le sole righe del primo che contengono P (suggerimento: usare la funzione `strstr`).
11. **accodalinee** – Scrivere un programma che, chiesto in input il nome di un file di testo, accodi le linee del file al file di output `risultato.txt`, sostituendo le linee più lunghe di 20 caratteri con la stringa `--LINEA TROPPO LUNGA--`.
12. **contalinee** – Scrivere una funzione che accetti due parametri (un nome di file e una stringa `S`) e restituisca il

conteggio delle righe contenenti `S` oppure -1 nel caso di errori nella gestione del file. Dopo aver creato un file di testo, utilizzare la funzione per contare le linee del file che contengono la parola `BIANCO`.

13. **paginaweb** – Scrivere un programma che generi automaticamente una pagina web di nome `test.html` avente il seguente contenuto testuale:

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
        <title>Pagina generata automaticamente</title>
    </head>
    <body>
        <p style='color: red;'>
            Questa pagina &egrave; stata generata automaticamente
            da un programma scritto in linguaggio C.
        </p>
    </body>
</html>
```

## Input/Output formattato

14. **numeri** – Scrivere un programma che memorizzi in un file di testo e su una sola linea i primi 15 numeri pari separati da uno spazio.

15. **multipli** – Scrivere una funzione a due parametri interi A e N che memorizzi nel file di testo `output.txt` i primi N multipli di A (un solo multiplo per ogni riga).

16. **casuali** – Scrivere una funzione che, ricevuti in ingresso un nome di file e un numero intero N, memorizzi nel file N numeri casuali disposti a coppie su linee diverse (una coppia per ogni linea).

17. **statistica** – Scrivere un programma che, a partire da uno dei file prodotti nell'esercizio precedente, determini, per ogni coppia, una statistica formata dalla somma e dalla media dei due valori, riportando i risultati nel file `statistica.txt` (una statistica per ogni linea; all'interno della linea, separare la somma dalla media con un tabulatore).

18. **occorrenze** ★ Scrivere una funzione che, ricevuti in ingresso un nome di file e una parola `S`, restituisca il numero di occorrenze di `S` all'interno del file oppure -1 in caso di errore nella gestione del file. La funzione opera sotto le seguenti condizioni: una parola è una sequenza di caratteri priva di *whitespace*; un file è composto da una sequenza di parole separate da *whitespace*; la ricerca delle occorrenze è di tipo *case insensitive*.

19. **maxmin** – Scrivere un programma che calcoli il massimo e il minimo di una sequenza di valori interi contenuti in un file `numeri.txt` e memorizzi i risultati in `maxmin.txt` su due righe diverse nel seguente formato:

    ```
    Valore minimo: ....
    Valore massimo: ....
    ```

    Si ipotizzi di non conoscere a priori né il numero esatto di valori contenuti nel file né il numero massimo di valori previsto per questo esercizio.

20. **voti1** – Un file di testo `voti.txt` contiene i voti di un certo numero di studenti secondo il seguente formato:

    ```
    <cognome>  <nome>  <N>  <voto1>  <voto2> ...  <votoN>
    ```

    Poiché ogni studente può avere un numero diverso di voti, ogni riga include (dopo il nome) il numero N di voti registrati per quello studente, seguito da un elenco di N voti (approssimati a una cifra decimale). Un possibile esempio di file è il seguente:

    ```
    Bianchi  Paolo  3  7.0  7.5  6.0
    Rossi  Maria  4  8.5  8.0  8.0  9.0
    ```

    Scrivere un programma che acquisisca i voti degli studenti e produca in output:

    a) Il file `medie.txt` contenente le medie (approssimate a due cifre decimali) dei voti di ciascuno studente, nel formato:

       ```
       <cognome>  <nome>  <media>
       ```

    b) Il file `esiti.txt` contenente l'esito di fine anno scolastico di ogni studente: "`debito`" se la media è inferiore a 6, "`promosso`" in caso contrario. Il formato del file è:

       ```
       <cognome>  <nome>  <esito>
       ```

21. **misurecampo** ★ Si vuole realizzare un programma che calcoli il perimetro e l'area di un terreno poligonale. Utilizzando un ricevitore GPS si determinano innanzitutto le coordinate x,y degli N vertici del poligono. I dati sono successivamente memorizzati in un file di testo secondo il seguente formato:

    ```
    u_m
    N
    x0  y0
    x1  y1
    ...
    xN-1  yN-1
    ```

    La prima riga del file contiene una stringa rappresentante l'unità di misura in cui sono espresse le coordinate; la seconda riga contiene un intero `N` che indica il numero dei vertici del poligono; le successive `N` righe contengono le coordinate reali dei vertici (una coppia di coordinate per ogni riga).

    Il programma deve svolgere le seguenti operazioni:

    a) chiedere all'utente il nome del file di input e acquisirne l'intero contenuto, memorizzando le coordinate in un'apposita struttura dati;

    b) determinare il perimetro e l'area del terreno utilizzando le seguenti formule:

       perimetro: $P = \sum_{k=0}^{N-1} \sqrt{(x[(i+1) \bmod N] - x[i])^2 + (y[(i+1) \bmod N] - y[i])^2}$ (somma dei lati)

       area: $A = \frac{1}{2} \left| \sum_{k=0}^{N-1} \left( x[i] * y[(i+1) \bmod N] - x[(i+1) \bmod N] * y[i] \right) \right|$ (formula di Gauss)

    c) chiedere all'utente il nome di un file di ouput e memorizzare al suo interno i risultati ottenuti (valore e unità di misura di perimetro e area).

Per esempio, a partire dal seguente file di input:

```
dam
6
-0.57  -1.05
 4.05   8.19
15.60  10.50
24.84   3.57
20.22  -1.05
 8.67  -3.36
```

il programma deve riportare in un nuovo file i messaggi:

```
Perimetro:      61.50 dam
     Area:     229.45 dam^2
```

22. **scrivitab** – Scrivere un programma che legga da tastiera e memorizzi in un'apposita struttura dati la seguente tabella:

| Cognome | Nome | Altezza (m) | Peso (kg) |
|---|---|---|---|
| Rossi | Mario | 1.75 | 76 |
| Ferraro | Carlo | 1.84 | 82 |
| Marelli | Chiara | 1.65 | 58 |

Terminato il caricamento, il programma deve memorizzare l'intera tabella nel file di testo `tabella.txt` (un record per linea, campi separati con un tabulatore; non deve essere memorizzata la riga d'intestazione). Si ipotizzi che tutti i campi testuali siano privi di spazi, che le altezze siano numeri *float* e i pesi numeri interi.

23. **mostratab** – Scrivere un programma che acquisisca da un file di testo la tabella dell'esercizio precedente e la visualizzi sullo schermo nel seguente formato:

```
+-----------------+-----------------+-------------+-------------+
|     Cognome     |      Nome       | Altezza (m) |  Peso (kg)  |
+-----------------+-----------------+-------------+-------------+
| Rossi           | Mario           |    1.75     |      76     |
| Ferraro         | Carlo           |    1.84     |      82     |
| Marelli         | Chiara          |    1.65     |      58     |
+-----------------+-----------------+-------------+-------------+
```

## Input/Output a blocchi su file binari

24. **scrivi-bin** – Scrivere un programma che legga da tastiera due numeri (di tipo int) e memorizzi nel file binario `numeri.bin` i numeri dati, la somma (int) e la media (double). Analizzare il file generato con un editor esadecimale.

25. **leggi-bin** – Scrivere un programma che legga da un file binario tre numeri di tipo int, un numero di tipo double e ne visualizzi i valori. Utilizzare il programma per mostrare il contenuto del file binario generato nell'esercizio precedente.

26. **scrivi-vett-bin** – Scrivere un programma che memorizzi in un file binario il seguente array di cinque valori interi: `(7, 34, -12, 2, -8)`.

27. **leggi-vett-bin** – Scrivere un programma che legga da un file binario un vettore di cinque valori interi e ne visualizzi gli elementi. Utilizzare il programma per mostrare il contenuto del file binario generato nell'esercizio precedente.

28. **scrivi-str-bin** – Scrivere un programma che acquisisca da tastiera il cognome, il nome, l'età in anni di una persona e memorizzi le informazioni nel file binario `anagrafica.bin` senza usare le struct. Il cognome e il nome hanno una lunghezza massima di 20 caratteri e possono contenere spazi.

29. **leggi-str-bin** – Scrivere un programma che legga dal file `anagrafica.bin` il cognome, il nome, l'età in anni di una persona secondo il formato definito nell'esercizio precedente (senza usare le struct) e mostri i dati su monitor.

30. **auguri-agenda** – Il file agenda.bin contiene, in formato binario, le date di nascita di un certo numero di persone. Ogni persona è caratterizzata da un record composto dai seguenti campi:

    a) cognome (stringa di lunghezza massima pari a 31 caratteri, compreso il terminatore di stringa)
    b) nome (stringa, lunghezza massima 31 caratteri)
    c) giorno di nascita (numero di tipo int)
    d) mese di nascita (numero di tipo int)
    e) anno di nascita (numero di tipo int)

    Scrivere un programma che, dopo aver letto da tastiera la data odierna e aver memorizzato il contenuto dell'agenda in un'apposita struttura, mostri un messaggio di auguri rivolto alle persone che oggi compiono gli anni. Per esempio, supponendo che la data immessa sia 20/10/2016, il programma deve mostrare il seguente output:

```
Auguri a Pierangelo Botto (nato/a il 20/10/2005) per il suo 11° compleanno.
Auguri a Mirella Alongi (nato/a il 20/10/1991) per il suo 25° compleanno.
Auguri a Sarah Boero (nato/a il 20/10/2000) per il suo 16° compleanno.
Auguri a Marjeta Pavan (nato/a il 20/10/2002) per il suo 14° compleanno.
```

# Soluzioni degli esercizi

## Esercizio n. 2 (stampavideo2)

```
/* stampavideo2.c
 *
 * Funzione per la stampa del contenuto di un file.
 */

#include <stdio.h>
#include <stdlib.h>

void mostra_file(char nome_file[]) {
	FILE *fp;
	int c;

	fp = fopen(nome_file, "r");
	if (fp == NULL) {
		printf("Impossibile aprire il file %s.\n", nome_file);
		exit(1);
	}

	printf("\nContenuto del file %s\n", nome_file);
	printf("============================================================\n");
	c = fgetc(fp);
	while (c != EOF) {
		putchar(c);
		c = fgetc(fp);
	}

	fclose(fp);
}

int main( void ) {

	mostra_file("in.txt");
	mostra_file("in2.txt");

	return 0;
}
```

## Esercizio n. 8 (accodaparole)

```
/* accodaparole.c
 *
 * Accoda il contenuto di un file a un altro file.
 */

#include <stdio.h>

#define NOME_FILE_1 "parole1.txt"
#define NOME_FILE_2 "parole2.txt"

int main( void ) {

	FILE *fp1, *fp2;
	int c;

	fp1 = fopen(NOME_FILE_1, "r");
	if (fp1 == NULL) {
		printf("Impossibile aprire il file %s.\n", NOME_FILE_1);
		return 1;
	}

	fp2 = fopen(NOME_FILE_2, "a");
	if (fp2 == NULL) {
		fclose(fp1);
		printf("Impossibile aprire il file %s.\n", NOME_FILE_2);
		return 1;
	}

	c = fgetc(fp1);

	while (c != EOF) {
		fputc(c, fp2);		// Scrive il carattere letto in coda a fp2
		c = fgetc(fp1);		// Legge il carattere successivo da fp1
	}

	fclose(fp2);
	fclose(fp1);

	return 0;
}
```

## Esercizio n. 10 (filtralinee)

```
/* filtralinee.c
 *
 * Copia le linee di un file che contengono una data parola.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NOME_FILE_IN        "input.txt"
#define NOME_FILE_OUT       "output.txt"

#define MAX_PAROLA 30
#define MAX_LINEA 1000

void filtra(char nome_file1[], char nome_file2[], char p[]) {
        FILE *fp_in, *fp_out;
        char linea[MAX_LINEA];

        fp_in = fopen(nome_file1, "r");
        if (fp_in == NULL) {
                printf("Impossibile aprire il file %s.\n", nome_file1);
                exit(1);
        }

        fp_out = fopen(nome_file2, "w");
        if (fp_out == NULL) {
                fclose(fp_in);
                printf("Impossibile aprire il file %s.\n", nome_file2);
                exit(1);
        }

        while (fgets(linea, MAX_LINEA, fp_in) != NULL) {
                // La funzione strstr cerca la stringa memorizzata in P all'interno
                // dell'array LINEA (confronto tra caratteri di tipo 'case sensitive').
                // Restituisce un puntatore diverso da NULL solo se P e' una sottostringa
                // di LINEA.
                if (strstr(linea, p) != NULL)
                        fputs(linea, fp_out);
        }

        fclose(fp_out);
        fclose(fp_in);
}
```

```
int main( void ) {
	char parola[MAX_PAROLA + 1];

	printf("Parola da cercare: ");
	gets(parola);

	filtra(NOME_FILE_IN, NOME_FILE_OUT, parola);

	return 0;
}
```

## Esercizio n. 18 (occorrenze)

```
/* occorrenze.c
 *
 * Calcolo del numero di occorrenze di una parola all'interno di un file.
 */

#include <stdio.h>
#include <stdbool.h>
#include <ctype.h>

#define NOME_FILE    "parole.txt"

#define MAX_PAROLA 30

bool stringhe_uguali(char s1[], char s2[]) {
	// Restituisce TRUE solo se le due stringhe sono uguali.
	// Implementa un algoritmo di confronto "case insensitive".

	int i = 0;
	bool uguali = true;

	// Si ripete il confronto se i caratteri precedenti sono risultati uguali
	// e se c'e' ancora un carattere diverso da \0 in almeno una delle
	// due stringhe

	while (uguali == true && (s1[i] != '\0' || s2[i] != '\0'))
		if (toupper(s1[i]) != toupper(s2[i]))
			uguali = false;
		else
			i++;

	return uguali;
}

int conta_parole(char nome_file[], char s[]) {
	// Conta e resituisce il numero di occorrenze di S all'interno del file.
	// Restituisce -1 in caso di errori nella gestione del file.

	FILE *fp;
	char parola_file[MAX_PAROLA + 1];  // Conterra' una parola caricata dal file
	int occorrenze = 0;

	fp = fopen(nome_file, "r");
	if (fp == NULL)
		return -1;
```

```
	// La funzione fscanf restituisce il numero di dati letti, oppure 0
	// in presenza di EOF.

	while (fscanf(fp, "%s", parola_file) > 0)
		if (stringhe_uguali(parola_file, s) == true)
			occorrenze++;

	fclose(fp);

	return occorrenze;
}

int main( void ) {
	char p[MAX_PAROLA + 1];
	int n;

	printf("Parola da cercare: ");
	scanf("%s", p);

	n = conta_parole(NOME_FILE, p);

	if (n == -1)
		printf("Errore durante l'elaborazione del file.\n");
	else
		printf("Numero di occorrenze di %s: %d\n", p, n);

	return 0;
}
```

## Esercizio n. 21 (misurecampo)

```
/* misurecampo.c
 *
 * Calcola perimetro e area di un poligono note le coordinate dei suoi vertici.
 */

#include <stdio.h>
#include <math.h>

struct coord {
	double x;
	double y;
};

#define MAX_NOME_FILE 50
#define MAX_UM 10
#define MAX_VERTICI 100

double perimetro(struct coord vett[], int n) {
	double somma = 0.0;

	for(int i = 0; i < n; i++) {
		somma += sqrt(pow(vett[(i+1) % n].x - vett[i].x, 2) +
					pow(vett[(i+1) % n].y - vett[i].y, 2)
				);
	}

	return somma;
}

double area(struct coord vett[], int n) {
	double somma = 0.0;

	for(int i = 0; i < n; i++)
		somma += vett[i].x * vett[(i+1) % n].y - vett[(i+1) % n].x * vett[i].y;

	return 0.5 * fabs(somma);
}

int main( void ) {
	FILE *fp;
	char um[MAX_UM + 1];
	char nome_file[MAX_NOME_FILE + 1];
	struct coord vertici[MAX_VERTICI];
	int n;
	double p, a;
```

```
	printf("Nome del file da elaborare: ");
	gets(nome_file);

	if ((fp = fopen(nome_file, "r")) == NULL) {
		printf("Impossibile aprire il file %s.\n", nome_file);
		return 1;
	}

	fscanf(fp, "%s", um);
	fscanf(fp, "%d", &n);

	for (int i = 0; i < n; i++)
		fscanf(fp, "%lf%lf", &vertici[i].x, &vertici[i].y);

	fclose(fp);	// File chiuso, si possono riutilizzare
				// le variabili fp e nome_file

	p = perimetro(vertici, n);
	a = area(vertici, n);

	printf("Nome del file di output: ");
	gets(nome_file);

	if ((fp = fopen(nome_file, "w")) == NULL) {
		printf("Impossibile scrivere nel file %s.\n", nome_file);
		return 1;
	}

	fprintf(fp, "Perimetro: %10.2lf %s\n", p, um);
	fprintf(fp, "	Area: %10.2lf %s^2\n", a, um);

	fclose(fp);


	return 0;
}
```

## Esercizio n. 26 (scrivi-vett-bin)

```
/*
 *        File: scrivi-vett-bin.c
 *
 *        Data: 19/10/2016
 *
 * Descrizione: Scrivere un programma che acquisisca da tastiera il cognome, il
 *              nome, l'eta' in anni di una persona e memorizzi le informazioni
 *              nel file binario "anagrafica.bin" senza usare le struct.
 *              Il cognome e il nome hanno una lunghezza massima di 20 caratteri
 *              e possono contenere spazi.
 *
 */

#include <stdio.h>
#define MAX_COGNOME 20
#define MAX_NOME 20

int main( void ) {
       char cognome[MAX_COGNOME + 1];
       char nome[MAX_NOME + 1];
       int eta;
       FILE *fp;

       printf("Cognome: ");
       gets(cognome);
       printf("Nome: ");
       gets(nome);
       printf("Eta': ");
       scanf("%d", &eta);

       if ((fp =fopen("anagrafica.bin", "wb")) == NULL) {
              printf("Errore durante l'apertura del file.\n");
              return 1;
       }
       fwrite(cognome, sizeof(cognome), 1, fp);
       fwrite(nome, sizeof(nome), 1, fp);
       fwrite(&eta, sizeof(int), 1, fp);
       fclose(fp);

       return 0;
}
```

## Esercizio n. 27 (leggi-vett-bin)

```
/*
 *        File: leggi-vett-bin.c
 *
 *        Data: 19/10/2016
 *
 * Descrizione: Scrivere un programma che legga da un file binario un vettore di
 *              cinque valori interi e ne visualizzi gli elementi.
 *              Utilizzare il programma per mostrare il contenuto del file
 *              binario generato nell'esercizio "scrivi-vett-bin".
 */

#include <stdio.h>
#define MAX_DIM 5

int main( void ) {
       int vett[MAX_DIM];
       FILE *fp;

       if ((fp =fopen("vett-int.bin", "rb")) == NULL) {
              printf("Errore durante l'apertura del file.\n");
              return 1;
       }

       fread(vett, sizeof(vett), 1, fp);

       fclose(fp);

       for (int i = 0; i < MAX_DIM; i++)
              printf("%d\n", vett[i]);

       return 0;
}
```