Fondamenti di Informatica T-1
Modulo 2

Obiettivo di questa esercitazione

- File di testo
- File binari

Esercizio 1
(file)

- Realizzare un programma che, aperto un file di testo di nome “Prova.txt” in modalità “scrittura”, provveda a leggere da input delle parole separate da spazi (stringhe di al più 63 caratteri) e le scriva nel file di testo.
- Il programma termina quando l’utente inserisce la parola “fine”. Si abbbia cura di chiudere il file prima di terminare definitivamente il programma.
- Si controlli il corretto funzionamento del programma accedendo direttamente al file di testo con un editor (ad es. Notepad).

Esercizio 1 - Soluzione

(file)

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

int main(void) {
    FILE * fp;
    char s[64];

    if ((fp=fopen("prova.txt", "w")) == NULL)
      exit(1);

    do {
      scanf("%s", s);
      if (strcmp("fine", s) != 0)
       fprintf(fp, "%s ", s);
    } while (strcmp("fine", s) != 0);

    fclose (fp);
    system("PAUSE");
    return (0);
}
```

Esercizio 2
(file)

Sia dato il file di testo "dati.txt" contenente i dati relativi agli studenti immatricolati al primo anno della Facoltà di Ingegneria

In particolare, le informazioni sono memorizzate nel file "dati.txt" come segue: ognuna delle linee del file contiene i dati relativi ad un nuovo studente; in particolare:

1 Matricola: un intero che indica il numero di matricola dello studente

2 CdL: un intero che indica il corso di laurea (CdL) dello studente (es. 2145)

Esercizio 2
(file)

Sia dato un secondo file di testo, “indirizzi.txt” che contiene, invece, l’indirizzo di ogni studente, e in particolare, in ogni linea, separati da uno spazio:

- Matricola: il numero di matricola dello studente (un intero)
- Nome: il nome dello studente, al più 20 caratteri senza spazi
- Cognome: il cognome dello studente, al più 30 caratteri senza spazi
- Via: una stringa di al più 30 caratteri senza spazi, che riporta la via di residenza dello studente
- Città: una stringa che riporta la città di residenza dello studente, al più 30 caratteri senza spazi
- CAP: un intero che rappresenta il codice di avviamento postale dello studente

Esercizio 2

(file)

Si scriva un programma in linguaggio C che:

1. A partire dai file "dati.txt" e “indirizzi.txt” **costruisca una tabella T** contenente, per ogni studente, Matricola, Nome, Cognome, Via, Città, CAP e CdL

Si ricorda l’esistenza della procedura di libreria void rewind (FILE *f) che riporta la testina di lettura a inizio file

Esercizio 2

(file)

2. A partire dalla tabella T, e dato da input un intero C che rappresenta un CdL, **stampi la percentuale di studenti** (rispetto al numero totale delle matricole) **iscritti al corso C** [Ad esempio, se il numero totale delle matricole è 1000, e quello degli studenti iscritti a C è 200, il programma stamperà “20%”]

3. Scriva su un terzo file di testo “bologna.txt”, nome, cognome e numero di matricola di tutti gli studenti che abitano a Bologna

Esercizio 2 - Soluzione

(file)

```
typedef struct {
    unsigned int matr;
    unsigned int CDL;
} dati;

typedef struct {
    unsigned int matr;
    char nome[21];
    char cognome[31];
    char via[31];
    char citta[31];
    unsigned int CAP;
} indirizzo;
```

```
typedef struct {
    unsigned int matr;
    char nome[21];
    char cognome[31];
    char via[31];
    char citta[31];
    unsigned int CAP;
    unsigned int CDL;
} elemento;

typedef elemento tabella[10];
```

Esercizio 2 - Soluzione

(file)

```
elemento riempiel(dati d, indirizzo i){
     elemento e;
     e.matr=d.matr;
     e.CDL=d.CDL;
     strcpy(e.nome, i.nome);
     strcpy(e.cognome, i.cognome);
     strcpy(e.via, i.via);
     strcpy(e.citta, i.citta);
     e.CAP=i.CAP;
     return e;
}

int main() {
     dati D;
     indirizzo I;
     elemento E;
     tabella T;
     FILE *f1, *f2;
     int i, trovato, ins=0, totC;
     unsigned int C;
```

Esercizio 2 - Soluzione

(file)

```
/*domanda 1: costruzione della tabella */

    f1=fopen("dati.txt", "r");
    f2=fopen("indirizzi.txt", "r");

    while (fscanf(f1,"%u%u", &D.matr, &D.CDL)>0) {
          trovato=0;
          rewind(f2);
          while(fscanf(f2, "%d %s %s %s %s %d",
                           &I.matr, I.nome, I.cognome,
                           I.via, I.citta, &I.CAP)==6
                     && !trovato)
                if (I.matr==D.matr) {
                     trovato=1;
                     E=riempiel(D, I);
                     T[ins]=E;
                     ins++;
                }
     }

    fclose(f1);fclose(f2);
    ...
```

Esercizio 2 - Soluzione

(file)

```
    /*domanda 2: stampa della percentuale degli
   iscritti a un corso dato*/
    printf("Inserire il corso C: ");
    scanf("%u", &C);
    totC=0;
    for(i=0; i<ins; i++)
      if(T[i].CDL==C)
           totC++;

    printf("\n Iscritti al corso %u: %f \%\n",
           C, (float)totC*100/ins);


    /*domanda 3: scrittura di "bologna.txt" */
    f1=fopen("bologna.txt", "w");
    for (i=0; i<ins; i++)
      if (strcmp("bologna", T[i].citta)==0)
           fprintf(f1,"%s %s %u\n",T[i].nome,T[i].cognome,T[i].matr);
    fclose(f1);
    return 0;
}
```

Esercizio 3
(file)

Sono dati due file di testo cineprogramma.txt e sale.txt che contengono, rispettivamente, il programma settimanale dei film in proiezione e le descrizioni delle sale in città. Più precisamente, ogni riga di cineprogramma.txt contiene, nell'ordine:

- titolo del film (non più di 30 caratteri senza spazi), uno e un solo spazio di separazione
- nome della sala (non più di 20 caratteri senza spazi), uno e un solo spazio di separazione
- 3 orari di inizio proiezione (3 numeri interi separati da caratteri '-'), terminatore di riga

mentre ogni riga di sale.txt contiene, nell'ordine:

- nome della sala (non più di 20 caratteri senza spazi), uno e un solo spazio di separazione
- costo del biglietto (numero reale), terminatore di riga

Esercizio 3
(file)

cineprogramma.txt:

```
TheKingdom Nosadella 18-20-22
Dogville Fellini 17-20-22
OttoEMezzo Capitol 17-20-23
BreakingWaves Odeon 15-19-23
```

sale.txt:

```
Capitol 6.00
Fellini 5.50
Modernissimo 6.00
Nosadella 6.50
```

Esercizio 3
(file)

1) Si scriva una procedura load() che riceva come parametri di ingresso due puntatori a file di testo e restituisca come parametri di uscita **un vettore y contenente strutture film** (titolo film, costo biglietto) **e il numero degli elementi N inseriti in y**

Per semplicità si supponga che tutte le sale contenute nel primo file siano presenti anche nel secondo, e una sola volta

Si ricorda inoltre l'esistenza della procedura di libreria void rewind (FILE *f) che riporta la testina di lettura a inizio file

Esercizio 3
(file)

2) Si scriva un programma C che, utilizzando la procedura load() precedentemente definita, **inserisca in un vettore prezzi** (supposto di dimensione massima DIM=100) **le strutture film di cui sopra**, derivanti dai file cineprogramma.txt e sale.txt

Il programma deve inoltre stampare a terminale tutti gli elementi di prezzi il cui costo del biglietto è **inferiore alla media di tutti i costi caricati nel vettore**

Esercizio 3 - Soluzione

(file)

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

#define DIM 100
typedef struct {
    char titolo[31];
    float prezzo;
} film;

void load(FILE * f1, FILE * f2, film * y, int * num) {
    char titolo[31], sala1[21], sala2[21];
    int orario; float p; *num=0;

    while (fscanf(f1,"%s %s %d-%d-%d\n", titolo, sala1, &orario,
                                          &orario, &orario) != EOF) {
       while ((fscanf(f2,"%s %f\n",sala2,&p) != EOF)&&
                              (strcmp(sala1,sala2))); //nessuna azione
// supponiamo per semplicità che la sala sia sempre compresa in f2
       strcpy(y[*num].titolo, titolo);
       y[*num].prezzo=p;
       (*num)++;
       rewind(f2); }
}
```

Esercizio 3 - Soluzione

(file)

```
int main() {
   FILE *f1, *f2; int N, i;
   float somma,media; film prezzi[DIM];

   if ((f1=fopen("cineprogramma.txt", "r"))==NULL) {
     printf("Non sono riuscito ad aprire file1 in lettura!");
     exit(1);
   }
   if ((f2=fopen("sale.txt", "r"))==NULL) {
     printf("Non sono riuscito ad aprire file2 in lettura!");
     exit(1);
   }

   load(f1,f2,prezzi,&N);
   fclose(f1); fclose(f2);
   ...
```

Esercizio 3 - Soluzione

(file)

```
    ...
    somma=0;
    for (i=0; i<N; i++) somma=somma+prezzi[i].prezzo;
    media=somma/N;

    for (i=0; i<N; i++)
      if (prezzi[i].prezzo<media)
            printf("Il costo del biglietto per il film %s è
            %f\n", prezzi[i].titolo, prezzi[i].prezzo);
}
```

Esercizio 4
(file)

Sono dati due file di testo anagrafe.txt e fatture.txt che contengono, rispettivamente, i dati anagrafici di alcuni clienti e l'elenco delle fatture

Più precisamente, ogni riga di anagrafe.txt contiene, nell'ordine:

- Codice Cliente (numero intero) , uno e un solo spazio di separazione
- Nome del cliente (non più di 30 caratteri senza spazi), uno e un solo spazio di separazione
- Città (non più di 20 caratteri senza spazi), uno e un solo spazio di separazione

Ogni cliente compare nel file di anagrafe una ed una sola volta

Ogni riga di fatture.txt contiene, nell'ordine:

- Codice Cliente (numero intero), uno e un solo spazio di separazione
- Numero della fattura (numero intero), uno e un solo spazio di separazione
- Importo della fattura (numero reale), uno e un solo spazio di separazione
- Un carattere ('p' se la fattura è stata pagata, 'n' altrimenti), terminatore di riga

Esercizio 4

(file)

anagrafe.txt:

```
1 Chesani Bologna
2 Bellavista Bologna
3 Mello Bologna
```

fatture.txt:

```
1 23 54.00 p
1 24 102.00 n
3 25 27.00 p
1 26 88.00 n
```

Esercizio 4
(file)

1) Si scriva una procedura load() che riceva come parametri di ingresso due puntatori a file di testo e restituisca come parametri di uscita **un vettore y contenente strutture debito** (nome cliente, importo) e il numero degli elementi N inseriti in y: questo vettore deve contenere solo i dati relativi a **fatture non pagate**

Si ricorda inoltre l'esistenza della procedura di libreria void rewind (FILE *f) che riporta la testina di lettura a inizio file

Esercizio 4
(file)

2) Si scriva un programma C che, utilizzando la procedura **load()** precedentemente definita, inserisca in un vettore **debitori** (supposto di dimensione massima **DIM=100**) le strutture **debito** di cui sopra, derivanti dai file **anagrafe.txt** e **fatture.txt**

Il programma chieda poi all'utente il nome di un cliente e stampi il numero di fatture (non pagate) intestate a tale cliente e la somma totale degli importi dovuti

Esercizio 4 - Soluzione

(file)

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

#define DIM 100

typedef struct {
   char nome[31];
   float importo;
} debito;
```

Esercizio 4 - Soluzione

(file)

```
void load(FILE * f1, FILE * f2, debito * y, int * num) {
   int codice1, codice2;
   char nome[31], citta[21], pagato;
   int num_fattura; float importo;

   *num=0;
   while(fscanf(f1,"%d %s %s\n", &codice1, nome, citta) != EOF){
         while ( fscanf(f2,"%d %d %f %c\n", &codice2,
               &num_fattura,&importo, &pagato) != EOF )

               if ( (codice1==codice2) && (pagato == 'n') ) {
                     strcpy(y[*num].nome, nome);
                     y[*num].importo = importo;
                     (*num)++;
               }
         rewind(f2);
   }
}
```

Esercizio 4 - Soluzione

(file)

```
int main() {
   FILE *f1, *f2; int N, i, count; float somma;
   char nome[31]; debito debitori[DIM];

   if ((f1=fopen("anagrafe.txt", "r"))==NULL) {
    printf("Non sono riuscito ad aprire file1 in lettura!");
    exit(1);
   }
   if ((f2=fopen("fatture.txt", "r"))==NULL) {
    printf("Non sono riuscito ad aprire file2 in lettura!");
    exit(1);
   }

   load( f1, f2, debitori, &N);
   fclose(f1); fclose(f2);
   ...
```

Esercizio 4 - Soluzione

(file)

```
  ...
  printf("Inserire nome cliente: ");
  scanf("%s", nome);

  somma = 0; count = 0;
  for (i=0; i<N; i++) {
    if (! strcmp(nome, debitori[i].nome)){ /* strcmp==0 */
      somma = somma + debitori[i].importo;
      count++;
    }
  }

  printf("%d fatture per l'ammontare di: %f", count, somma);
}
```

Esercizio 5

(file)

- Un registratore di cassa registra su di un file binario alcuni dati relativi agli scontrini emessi. In particolare, tramite una struttura dati di nome scontrino, il registratore di cassa tiene traccia dell'importo (un float) e del numero di oggetti acquistati (un intero).
- In un apposito file registratore.h, si definisca una struttura dati "scontrino" atta a contenere/rappresentare i dati forniti dal registratore di cassa.
- In un apposito modulo software registratore.h / registratore.c, si devono realizzare due funzioni, una di lettura ed una di scrittura delle strutture dati scontrino:

```
int leggi(FILE * fp, scontrino * dest);
int scrivi(FILE * fp, scontrino src);
```

Esercizio 5

(file)

- Si realizzi un programma main che apra il file binario reg.dat in scrittura, e poi simuli il funzionamento del registratore di cassa per testare il modulo registratore. In particolare il programma chieda all'utente di inserire coppie importo / numero di oggetti, e le registri sul file binario usando la funzione scrivi(…). L'utente può segnalare la fine dell'inserimento dei valori indicando una coppia 0.00/0, che ovviamente non deve essere registrata nel file.
- Terminata la fase di inserimento, il programma chiuda il file e lo ri-apra in lettura, e stampi a video tutte le coppie memorizzate (per la lettura, si utilizzi la funzione leggi(…))
- Al termine il main chiuda correttamente il file.

Esercizio 5 - Soluzione
(file)

- “registratore.h”:

```
#include <stdio.h>

typedef struct {
      float val;
      int num;
} scontrino;


int leggi(FILE * fp, scontrino * dest);
int scrivi(FILE * fp, scontrino src);
```

Esercizio 5 - Soluzione
(file)

- “registratore.c”:

```
#include "registratore.h"


int leggi(FILE * fp, scontrino * dest) {
      return fread(dest, sizeof(scontrino), 1, fp);
}

int scrivi(FILE * fp, scontrino src) {
      return fwrite(&src, sizeof(scontrino), 1, fp);
}
```

Esercizio 5 - Soluzione

(file)

- "main.c":

```
#include <stdio.h>
#include <stdlib.h>
#include "registratore.h"

int main(void) {
        scontrino temp;
        FILE * fp;

        if ( (fp=fopen("reg.dat", "wb")) == NULL)
                exit(-1);
        do {
                printf("Inserisci valore e numero di pezzi: ");
                scanf("%f %d", &(temp.val), &(temp.num) );
                if (temp.val!=0 || temp.num!=0)scrivi(fp, temp);
        } while (temp.val!=0 || temp.num!=0);
        fclose(fp);

        if ( (fp=fopen("reg.dat", "rb")) == NULL)
                exit(-1);
        while (leggi(fp, &temp)>0)
                printf("Prezzo: %f, pezzi: %d\n", temp.val, temp.num);
        fclose(fp);
        system("PAUSE"); return (0); }
```

Esercizio 6

(file)

- Un sito web per le speculazioni borsistiche registra su un file di testo gli andamenti di alcuni titoli azionari. In particolare, su ogni riga del file, registra il nome di un titolo (al più 63 caratteri senza spazi), i valori di apertura e di chiusura del titolo (tramite due float), ed il giorno dell'anno corrente in cui è stato monitorato il titolo (un intero).
- Nel file non c'è nessun ordine preciso con cui le righe sono state memorizzate, e mano a mano che passano i giorni vengono registrate più e più righe relative allo stesso titolo. Il sito web ha comunque l'accortezza di registrare nel file al massimo 100 righe relative allo stesso titolo.
- Si vuole realizzare un programma per valutare la volatilità di un titolo (cioè i valori minimi e massimi raggiunti), relativamente alle informazioni presenti sul file.

Esercizio 6

(file)

A tal scopo, in un opportuno modulo software azioni.h/azioni.c:

- Si definisca una apposita struttura dati, di nome “azione”, per memorizzare le informazioni relative ad un titolo.
- Si realizzi una funzione:

```
int leggi(FILE* fp, azione dest[], int dim, char * nome);
```

- che, ricevuti in ingresso un puntatore ad un file già opportunamente aperto, un array di strutture azione e la sua dimensione fisica dim, ed il nome di un titolo azionario, provveda a copiare in dest tutte le strutture relative all’azione specificata. La funzione deve restituire il numero di elementi copiati, ovvero la dimensione logica del vettore dest.

Esercizio 6
(file)

Quindi, in un altro modulo software “trova.h/trova.c”:

- Si realizzi una funzione:

```
azione trovaMin(azione src[], int dim, float* val);
```

  che, ricevuti in ingresso un array di strutture azione e la sua dimensione dim, ed il nome di una azione, determini il valore minimo raggiunto dal titolo azionario (le cui fluttuazioni sono memorizzate in src). La funzione deve restituire la struttura dati azione relativa a quando si è verificato il minimo, e tramite il parametro val passato per riferimento deve restituire il minimo raggiunto.

- Si realizzi una funzione:

```
azione trovaMax(azione src[], int dim, float* val);
```

  che, ricevuti in ingresso un array di strutture azione e la sua dimensione dim, ed il nome di una azione, determini il valore massimo raggiunto dal titolo azionario (le cui fluttuazioni sono memorizzate in src). La funzione deve restituire la struttura dati azione relativa a quando si è verificato il massimo, e tramite il parametro val passato per riferimento deve restituire il massimo raggiunto.

Esercizio 6
(file)

- Si realizzi un programma main.c che, utilizzando le funzioni già definite, provveda a chiedere all'utente il nome di un titolo azionario, e stampi a video i valori minimi e massimi raggiunti dalle quotazioni del titolo.
- Si estenda il modulo "trova.h/trova.c" con una funzione opportuna per calcolare la media del valore del titolo azionario rispetto alle varie quotazioni registrate, e poi si provveda a stampare a video l'oscillazione (in percentuale rispetto al valore medio) del minimo e del massimo raggiunti dal titolo in borsa.

Esercizio 6 - Soluzione

(file)

- "azioni.h":

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


typedef struct {
        char nome[64];
        float ap;
        float cl;
        int day;
} azione;


int leggi(FILE* fp, azione dest[], int dim, char * nome);
```

Esercizio 6 - Soluzione

(file)

- “azioni.c”:

```
#include "azioni.h"


int leggi(FILE* fp, azione dest[], int dim, char * nome) {
        azione temp;
        int result = 0;

        while (fscanf(fp, "%s %f %f %d", temp.nome, &(temp.ap),
                                &(temp.cl), &(temp.day)) == 4 && result<dim) {
                if (strcmp(temp.nome, nome) == 0) {
                        dest[result] = temp;
                        result++;
                }
        }
        return result;
}
```

Esercizio 6 - Soluzione

(file)

- “trova.h”:

```
#include "azioni.h"

azione trovaMin(azione src[], int dim, float* val);
azione trovaMax(azione src[], int dim, float* val);
float media(azione src[], int dim);
```

Esercizio 6 - Soluzione

(file)

- "trova.c":

```
#include "trova.h"

azione trovaMin(azione src[], int dim, float* val) {
        int i;
        azione result;

        result = src[0]; // ATTENZIONE!!! SE dim == 0?
        if (result.ap < result.cl)
                *val = result.ap;
        else
                *val = result.cl;
        for (i=1; i<dim; i++) {
                if (src[i].ap < *val) {
                        *val = src[i].ap;
                        result = src[i];
                }
                if (src[i].cl < *val) {
                        *val = src[i].cl;
                        result = src[i];
                }
        }
        return result;
}
```

Esercizio 6 - Soluzione

(file)

- "trova.c":

```
azione trovaMax(azione src[], int dim, float* val) {
        int i;
        azione result;

        *val = -1;
        result = src[0]; // ATTENZIONE!!! SE dim == 0?

        for (i=0; i<dim; i++) {
                if (src[i].ap > *val) {
                        *val = src[i].ap;
                        result = src[i];
                }
                if (src[i].cl > *val) {
                        *val = src[i].cl;
                        result = src[i];
                }
        }
        return result;
}
```

Esercizio 6 - Soluzione

(file)

- "trova.c":

```
float media(azione src[], int dim) {
        int tot=0;
        int i;
        float sum=0;

        for (i=0; i<dim; i++) {
                sum = sum + src[i].ap + src[i].cl;
                tot+=2;
        }
        return sum/tot;
}
```

Esercizio 6 - Soluzione

(file)

- "main.c":

```
#include <stdio.h>
#include <stdlib.h>
// #include "azioni.h" NON NECESSARIO perche' gia' incluso da trova.h...
#include "trova.h"

int main(void) {
        FILE * fp;
        azione temp1, temp2;
        azione lista[100];
        int dim;
        float min, max, med;
        char nome[64];

        if ((fp=fopen("azioni.txt", "r")) == NULL)
                exit(-1);
        printf("Nome titolo azionario: ");
        scanf("%s", nome);
        dim = leggi(fp, lista, 100, nome);
        temp1 = trovaMin(lista, dim, &min);
        temp2 = trovaMax(lista, dim, &max);

        ...
```

Esercizio 6 - Soluzione

(file)

- "main.c":

```
        ...

        printf("Minimo raggiunto da %s il giorno %d, valore %f euro.\n",
                temp1.nome, temp1.day, min);
        printf("Massimo raggiunto da %s il giorno %d, valore %f euro.\n",
                temp2.nome, temp2.day, max);
        med = media(lista, dim);
        printf("Media: %f\n", med);
        printf("Volatilita': %f...%f\n", (min-med)/med*100, (max-med)/med*100);


        system("PAUSE");

        return (0);
}
```

Esercizio 7

(file)

Un venditore di ortofrutta è solito fare credito ai propri clienti. Al fine di tenere traccia dei crediti, utilizza una funzione del registratore di cassa che salva, su un file binario di nome "log.dat", strutture dati del tipo `transaction` contenenti i seguenti dati:

- una stringa `customer`, contenente il nome del cliente (al più 128 caratteri, senza spazi);
- un intero `transactionId`, recante un codice identificativo della transazione;
- un float `value`, contenente l'ammontare del credito concesso.

```
#define DIM 129
typedef struct {
        char customer[DIM];
        int transactionId;
        float value;
} transaction;
```

Esercizio 7

(file)

Al momento di riscuotere i crediti, il commerciante deve però poter accedere ai valori registrati nel file binario.

A tal scopo, egli vuole avere un programma che, richiesto il nome del cliente, scriva su un file in formato testo gli importi relativi solo al cliente specificato.

Inoltre il file deve avere come nome il nome del cliente con l'aggiunta dell'estensione "`.txt`". Ad esempio, se il cliente richiesto si chiama "Federico", allora il file dovrà chiamarsi "`Federico.txt`".

Esercizio 7

(file)

In un apposito modulo software, denominato "registratore.h / registratore.c", dopo aver definito opportunamente le strutture dati necessarie, si realizzi una procedura

```
void copy(FILE* source, FILE* dest, char *name, int *result)
```

che, ricevuti in ingresso un puntatore **source** al file binario, un puntatore **dest** al file di testo, un puntatore a carattere **name** e un intero **result** passato per riferimento, copi su **dest** tutti gli importi presenti in **source** e relativi al cliente specificato con parametro **name**

La funzione deve tenere traccia del numero di importi di credito copiati e deve restituire tale numero tramite il parametro **result**. Gli importi devono essere scritti su una sola linea, separati da uno spazio, con al termine un carattere di "newline" ('**\n**'). Al fine di confrontare due stringhe, si utilizzi la funzione **strcmp(…)**, che restituisce 0 se le due stringhe passate come parametri sono identiche

Esercizio 7

(file)

Realizzare poi un programma "main.c" che chieda inizialmente all'utente il nome di un cliente. Per creare un opportuno nome per il file di destinazione, il candidato può utilizzare le funzioni di libreria:

- `strcpy(char *s, char *ct)`, che provvede a copiare la stringa ct nella stringa s;
- `strcat(char * s, char * ct)`, che concatena il contenuto della stringa ct in fondo alla stringa s (si faccia particolare attenzione a dimensionare opportunamente la stringa s per contenere i 4 caratteri dell'estensione ".txt"). La stringa s, al termine dell'invocazione, è sempre una stringa ben formata

Dopo aver aperto i file nell'opportuna modalità di lettura/scrittura, il programma utilizzi la funzione copy(…) definita al punto precedente per filtrare i dati. Il programma stampi infine a video il numero totale di crediti che sono stati copiati da un file all'altro

Esercizio 7 - Soluzione

(file)

```
void copy(FILE *source, FILE *dest, char *name, int
*result) {

      transaction temp;
      *result = 0;

      while (
        fread(&temp, sizeof(transaction), 1, source) >0
      ) {
            if (strcmp(name, temp.customer) == 0) {
                  fprintf(dest, "%f ", temp.value);
                  (*result)++;
            }
      }
      fprintf(dest, "\n");
}
```

Esercizio 7 - Soluzione

(file)

```
int main() {
        char name[DIM], filename[DIM+4];
        FILE *source, *dest;
        int result = 0;

        printf("Insert customer name: ");
        scanf("%s", name);
        if ((source = fopen("log.dat", "rb")) == NULL) {
                        printf("Error opening the file %s\n", "log.dat");
                        exit(-1);}

        strcpy(filename, name);
        strcat(filename, ".txt");
        if ((dest = fopen(filename, "w")) == NULL) {
                        printf ("Error opening the file %s\n", filename);
                        exit(-1);}

        copy(source, dest, name, &result);

        fclose(source); fclose(dest);
        printf("%d records copied by log.dat to %s\n",result,filename);
        return 0;
}
```

Esercizio 8

(file)

- È dato un file di testo **PEOPLE.TXT** che contiene i dati di una serie di persone (non più di 20), una persona per riga
- Si vuole realizzare un programma che, una volta letti da file i dati di queste persone, ne estragga l'insieme di persone compatibili con una nuova persona data, salvando il risultato sul file binario **PARTNERS.DAT**
  - ***Due persone sono compatibili*** se sono di sesso diverso e la differenza di età, *riferita solo all'anno*, non supera i 5 anni

Esercizio 8

(file)

- Ogni riga del file contiene, nell'ordine:
  - cognome (non più di 20 caratteri)
  - un separatore ‘;’
  - nome (non più di 20 caratteri)
  - un separatore ‘;’
  - data di nascita nel formato **gg/mm/aaaa**
  - uno e un solo spazio
  - un carattere ('M' o 'F') che indica il sesso

- Si definisca opportunamente una struttura dati **persona** di tipo struct

Esercizio 8

(file)

Realizzare le seguenti funzioni

- una funzione **lettura(…)** che, dato il nome del file (ed eventualmente altri parametri se opportuno), legga i dati delle persone dal file e li metta in un *array di* ***persona*** di nome **elenco**
  - Quanto deve essere grande l'array?
  - Si mostri a video l'array così costruito
- una funzione **compatibili(…)** che, date due persone, restituisca *vero* solo se le due persone sono compatibili
  - si mostri a video un esempio d'uso della funzione con due persone scelte all'array a vostro piacere
- **main()** che invochi **lettura** per acquisire le persone, poi chieda all'utente i dati di una nuova persona per scrivere infine su file binario l'insieme delle persone **compatibili** con la persona data

Esercizio 8
(file)

- Analizzare i “dati”
  - Quanti “dati complessi” sono chiamati in causa?
    - Persona
    - Data (di nascita)
  - Quali campi? E di che tipo (e dimensione…)?
- Seguire un approccio a moduli
  - Sicuramente un ***modulo separato*** per la definizione delle strutture dati
  - Un modulo per le funzioni chiamate a manipolare queste strutture dati
- Lettura da file di testo
  - Individuare il tipo dei campi che vogliamo leggere
  - …e i relativi separatori

Esercizio 8 - Soluzione

(file)

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

#define NUMEROPERSONE 20
#define DIMCOGNOME 21
#define DIMNOME 21

typedef struct dataStruct
{
      int giorno, mese, anno;
} Data;

typedef struct personaStruct
{
    char cognome[DIMCOGNOME], nome[DIMNOME], sesso;
    Data nascita;
} Persona;
```

Esercizio 8 - Soluzione

(file)

```
void lettura(  char nomefile[],
               Persona v[],
               int* pindice)
{
    Persona x;
    Boolean more = true;
    *pindice = 0;
    FILE *f = fopen(nomefile, "r");
    if (f == NULL)
    {
      printf("Impossibile aprire file di
    ingresso");
      exit(1);
    }
    ...
```

Esercizio 8 - Soluzione

(file)

```
    do
    {
        more = readField(x.cognome, ';', f);
        more = more && readField(x.nome, ';', f);
        more = more && fscanf(f, "%d/%d/%d %c\n",
                                &x.nascita.giorno,
                                &x.nascita.mese,
                                &x.nascita.anno,
                                &x.sesso);
        if(more)
        {
                v[*pindice] = x;
                (*pindice)++;
        }
    }
    while(more);
    fclose(f);
}
```

Esercizio 8 - Soluzione

(file)

```
Boolean compatibili(Persona p1,
                    Persona p2)
{
    Boolean compSesso,compAnno;
    compSesso = ( p1.sesso != p2.sesso );
    compAnno =
      abs(p1.nascita.anno - p2.nascita.anno)<=5;
    return compSesso && compAnno;
}
```

Esercizio 8 - Soluzione

(file)

```
int main()
{
    Persona elenco[NUMEROPERSONE], utente;
    int indiceElenco = 0, i;
    FILE *fbin;
    lettura("PEOPLE.TXT", elenco, &indiceElenco);
    printf("\n\nNome e cognome: ");
    gets(utente.nome);
    gets(utente.cognome);
    printf("data di nascita (gg/mm/aaaa): ");
    scanf("%d/%d/%d",
            &utente.nascita.giorno,
            &utente.nascita.mese,
            &utente.nascita.anno);
    /* sopprime il fine linea rimasto sull'input */
    scanf("%*c");
    printf("Inserire il sesso (M/F): ");
    scanf("%c%*c",&utente.sesso);
    ...
```

Esercizio 8 - Soluzione

(file)

```
...
fbin = fopen("PARTNERS.DAT", "wb");
if (fbin==NULL)
{
  printf("Impossibile aprire file di uscita\n");
  exit(2);
}
printf("Compatibili con %s %s:\n",utente.nome,utente.cognome);
for (i=0; i<indiceElenco; i++)
  if (compatibili(utente,elenco[i]))
  {
        fwrite(&elenco[i], sizeof(Persona), 1, fbin);
        printf("%s %s nato(a) il %d/%d/%d\n",
              elenco[i].nome,
              elenco[i].cognome,
              elenco[i].nascita.giorno,
              elenco[i].nascita.mese,
              elenco[i].nascita.anno);
  }
fclose(fbin);
}
```

Esercizio 9

(file)

Gestione di una libreria

- Realizzare un programma di gestione per una libreria
- Il programma deve acquisire da standard input una serie di *libri* (con numero massimo fissato a priori)
  - NOTA: si supponga che l'acquisizione avvenga per codici ordinati in senso crescente
- Il programma deve successivamente presentare un menu a tre voci:
  - Stampa a video dell'insieme di libri
  - Ricerca e stampa di un libro a partire dal codice
    - Si usi a tal scopo la RICERCA BINARIA (vedi NOTA)
  - Uscita dal programma
- Estendere poi il programma supportando il salvataggio/caricamento dei libri su/da file binario

Esercizio 9
(file)

- Ogni libro è caratterizzato da
  - Un codice (intero)
  - Un titolo (stringa)
  - Un autore
    - Un autore è descritto da nome e cognome (stringhe)
  - Il numero di copie presenti in libreria (intero)
- Si suddivida l'acquisizione in sottofunzioni
  - Acquisizione del vettore di libri
    - Acquisizione del singolo libro
    - Acquisizione del singolo autore

Esercizio 9
(file)

Estensioni della libreria (utile esercizio “per casa”)

- Un’ulteriore voce di menu per la gestione dei libri in esaurimento
  - In questo caso all’utente viene chiesto un valore soglia
  - Il programma deve scrivere su un file di testo il codice e il numero di copie di tutti i libri il cui numero di copie è inferiore alla soglia data
- Fare in modo che tale elenco sia ***ordinato in senso crescente*** rispetto al numero di copie presenti
  - Alternativa 1: il vettore risultato viene riempito usando l’inserimento ordinato
  - Alternativa 2: riempimento del vettore e poi uso di un algoritmo di ordinamento (modificando l’operatore di confronto)

Esercizio 9 - Soluzione

(file)

```
typedef enum {CRITICAL_ERROR_LOAD, NOT_ENOUGH_SPACE, LOAD_OK}
       LoadResult;
typedef enum{CRITICAL_ERROR_SAVE, SAVE_OK} SaveResult;
typedef enum{NOT_FOUNDED, FOUNDED} FindResult;

typedef struct authorStruct
{
   char name[20];
   char surname[20];
} Author;
typedef struct bookStruct
{
   int code;
   char title[20];
   Author auth;
   int copies;
} Book;
typedef Book BookShop[MAXDIM];
```

Esercizio 9 - Soluzione

(file)

```
Author loadAuthor() {
   Author a;
   printf("author name: "); scanf("%s", a.name);
   printf("author surname: "); scanf("%s", a.surname);
   return a;
}

Book loadBook() {
   Book b;
   printf("book ID: "); scanf("%d", &b.code);
   printf("book title: "); scanf("%s", b.title);
   b.auth = loadAuthor();
   printf("copies number: "); scanf("%d", &b.copies);
   return b;
}
```

Esercizio 9 - Soluzione

(file)

```
LoadResult loadBookShop(BookShop shop, int* num, int maxdim)
{
   int i = 0;
   do
   {
        if(i > maxdim)
               return NOT_ENOUGH_SPACE;
        printf("Inserisci un libro\n");
        shop[i] = loadBook();
        i++;
        printf("Altro libro (s/n)? ");
        getchar();
   }
   while(getchar() == 's');
   *num = i;
   return LOAD_OK;
}
```

Esercizio 9 - Soluzione

(file)

```
void printAuthor(Author a)
{
   printf("author: %s %s\n", a.name, a.surname);
}
void printBook(Book b)
{
   printf(”code: %d\n", b.code);
   printf("title: %s\n", b.title);
   printAuthor(b.auth);
   printf("copies: %d\n", b.copies);
   printf("----------------\n");
}
void printBookShop(BookShop shop, int dim)
{
   int i;
   for(i = 0; i < dim; i++)
       printBook(shop[i]);
}
```

Esercizio 9 - Soluzione

(file)

```
int compare(Book b1, Book b2)
{
  return b1.code - b2.code;
}
```

Esercizio 9 - Soluzione

(file)

```
int binarySearch(BookShop shop, int dim, int toSearch)
{
   int midPos = dim / 2;
   Book bookToSearch;
   bookToSearch.code = toSearch;
   if(compare(shop[midPos], bookToSearch) == 0)
        return midPos;
   if(midPos == 0) return INT_MIN;
   if(compare(shop[midPos], bookToSearch) > 0)
        return binarySearch(shop, midPos, toSearch);
   else
   {
        int startPos = midPos + 1;
        return startPos + binarySearch(   &shop[startPos],
                                          dim - startPos,
                                          toSearch);
   }
}
```

Esercizio 9 - Soluzione

(file)

```
FindResult findCode(BookShop shop, int dim)
{
   int code, pos;
   printf("Insert code: ");
   scanf("%d", &code);
   pos = binarySearch(shop, dim, code);
   if(pos < 0)
       return NOT_FOUNDED;
   printBook(shop[pos]);
   return FOUNDED;
}
```

Esercizio 9 - Soluzione

(file)

```
int main()
{
   BookShop shop;
   int loaded;
   char resp;
   LoadResult lRes;
   FindResult fRes;
   lRes = loadBookShop(shop, &loaded, MAXDIM);
   handleLoadResult(lRes);
   if(lRes != OK)
       return 1;
   …
```

Esercizio 9 - Soluzione

(file)

```
    …
    do
    {
        printf("p)\t print\n f)\t find\n e)\t exit\n");
        while(getchar()!=‘\n’);
        resp = getchar();
        switch(resp)
        {
            case 'p':
                printBookShop(shop, loaded); break;
            case 'f':
                fRes = findCode(shop, loaded);
                handleFindResult(fRes); break;
            case 'e': break;
        }
    }
    while(resp != 'e');
    return 0;
}
```

Esercizio 9 - Soluzione

(file)

```
typedef enum{CRITICAL_ERROR_SAVE, SAVE_OK} SaveResult;

SaveResult saveBookShop(BookShop shop, int dim, char* fileName)
{
   int i;
   FILE *f = fopen(fileName, "wb");
   if(f == NULL)
       return CRITICAL_ERROR_SAVE;
   for(i = 0; i < dim; i++)
       fwrite(&shop[i], sizeof(book), 1, f);
   fclose(f);
}
```

Esercizio 9 - Soluzione

(file)

```
typedef enum{CRITICAL_ERROR, NOT_ENOUGH_SPACE, OK} LoadResult;

loadResult loadBookShopFromFile( BookShop shop, int *dim,
                                 char* fileName, int maxdim)
{
   int i = 0, result;
   FILE *f = fopen(fileName, "r");
   if(f == NULL)
       return CRITICAL_ERROR_LOAD;
   do
   {
       if (i > maxdim)
              return NOT_ENOUGH_SPACE;
       result = fread(&shop[i], sizeof(book), 1, f);
       i++;
   }
   while(result>0);
   *dim = i;
   return OK;
}
```

Esercizio 10
(file)

Conta parole

- Si scriva un programma C che conti il numero dei ***caratteri***, delle ***parole*** e delle ***linee*** contenute in un file di testo di nome specificato e uguale a **PAROLE.TXT**

Esercizio 10 - Soluzione

(file)

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

int main (void)
{
   char nomefile[13];
   FILE *fp;
   int caratteri = 0;
   int linee = 0;
   int parole = 0;
   char ch, prec = ' ';

   printf("Immetti il nome del file: ");
   scanf("%12s", nomefile);

   if ((fp = fopen(nomefile, "r")) == NULL) {
       printf("Errore in apertura in lettura del file %s!\n",
              nomefile);
       exit(1);
   }

   ...
```

Esercizio 10 - Soluzione

(file)

```
  ...
  while (fscanf(fp, "%c", &ch) == 1)
  {
      caratteri++;
      if (ch == '\n')
            linee++;
      if (isspace(prec) && !isspace(ch))
            parole++;
      prec = ch;
  }

  fclose(fp);

  printf("Il numero di caratteri e' %d.\n",
  caratteri);
  printf("Il numero di parole e' %d.\n", parole);
  printf("Il numero di linee e' %d.\n", linee);

  return 0;
}
```