Seconda Esercitazione

Argomenti che verranno trattati in questa e nella prossima esercitazione:

- matrici
- stringhe
- file (lettura/scrittura, di testo/binari)
- strutture, puntatori
- allocazione dinamica della memoria (malloc)
- liste, pile, code

Matrici

Realizzare una funzione **sudoku()** che, data in ingresso una matrice 9x9 di interi, restituisca 0 se la matrice non rappresenta un sudoku, 1 se la matrice rappresenta un sudoku. Inoltre tale funzione deve restituire il numero di valori errati.

5	6	7	2	4	3	1	8	9
3	9	2	8	1	6	7	4	5
4	8	1	5	7	9	6	2	3
8	2	5	3	6	7	4	9	1
9	3	6	1	5	4	8	7	2
7	1	4	9	8	2	3	5	6
1	7	8	6	2	5	9	3	4
6	5	3	4	9	8	2	1	7
2	4	9	7	3	1	5	6	8

Matrici

Una matrice 9x9 è un sudoku se:

1. ogni riga contiene tutte le cifre da 1 a 9;
2. ogni colonna contiene tutte le cifre da 1 a 9;
3. suddividendo regolarmente la matrice 9x9 in 9 matrici 3x3, ogni sottomatrice contiene tutte le cifre da 1 a 9

E' necessario tenere traccia del numero totale di valori che non rispettano i requisiti (tale valore verrà poi restituito per riferimento alla funzione chiamante): se non ci sono errori la funzione restituisce 1, altrimenti 0

```
int sudoku(int matrice[9][9],int *count_err);
```

Matrici

1) per ciascuna riga tenere traccia delle cifre che sono presenti nelle colonne (scansione da sinistra a destra)
2) per ciascuna colonna tenere traccia delle cifre che sono presenti nelle righe (scansione dall'alto al basso)

In entrambi i casi se una cifra appare più di una volta allora è presente un errore

Matrici

```
int checkRows(int matrice[9][9], int *count_err) {
 int check[9],riga,colonna, temp;

 temp = *count_err;
 checkInit(check,9); // elementi di check inizializzati a 0

 // controllo ogni riga
 for (riga=0; riga<9; riga++){
      for (colonna=0; colonna<9; colonna++){
           if (check[matrice[riga][colonna]-1]!=0)
                (*count_err)++; // errore!
           check[matrice[riga][colonna]-1]++;
      }
      checkInit(check,9);
 }

 if(*count_err > temp)
      return 0;
 else
      return 1;
}
```

Matrici

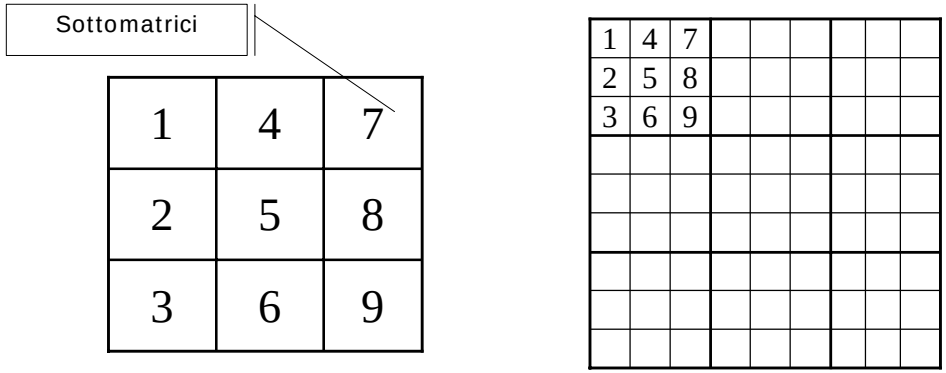
```
int checkColumns(int matrice[9][9], int *count_err) {
 int check[9],riga,colonna, temp;

 temp = *count_err;
 checkInit(check,9); // elementi di check a 0

 for (colonna=0; colonna<9; colonna++){
      for (riga=0; riga<9; riga++){
           if (check[matrice[riga][colonna]-1]!=0)
                (*count_err)++; // errore!
           check[matrice[riga][colonna]-1]++;
      }
      checkInit(check,9);
 }

 if(*count_err > temp)
      return 0;
 else
      return 1;
}
```

Matrici

3) per ogni sottomatrice 3x3 tenere traccia delle cifre che sono presenti in ogni riga e colonna

controllare le sottomatrici nell'ordine indicato in figura a sinistra

per ogni sottomatrice controllare ogni elemento nell'ordine indicato in figura a destra



Matrici

```
int checkSubMatrix(int matrice[9][9], int * count_err) {
 int check[9],sub,c_ini,r_ini,i, temp;

 temp = *count_err;
 checkInit(check,9); // elementi di check a 0

 // controllo ogni sotto matrice
 for (sub=0;sub<9;sub++) {
      r_ini=(sub%3)*3; // prima riga della sottomatrice
      c_ini=(sub/3)*3; // prima colonna della sottomatrice

      // controllo ogni elemento della sottomatrice in esame
      for (i=0;i<9;i++){
           if (check[matrice[r_ini+i%3][c_ini+i/3]-1]!=0)
                (*count_err)++;
           check[matrice[r_ini+i%3][c_ini+i/3]-1]++;
      }
      checkInit(check,9);
 }
 if(*count_err > temp) return 0;
 else return 1; }
```

Matrici

```
int sudoku(int matrice[9][9],int *count_err){

     int result1, result2, result3;
     *count_err=0;

     // controllo ogni riga
     result1 = checkRows(matrice, count_err);

     // controllo ogni colonna
     result2 = checkColumns(matrice, count_err);

     // controllo ogni sotto-matrice
     result3 = checkSubMatrix(matrice, count_err);

     return result1 && result2 && result3;
}

void checkInit(int m[], int length){
   int i;
   for (i=0;i<length;i++) m[i]=0;
}
```

Matrici

```
int main(){
  int err,res;
  int matrix[9][9]={
    {5,6,7,2,4,3,1,8,9},
    {3,9,2,8,1,6,7,4,5},
    {4,8,1,5,7,9,6,2,3},
    {8,2,5,3,6,7,4,9,1},
    {9,3,6,1,5,4,8,7,2},
    {7,1,4,9,8,2,3,5,6},
    {1,7,8,6,2,5,9,3,4},
    {6,5,3,4,9,8,2,1,7},
    {2,4,9,7,3,1,5,6,8}
  };
  res=sudoku(matrix,&err);
  if(res)
       printf("La matrice rappresenta un sudoku\n");
  else
       printf("Sono stati individuati %d errori\n",err);
  return 0;
}
```

Vettori di stringhe

Realizzare una funzione **piuLunga()** che, dato in ingresso un array di stringhe **ben formate** e la lunghezza di tale array, restituisca la stringa più lunga. A tal fine non si faccia uso delle funzioni della libreria standard <string.h>.

Vettori di stringhe

```
#include <stdio.h>

char* piuLunga(char** vettore, int dim){
       int i, lungh, piuLungh=0;
       char* result=NULL;

       for(i=0;i<dim;i++){
            lungh=0;
            while(vettore[i][lungh] != '\0')
                 lungh++;
            if(lungh > piuLungh){
                 result=vettore[i];
                 piuLungh=lungh;
            }
       }

       return result;
}
```

Vettori di stringhe

```
#define DIM 5

int main(){
  int i; char* res; int L;
  char* v[DIM];

  printf("Massima lunghezza parole?\n"); scanf("%d",&L);

  for(i=0;i<DIM;i++){
       v[i]=(char*)malloc(sizeof(char)*(L+1));
       printf("inserisci %d\n",i);
       scanf("%s",v[i]);
  }

  res=piuLunga(v,DIM);
  if(res!=NULL) printf("Piu Lunga %s\n",res);
  else printf("elenco vuoto\n");

  return 0;
}
```

File di testo/binario

Scrivere i numeri da 0 a 9 seguiti da uno spazio in un file di testo ed in uno binario: quali differenze?

```
int main(){
     int i; FILE* fp;

     fp=fopen("file.txt","wt");
     for(i=0;i<10;i++)
       fprintf(fp,"%d ",i);
     fclose(fp);

     fp=fopen("file.bin","wb");
     for(i=0;i<10;i++)
       fwrite(&i,sizeof(int),1,fp);
     fclose(fp);

     return 0;
}
```

	rappresentazione esadecimale	rappresentazione a carattere ASCII
.txt	30 20 31 20 32 20 **33** 20 34 20 35 20 36 20 37 20 38 20 39 20	0 1 2 **3** 4 5 6 7 8 9
.bin	00 00 01 00 02 00 **03 00** 04 00 05 00 06 00 07 00 08 00 09 00	?? ? ? ? ? ? ? ? ? ?

Es. 1

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);

Es. 1

Sia dato un secondo file binario “indirizzi.bin” che contiene, invece, l’indirizzo di ogni studente, e in particolare:

- Matricola: il numero di matricola dello studente;
- Nome: il nome dello studente;
- Cognome: il cognome dello studente;
- Via: una stringa che riporta la via di residenza dello studente;
- Città: una stringa che riporta la città di residenza dello studente;
- CAP: un intero che rappresenta il codice di avviamento postale dello studente.

Es. 1

Si scriva un programma in linguaggio C che:

1. A partire dai file "dati.txt" e “indirizzi.bin” 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.

Es. 1

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.

Es. 1 - Soluzione

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

typedef struct {
    unsigned int matr;
    char nome[20];
    char cognome[30];
    char via[30];
    char citta[30];
    unsigned int CAP;
} indirizzo;
```

```
typedef struct {
     unsigned int matr;
     char nome[20];
     char cognome[30];
     char via[30];
     char citta[30];
     unsigned int CAP;
     unsigned int CDL;
} elemento;

typedef elemento tabella[10];
```

Es. 1 - Soluzione

```
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;
```

Es. 1 - Soluzione

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

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

    while (fscanf(f1,"%u%u", &D.matr, &D.CDL)>0) {
         trovato=0;
         rewind(f2);
         while(fread(&I,sizeof(indirizzo),1,f2)>0 && !trovato)
              if (I.matr==D.matr) {
                   trovato=1;
                   E=riempiel(D, I);
                   T[ins]=E;
                   ins++;
              }
    }

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

Es. 1 - Soluzione

```
    /*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;
}
```

Es. 2

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.

Es. 2

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
```

Es. 2

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.

Es. 2 - soluzione

```
#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); }
}
```

Es. 2

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.

Es. 2 - soluzione

```
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);
   ...
```

Es. 2 - soluzione

```
  ...
  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);
}
```

Es. 3

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.

Es. 3

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
```

Es. 3

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 (6 punti).

Es. 3

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 provveda poi a chiedere 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.

Es. 3 - Soluzione

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

#define DIM 100

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

Es. 3 - Soluzione

```
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);
   }
}
```

Es. 3 - Soluzione

```
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);
   ...
```

Es. 3 - Soluzione

```
   ...
   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);
}
```

Es. 4

Una società di telefonia cellulare gestisce un programma di premiazione per "utenti fedeli". In particolare, per ogni cliente viene salvato su un file binario "**punti.dat**" il nome del cliente (al massimo 31 caratteri) e un numero intero che rappresenta i punti accumulati. Tali informazioni sono organizzate come una struttura **user**, opportunamente definita dal candidato.

```
#define DIM 32

typedef struct {
   char name[DIM];
   int points;
} user;
```

Es. 4

1) Si scriva una funzione:

int readPoints (char usersFile[], user results[], int maxDim, int minPoints)

che, ricevuto in ingresso il nome di un file **usersFile**, un array **results** di strutture **user**, la dimensione massima dell'array **maxDim**, e un limite inferiore di punti **minPoints**, copi nell'array **results** i dati dei clienti che hanno almeno i punti specificati da **minPoints**.

La funzione deve restituire come risultato il numero di utenti con almeno **minPoints**; si noti che tale risultato rappresenta anche la dimensione logica dell'array **results**. Qualora il file non sia accessibile, la funzione deve restituire il valore -1

Es. 4

2) Si scriva poi un programma **main()** che chieda all'utente il numero di clienti salvati sul file (tale numero sarà noto solo a tempo di esecuzione), e allochi dinamicamente un vettore **V** di **user** sufficientemente grande per poter contenere, nel caso peggiore, i dati di tutti gli utenti salvati in **usersFile**. Il programma dovrà poi chiedere all'utente il minimo punteggio e, utilizzando la funzione **readPoints()**, leggere da file e memorizzare in **V** i dati degli utenti che hanno almeno il punteggio minimo specificato. Il programma infine deve stampare a video il nome ed il punteggio degli utenti contenuti in **V** se e solo se il nome comincia per "Me"

Il file contiene una quantità indefinita di informazioni:
non è possibile contenerle tutte in un array di dimensione fissata a priori
→ malloc

Es. 4

```
int readPoints (char usersFile[], user results[], int
   maxDim, int minPoints){

   FILE * f; int logicDim = 0;

   f = fopen(usersFile, "rb");
   if (f == NULL) return -1;

   while(logicDim<maxDim &&
     fread( &results[logicDim], sizeof(user), 1, f) > 0)) {
         if (results[logicDim].points >= minPoints)
              logicDim++;
   }

   fclose(f);
   return logicDim;
}
```

Es. 4

```
int main() {
   user * V; int i, maxUtenti; int logicDim, minPoints;

   printf("Inserire numero massimo di utenti da leggere: ");
   scanf("%d", &maxUtenti);
   V = (user *) malloc(sizeof(user) * maxUtenti);

   printf("Inserire punteggio minimo: ");
   scanf("%d", &minPoints);

   logicDim = readPoints("punti.dat", V, maxUtenti, minPoints);
   if (logicDim < 0) exit(-1);

   for (i=0; i<logicDim; i++)
      if ((V[i].name[0] == 'M') && (V[i].name[1] == 'e'))
         printf("L'utente %s ha %d punti.\n", V[i].name,
                                                V[i].points);
   free(V);
   return 0;
}
```

Es. 5

Un negozio di noleggio CD registra, tramite un PC collegato al registratore di cassa, i dati relativi al noleggio dei Compact Disc. Per ogni utente che restituisce un disco, su un file di testo di nome "**RentedLog.txt**" viene scritto su ogni riga, in ordine:

- un intero **cd_code**, identificativo univoco di un cd;
- una stringa, contenente il nome del cliente (al più 64 caratteri, senza spazi);
- un intero **days**, che indica la durata in giorni del noleggio.

Dopo aver definito opportunamente una struttura **rent** per contenere tali informazioni, il candidato realizzi un programma che chieda all'utente il nome di un cliente e il numero massimo di record che si vogliono ottenere, e stampi a video la lista dei CD noleggiati dal cliente, subito seguito dalla durata media di un noleggio per tale cliente.

```
#define DIM 65
typedef struct {
      int cd_code;
      char renter[DIM];
      int days;
} rent;
```

Es. 5

1. Il candidato scriva una funzione **readRented(…)** che riceve in ingresso il nome di un file di testo, il nome di un utente, un puntatore a strutture **rent** (che punta ad un'area di memoria opportunamente allocata in precedenza) e la dimensione massima di tale area di memoria (in termini di numero di strutture di tipo **rent**). La funzione apra il file e salvi in memoria (tramite il puntatore ricevuto come parametro) i record relativi all'utente specificato (per controllare se un record è relativo al cliente specificato, si utilizzi la funzione **strcmp(…)**). La funzione restituisca il numero di record effettivamente letti, che deve risultare minore o uguale alla dimensione massima specificata. Qualora si raggiunga la dimensione massima di record letti prima di aver terminato il file, si ignorino i record rimanenti.

Es. 5

```
int readRented(char * fileName, char * name, rent * stat,
   int maxDim) {
   int dim = 0; FILE * f;

   if ( (f = fopen(fileName, "r")) == NULL ) {
        printf("Error opening the file %s\n", fileName);
        exit(-1);
   }

   while ( fscanf(f, "%d %s %d", &(stat[dim].cd_code),
       stat[dim].renter, &(stat[dim].days)) != EOF
                              && dim<maxDim) {
         if (strcmp(stat[dim].renter, name) == 0)
             dim = dim + 1;
   }

   fclose(f);
   return dim;
}
```

Es. 5

2. Il candidato realizzi poi un programma C che chieda inizialmente all'utente il nome di un cliente ed il numero massimo di elementi su cui si vuole effettuare la statistica. Dopo aver allocato dinamicamente memoria sufficiente secondo le istruzioni ricevute dall'utente, il programma utilizzi la funzione **readRented(…)** per ottenere i dati relativi al determinato cliente. Si stampi a video poi, in ordine, per ogni CD noleggiato, il nome del cliente, il codice del CD e la durata del noleggio. Si stampi infine la durata media del noleggio.

Es. 5

```
int main() {
   char userName[DIM]; int maxDim = 0; int realDim = 0;
   int i; float total = 0; rent * stat;

   printf("Inserire nome e dimensione massima: ");
   scanf("%s %d", userName, &maxDim);
   stat = (rent *) malloc(sizeof(rent) * maxDim);

   realDim = readRented("RentedLog.txt",userName,stat,maxDim);

   for (i=0; i< realDim; i++) {
         printf("User: %s, CD: %d, Rented for: %d days\n",
            stat[i].renter, stat[i].cd_code, stat[i].days);
         total = total + stat[i].days;
   }

printf("\nAverage length of a rent: %6.2f\n\n",total/realDim);

   free(stat);
   return 0;
}
```

Es6: ADT e allocazione dinamica della memoria

Si vuole realizzare un programma per eseguire calcoli elementari su numeri complessi. A tal scopo si definisca una opportuna struttura dati per rappresentare i numeri complessi, con relative operazioni di **somma** e **sottrazione**.

Si definiscano poi due funzioni, una per *stampare* ed una per *leggere* numeri complessi, che ricevano come parametro d'ingresso almeno un puntatore a FILE.

Es6: ADT e allocazione dinamica della memoria
ADT **complex_number**

```
struct complex {
  float re;
  float im;
};
typedef struct complex complex_number;

complex_number create(float real, float imm) {
  complex_number result;
  result.re = real;
  result.im = imm;
  return result;
}

float getReal(const complex_number num) {
  return num.re;
}

float getIm(const complex_number num) {
  return num.im;
}
```

Es6: ADT e allocazione dinamica della memoria
ADT **complex_number**

```
complex_number sum(   const complex_number a,
                      const complex_number b) {
 complex_number result;
 result = create(getReal(a)+getReal(b), getIm(a)+getIm(b));
 return result;
}

complex_number dif(   const complex_number a,
                      const complex_number b) {
 complex_number result;
 result = create(getReal(a)-getReal(b), getIm(a)-getIm(b));
 return result;
}


void printComplexNumber(const complex_number a, FILE * f) {
 fprintf(f, "%f:%f\n", getReal(a), getIm(a));
}
```

Es6: ADT e allocazione dinamica della memoria
ADT **complex_number**

```
int scanComplexNumber(FILE * f, complex_number * result) {
 int val;
 float real;
 float imm;

 val = fscanf(f, "%f:%f", &real, &imm);

 if (val != EOF) {
   *result = create(real, imm);
   return 0;
 }
 else
   return 1;

}
```

Es6: ADT e allocazione dinamica della memoria
ADT **complex_number**

Si scriva poi un programma che legge da stdin un numero intero N; il programma provvederà poi a chiedere all'utente N numeri complessi che saranno salvati temporaneamente in un array.

Il programma chieda infine il nome di un file di testo all'utente, e provveda a scrivere su tale file i numeri complessi inseriti.

Si discuta infine eventuali varianti nel caso in cui invece di leggere i numeri complessi da stdin, si voglia utilizzare un file contenente un numero imprecisato di elementi.

Es6: ADT e allocazione dinamica della memoria
ADT **complex_number**

```
int main() {

  int dim, i, j, result;
  complex_number * V;
  complex_number temp;
  char nomefile[MAX_DIM];
  FILE * f;

  printf("Inserire dimensione vettore: ");
  scanf("%d", &dim);

  V = (complex_number*) malloc(sizeof(complex_number) * dim);
  for (i=0; i<dim; i++) {
    printf("Inserire parte reale ed immaginaria, xxx:yyy : ");
    result = scanComplexNumber(stdin, &temp);
    if (result == 0)
         V[i] = temp;
    else
         V[i] = create(0,0);
  }
```

Es6: ADT e allocazione dinamica della memoria
ADT **complex_number**

```
  ...

  printf("Nome file su cui salvare: ");
  scanf("%s", nomefile);

  if ((f=fopen(nomefile, "w")) == NULL) {
    printf("Errore durante l'apertura del file %s.", nomefile);
    exit(-1);
  }

  for (i=0; i<dim; i++)
    printComplexNumber(V[i], f);

  fclose(f);
  free(V);
  return 0;
}
```

Es7: ADT
Astrazione di vettore infinito

Si progetti un programma capace di leggere da stdin un numero (teoricamente) infinito di interi positivi, terminati da uno 0.

A tal scopo si realizzi un ADT che implementi l'idea di un vettore di interi a dimensione infinita, utilizzando apposite strutture e array allocati dinamicamente.

Devono essere inoltre definite funzioni e predicati per:

a) Creare un vettore vuoto

b) Aggiungere un intero in coda al vettore

c) Ottenere la dimensione (logica) del vettore

d) Ottenere un elemento del vettore data la sua posizione

Es7: ADT
Astrazione di vettore infinito

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

#define FIRST_SIZE 5
#define INCREMENT_SIZE 5


typedef struct {
  int p_dim;            /* physical dimension */
  int l_dim;            /* logical dimension */
  int * pointer;        /* pointer to the data */
} array_int;


array_int createEmpty();
int size(array_int a);
void add(int num, array_int * a);
int get(const array_int a, int pos);
```

Es7: ADT
Astrazione di vettore infinito

```
array_int createEmpty() {
  array_int result;
  result.pointer = malloc( sizeof(int) * FIRST_SIZE);
  result.p_dim = FIRST_SIZE;
  result.l_dim = 0;
  return result;
}


int size(array_int a) {
  return a.l_dim;
}
```

Es7: ADT
Astrazione di vettore infinito

```
void add(int num, array_int * a) {
  int * temp;
  int i;

  if (size(*a) < (a->p_dim)) {
    (a->pointer)[size(*a)] = num;
    (a -> l_dim) = (a -> l_dim) + 1;
  }
  else {
    temp = a -> pointer;
    (a->pointer)
         = malloc(sizeof(int) * ((a->p_dim)+INCREMENT_SIZE));

    for (i=0; i<(a->l_dim); i++)
         (a->pointer)[i] = temp[i];

    (a->pointer)[a->l_dim] = num;
    (a -> l_dim) = (a -> l_dim) + 1;
    (a->p_dim)= (a->p_dim)+INCREMENT_SIZE;
    free(temp);
  }
}
```

Es7: ADT
Astrazione di vettore infinito

```
int get(const array_int a, int pos) {
  if (pos < 0)
    pos = 0;
  if (size(a) <= pos)
    pos = size(a) - 1;

  if (size(a) == 0)
    return -1;

  return (a.pointer)[pos];
}
```

Es7: ADT
Astrazione di vettore infinito

```
int main() {

  int i;
  int num;
  array_int V = createEmpty();

  printf("Inserisci i numeri, 0 per terminare: ");
  scanf("%d", &num);
  while (num >0) {
    add( num, &V);
    scanf("%d", &num);
  }

  for (i=0; i<size(V); i++) {
    printf("%d\n", get(V, i));
  }

}
```