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)

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

È 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

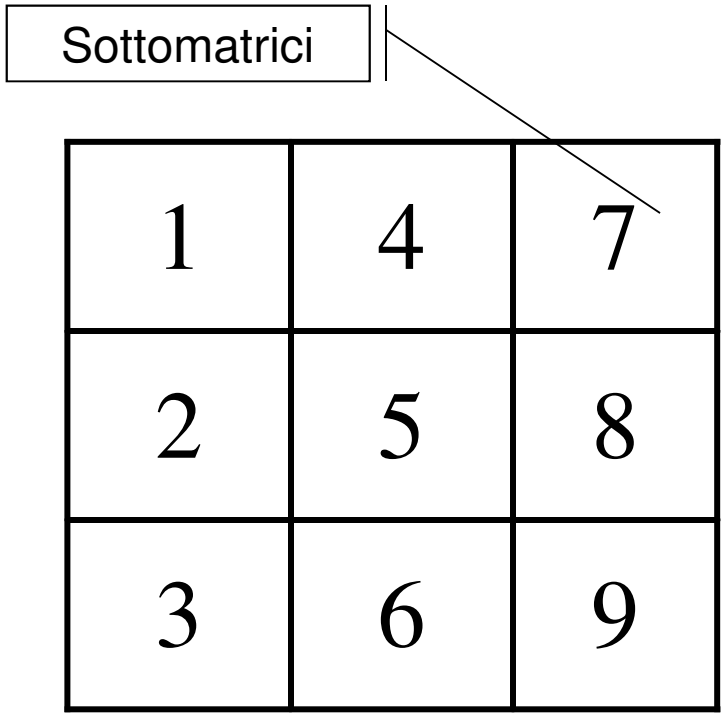
```
int checkColumns(int matrice[9][9], int *count_err) {
  int check[9],riga,colonna, temp;

  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;
}
```

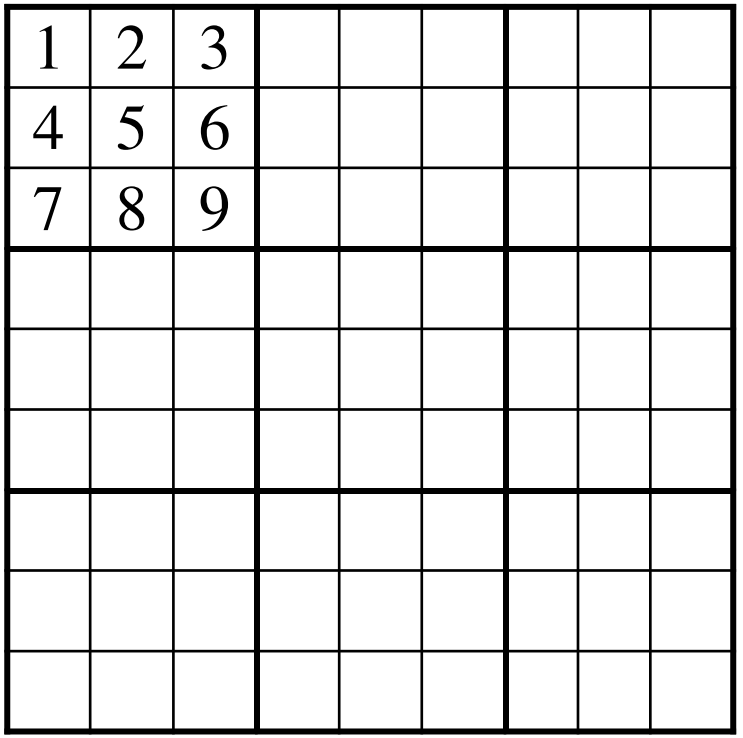
Matrici

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_row,sub_col,c_ini,r_ini,i,j,temp;
  temp = *count_err;

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

        // controllo ogni elemento della sottomatrice in esame
        checkInit(check,9); // elementi di check a 0
        for (i=r_ini;i<r_ini+3;i++){
             for (j=c_ini;j<c_ini+3;j++){
                  if (check[matrice[i][j]-1]!=0)
                       (*count_err)++;
                  check[matrice[i][j]-1]++;
             }
        } } }
  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 e 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

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. 1

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. 1

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. 1

```
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. 1

```
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. 2

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. 2

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. 2

```
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. 2

2) Il candidato realizzi poi un programma C che chieda inizialmente all'utente il nome di un cliente e 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. 2

```
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;
}
```

Es3: 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

Es3: 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;
}
```

Es3: 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));
}
```

Es3: 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;

}
```

Es3: ADT e allocazione dinamica della memoria
ADT **`complex_number`**

Si scriva poi un programma che legge da stdin un numero intero N; il programma chiederà poi 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

Es3: 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);
  }
```

Es3: 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;
}
```

Es4: 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

Es4: 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);
```

Es4: ADT
Astrazione di vettore infinito

```
array_int createEmpty() {
  array_int result;
  result.pointer = (int*) 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;
}
```

Es4: 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)
      =(int*) 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);
  }
}
```

Es4: 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];
}
```

Es4: 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));
  }

}
```