

Esercitazione

Argomenti che verranno trattati in questa 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

Sottomatrici

1	4	7
2	5	8
3	6	9

1	2	3						
4	5	6						
7	8	9						

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** (i.e., correttamente terminate da '\0') 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;
```

```
}
```

Oppure:

```
vettore[i][lungh] != '\0'
```

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 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;
}
```

Esadecimale del byte corrispondente al carattere '3' in ASCII

	rappr. esadecimale	rappr. a carattere ASCII
.txt	3031 32 33 3435 3637 3839	012 <u>3</u> 456789
bin	00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00 09 00	???????? ?? ?

Esadecimale corrispondente al numero decimale 3 (scritto in memoria su 4byte in binario)

Es1: 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 su file ed una per leggere da file i numeri complessi*.

Es1: 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(Complex_number num) {
    return num.re;
}

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


Es1: ADT e allocazione dinamica della memoria

ADT `Complex_number`

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

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

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

Es1: 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;
}
```

Es1: 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

Es1: 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);
    }
}
```

Es1: 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;
}
```

Es2: 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

Es2: 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);
```

Es2: 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;
}
```


Es2: 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);
    }
}
```

Es2: 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];
}
```

Es2: 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));  
    }  
  
}
```