

Esercizio 1: Quadrato magico

- Realizzare una funzione che, presa in input una matrice quadrata, determini se è un quadrato magico
- Un quadrato magico è una matrice NxN
 - I cui elementi sono TUTTI i numeri interi da 1 a N^2in cui
 - Le somme degli elementi per tutte le righe, tutte le colonne e le diagonali sono equivalenti
 - Tale somma è detta “magic constant”

1

Esercizio 1 - Soluzione (1)

```
#define N 3
typedef int matrice[N][N];
typedef enum{false, true} boolean;
typedef boolean covered[N*N];
boolean verifyMatrix(matrice m, int dim)
{
    covered c;
    int i, j;
    for(i = 0; i < dim*dim; i++)
        c[i] = false;
    ...
}
```

Vettore che verifica la presenza, nella matrice, dei numeri da 1 a N^2
Se $c[i]$ è true significa che il numero $i+1$ è presente nella matrice

2

Esercizio 1 - Soluzione (2)

```
...
for(i = 0; i < dim; i++)
{
    for(j = 0; j < dim; j++)
    {
        if(m[i][j] < 1 || m[i][j] > dim*dim)
        {
            return false;
        }
        if( c[ m[i][j] -1] == false)
        {
            c[ m[i][j] -1] = true;
        }
        else
        {
            return false;
        }
    }
}
return true;
}
```

Controllo sull'intervallo dei valori $[1, N^2]$

I numeri non devono essere ripetuti

3

Esercizio 1 - Soluzione (3)

- Essendo il calcolo su righe e colonne molto simile, realizziamo **un'unica funzione sufficientemente generica**
 - La somiglianza è dovuta al fatto che la matrice è quadrata
 - Un ulteriore parametro di questa funzione indica se il conteggio va effettuato sulle righe o sulle colonne

```
boolean verifyRows(matrice m, int dim, int* sum)
{
    return verify(m, dim, true, sum);
}

boolean verifyColumns(matrice m, int dim, int* sum)
{
    return verify(m, dim, false, sum);
}
```

4

Esercizio 1 - Soluzione (4)

```
boolean verify(matrice m, int dim, boolean verifyRows, int* sum)
{
    int i, j, partial;
    *sum = -1;
    for(i = 0; i < dim; i++)
    {
        partial = 0;
        for(j = 0; j < dim; j++)
        {
            if(verifyRows)
                partial = partial + m[i][j];
            else
                partial = partial + m[j][i];
        }
        if(*sum < 0)
            *sum = partial;
        else if (*sum != partial)
            return false;
    }
    return true; }
```

Inizializzazione
della somma

5

Esercizio 1 - Soluzione (5)

```
boolean verifyDiagonals(matrice m, int dim, int* sum)
{
    int i, j, sum2 = 0;
    *sum = 0;
    for(i = 0; i < dim; i++)
        *sum = *sum + m[i][i];
    for(i = 0; i < dim; i++)
        sum2 = sum2 + m[i][dim-i-1];
    return (*sum == sum2);
}
```

6

Esercizio 1 - Soluzione (6)

Definizione dei codici di ritorno della funzione

```
#define RESULT int

#define NOT_WELL_FORMED 0;
#define ROWS_CHECK_FAILED 1;
#define COLUMNS_CHECK_FAILED 2;
#define DIAGONALS_CHECK_FAILED 3;
#define DIFFERENT_SUM 4;
#define MAGIC 5;
```

7

Esercizio 1 - Soluzione (7)

```
RESULT magicSquared( matrice m, int dim,
                    int*magic_constant)
{
    boolean result;
    int sum, sum2;
    result = verifyMatrix(m, dim);
    if(!result)
        return NOT_WELL_FORMED;
    result = verifyRows(m, dim, &sum);
    if(!result)
        return ROWS_CHECK_FAILED;
    result = verifyColumns(m, dim, &sum2);
    if(!result)
        return COLUMNS_CHECK_FAILED;
    ...
}
```

8

Esercizio 1 - Soluzione (8)

```
...
if(sum != sum2)
    return DIFFERENT_SUM;
result = verifyDiagonals(m, dim, &sum2);
if(!result)
    return DIAGONALS_CHECK_FAILED;
if(sum != sum2)
    return DIFFERENT_SUM;
*magic_constant = sum;
return MAGIC;
}
```

9

Esercizio 2: Algoritmi di ordinamento

“Astrazione” degli algoritmi di ordinamento

- Implementare i diversi algoritmi di ordinamento, facendo in modo di **astrarre completamente dal tipo degli elementi del vettore**
- Fare anche in modo che vengano stampate delle **statistiche sul numero di confronti e di scambi** effettuati
- Testare la soluzione su un vettore di interi, un vettore di caratteri, un vettore di stringhe

10

Esercizio 2 - elementdef.h

- Nel caso di interi...

```
#ifndef ELEMENTDEF
#define ELEMENTDEF
    typedef int Element;
#endif
```

- Nel caso di stringhe...

```
#ifndef ELEMENTDEF
#define ELEMENTDEF
    typedef char* Element;
#endif
```

11

Esercizio 2 - element.h

```
#ifndef ELEMENT
#define ELEMENT
    int compare(Element e1, Element e2);
    void swap(Element *e1, Element *e2);
    void assign(Element *lvalue, Element rvalue);
    void printElement(Element e);
    void printStatistics();
#endif
```

12

Esercizio 2 - element.c (1)

```
#include "elementdef.h"
#include <stdio.h>
#include <string.h> //se il tipo è stringa...

int compareCounter = 0;
int swapCounter = 0;
void incrementCompareCounter()
{
    compareCounter++;
}

void incrementSwapCounter()
{
    swapCounter++;
}
```

13

Esercizio 2 - element.c (2)

```
void assign(Element *lvalue, Element rvalue)
{
    *lvalue = rvalue;
}

void swap(Element *e1, Element *e2)
{
    Element tmp;
    assign(&tmp, *e1);
    assign(e1, *e2);
    assign(e2, tmp);
    incrementSwapCounter();
}
```

14

Esercizio 2 - element.c (3)

- Nel caso di interi...

```
int compare(Element e1, Element e2)
{
    incrementCompareCounter();
    return e1 - e2;
}

void printElement(Element e)
{
    printf("%d\n", e);
}
```
- Nel caso di stringhe...

```
int compare(Element e1, Element e2)
{
    incrementCompareCounter();
    return strcmp(e1, e2);
}

// e ovviamente printElement con %s
```

15

Esercizio 2 - altri header

- commondef.h

```
#ifndef COMMONDEF
#define COMMONDEF
#define DIM 20
    typedef Element Array[DIM];
#endif
```
- sort.h

```
void printArray(Array a, int dim);
void naiveSort(Array a, int dim);
// e tutti gli altri tipi di sort...
```

16

Esercizio 2 – ad esempio printArray

```
#include "elementdef.h"
#include "commondef.h"
#include "element.h"
#include <stdio.h>

void printArray(Array a, int dim)
{
    int i;
    printf("---VETTORE---\n");
    for(i = 0; i < dim; i++)
        printElement(a[i]);
    printf("-----\n");
}
```

17

Esercizio 2 – ad esempio naiveSort

```
void naiveSort(Array a, int dim)
{
    int j, i, posmin;
    Element min;
    for (j = 0; j < dim; j++)
    {
        posmin = j;
        for ( assign(&min, a[j]), i = j + 1; i < dim; i++)
        {
            if(compare(a[i], min) < 0)
            {
                posmin = i;
                assign(&min, a[i]);
            }
        }
        if (posmin != j)
            swap(&a[j], &a[posmin]);
    }
}
```

18