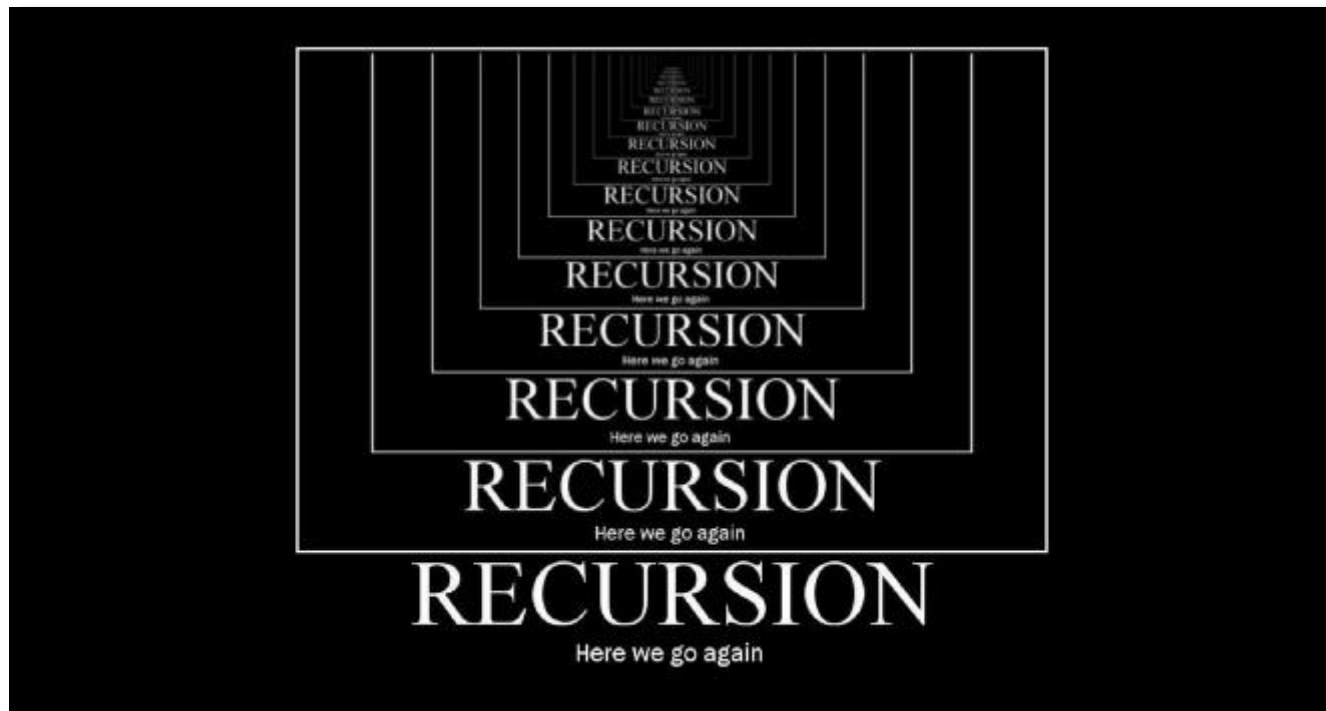


# Ricorsione, Serie di Fibonacci, Bubble sort

Dott. Emanuele Pio Barracchia

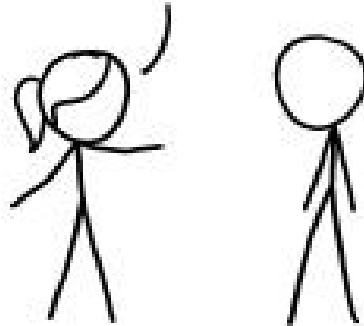
# Ricorsione

- La **ricorsione** in informatica è un metodo che si usa quando la soluzione di un problema dipende dalle soluzioni di piccole istanze dello stesso problema
- La definizione di una funzione ricorsiva ha uno o più **casi base**, cioè input per i quali la funzione produce un risultato senza richiamare se stessa
- Ad ogni esecuzione della funzione ricorsiva il problema in input è semplificato in modo tale che dopo un numero di passi riuscirà ad arrivare ad un **caso base**



# Ricorsione

I MET A TRAVELER FROM AN ANTIQUE LAND  
WHO SAID: "I MET A TRAVELER FROM AN AN-  
TIQUE LAND, WHO SAID: "I MET A TRAVELER FROM  
AN ANTIQUE LAND, WHO SAID: "I MET...



# Approccio ricorsivo Vs Approccio iterativo

- Per ogni soluzione ricorsiva esiste la corrispondente soluzione iterativa
- La ricorsione occupa più spazio in memoria ed è più lenta rispetto all'approccio iterativo
- Per alcuni problemi le soluzioni ricorsive sono spesso più semplici, più leggibili e più eleganti rispetto alle soluzioni iterative

# Un esempio: il fattoriale di un numero

**$n!$**

- In matematica, si definisce fattoriale di un numero naturale il prodotto dei numeri interi positivi minori o uguali a tale numero.
- Inoltre il fattoriale di 0 è pari a 1.
- **Esercizio:** scrivere una funzione ricorsiva in C che, dato un numero, calcoli il fattoriale.
  - Tip: prima di tutto pensate a quale possa essere il *caso base*.

# Un esempio: il fattoriale di un numero

## Senza puntatore

```
long factorial(long n)
{
    if (n == 0) //passo base
        return 1;
    else
        return n*factorial(n-1);
}
```

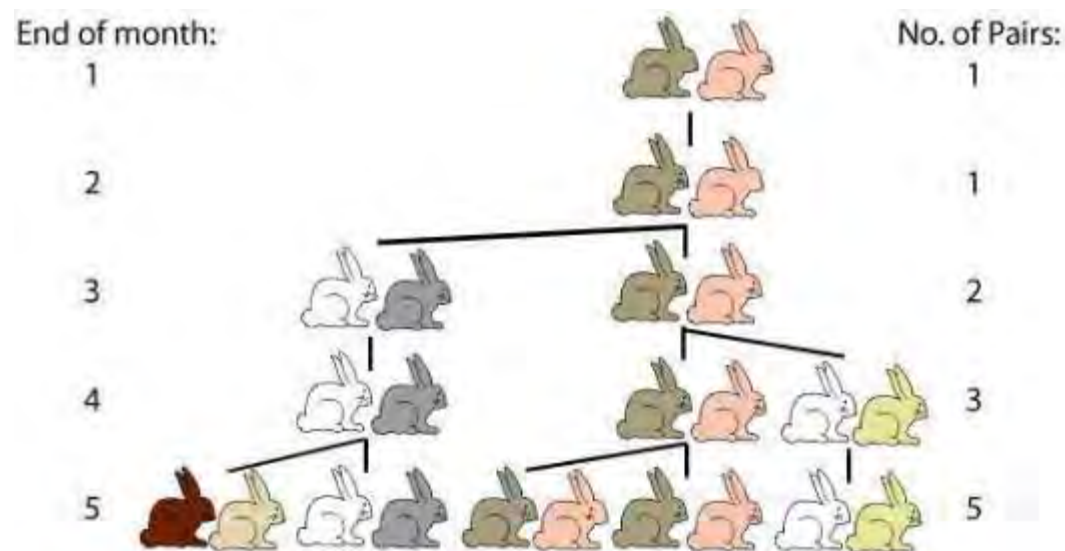
## Con puntatore

```
long fact_point(long* pointer)
{
    if (*pointer == 0){
        return 1;
    }
    else if (*pointer == 1){ //passo base
        return *pointer;
    }
    else
    {
        long tmp = *pointer - 1;
        fact_point(&tmp);
        return *pointer *= tmp;
    }
}
```

# Successione di Fibonacci - Origini

Fibonacci cercò un modo per descrivere la crescita di una popolazione di conigli. In particolare assunse per ipotesi che:

- Si dispone di una coppia di conigli appena nati
- Ogni coppia diventa fertile dopo un mese e dà alla luce una nuova coppia di conigli dopo un ulteriore mese
- Le coppie fertili danno alla luce una coppia di figli al mese



# Successione di Fibonacci

- In matematica, la successione di Fibonacci è una successione di numeri interi positivi in cui ciascun numero è la somma dei due numeri precedenti.
- I primi due termini della successione sono, per definizione,  $F_1 = 1$  e  $F_2 = 1$
- La regola è la seguente:
  - $F_1 = 1$
  - $F_2 = 1$
  - $F_n = F_{n-1} + F_{n-2}$  (per ogni  $n > 2$ )
- **Esercizio:** scrivere una funzione ricorsiva in C che, dato un numero  $i$ , calcoli l' $i$ -esimo numero della successione di Fibonacci.
  - Tip: prima di tutto pensate a quale possa essere il *caso base*.

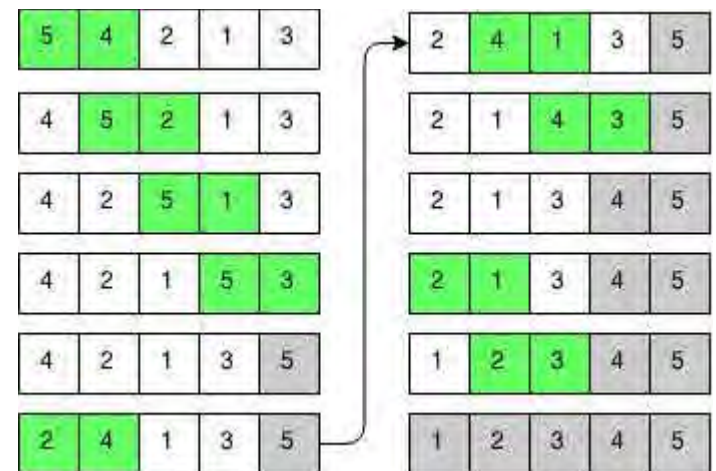


# Successione di Fibonacci

```
long fibonacci(long n){  
    //inizio passo base  
    if (n == 0){  
        return 0;  
    } else if (n == 1){  
        return 1;  
    }  
    //fine passo base  
    else {  
        return fibonacci(n-1) + fibonacci(n-2);  
    }  
  
    return 0;  
}
```

# Bubble sort

- In informatica il **Bubble sort**, o ordinamento a bolle, è un algoritmo di ordinamento che confronta ogni coppia di elementi adiacenti e li inverte nel caso in cui siano nell'ordine sbagliato.
- Il nome dell'algoritmo deriva dal modo in cui gli elementi vengono ordinati in questa lista: quelli più **“leggeri”** **“risalgono”** verso un'estremità, mentre quelli più **“pesanti”** **“affondano”** verso l'estremità opposta.



# Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){  
    int i = 0;  
    int j = 0;  
    int sorted = 0; //sorted = false  
  
    while (i < size && sorted == 0){  
        sorted = 1;  
        i++;  
        for (j = 0; j < size-i; j++){  
            if (array[j] > array[j+1]){  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1] = temp;  
                sorted = 0;  
            }  
        }  
    }  
}
```

# Bubble sort - Algoritmo

i = 0

j = 0

21	10	3	8	30
----	----	---	---	----

```
void bubblesort(int array[], int size){  
    int i = 0;  
    int j = 0;  
    int sorted = 0; //sorted = false  
  
    while (i < size && sorted == 0){  
        sorted = 1;  
        i++;  
        for (j = 0; j < size-i; j++){  
            if (array[j] > array[j+1]){  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1] = temp;  
                sorted = 0;  
            }  
        }  
    }  
}
```

# Bubble sort - Algoritmo

**i = 1**

**j = 0**

21	10	3	8	30
----	----	---	---	----

```
void bubblesort(int array[], int size){  
    int i = 0;  
    int j = 0;  
    int sorted = 0; //sorted = false  
  
    while (i < size && sorted == 0){  
        sorted = 1;  
        i++;  
        for (j = 0; j < size-i; j++){  
            if (array[j] > array[j+1]){  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1] = temp;  
                sorted = 0;  
            }  
        }  
    }  
}
```

# Bubble sort - Algoritmo

i = 1

j = 0

21	10	3	8	30
----	----	---	---	----

```

void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}

```

# Bubble sort - Algoritmo

i = 1

j = 1

10	21	3	8	30
----	----	---	---	----

```

void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}

```

# Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){  
    int i = 0;  
    int j = 0;  
    int sorted = 0; //sorted = false  
  
    while (i < size && sorted == 0){  
        sorted = 1;  
        i++;  
        for (j = 0; j < size-i; j++){  
            if (array[j] > array[j+1]){  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1] = temp;  
                sorted = 0;  
            }  
        }  
    }  
}
```

i = 1

j = 1

10	21	3	8	30
----	----	---	---	----



# Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){  
    int i = 0;  
    int j = 0;  
    int sorted = 0; //sorted = false  
  
    while (i < size && sorted == 0){  
        sorted = 1;  
        i++;  
        for (j = 0; j < size-i; j++){  
            if (array[j] > array[j+1]){  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1] = temp;  
                sorted = 0;  
            }  
        }  
    }  
}
```

i = 1

j = 2

10	3	21	8	30
----	---	----	---	----

# Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){  
    int i = 0;  
    int j = 0;  
    int sorted = 0; //sorted = false  
  
    while (i < size && sorted == 0){  
        sorted = 1;  
        i++;  
        for (j = 0; j < size-i; j++){  
            if (array[j] > array[j+1]){  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1] = temp;  
                sorted = 0;  
            }  
        }  
    }  
}
```

i = 1

j = 2

10	3	21	8	30
----	---	----	---	----

# Bubble sort - Algoritmo

i = 1

j = 3

10	3	8	21	30
----	---	---	----	----

```
void bubblesort(int array[], int size){  
    int i = 0;  
    int j = 0;  
    int sorted = 0; //sorted = false  
  
    while (i < size && sorted == 0){  
        sorted = 1;  
        i++;  
        for (j = 0; j < size-i; j++){  
            if (array[j] > array[j+1]){  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1] = temp;  
                sorted = 0;  
            }  
        }  
    }  
}
```

# Bubble sort - Algoritmo

i = 1

j = 3

10	3	8	21	30
----	---	---	----	----

```

void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}

```

# Bubble sort - Algoritmo

$i = 2$

$j = 0$

10	3	8	21	30
----	---	---	----	----

```
void bubblesort(int array[], int size){  
    int i = 0;  
    int j = 0;  
    int sorted = 0; //sorted = false  
  
    while (i < size && sorted == 0){  
        sorted = 1;  
        i++;  
        for (j = 0; j < size-i; j++){  
            if (array[j] > array[j+1]){  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1] = temp;  
                sorted = 0;  
            }  
        }  
    }  
}
```

# Bubble sort - Algoritmo

i = 2

j = 0

10	3	8	21	30
----	---	---	----	----

```

void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}

```

# Bubble sort - Algoritmo

i = 2

j = 1

3	10	8	21	30
---	----	---	----	----

```
void bubblesort(int array[], int size){  
    int i = 0;  
    int j = 0;  
    int sorted = 0; //sorted = false  
  
    while (i < size && sorted == 0){  
        sorted = 1;  
        i++;  
        for (j = 0; j < size-i; j++){  
            if (array[j] > array[j+1]){  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1] = temp;  
                sorted = 0;  
            }  
        }  
    }  
}
```





**A FEW  
MOMENTS LATER**



# Bubble sort - Algoritmo

Array iniziale:

21	10	3	8	30
----	----	---	---	----

Array dopo l'esecuzione dell'algoritmo **Bubble sort**:

3	8	10	21	30
---	---	----	----	----

# Bubble sort - Algoritmo

## *Senza puntatore*

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

## *Con puntatore*

```
void bubblesort_pointer (int* pointer, int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (*(pointer+j) > *(pointer+j+1)){
                int temp = *(pointer+j);
                *(pointer+j) = *(pointer+j+1);
                *(pointer+j+1) = temp;
                sorted = 0;
            }
        }
    }
}
```

Domande?

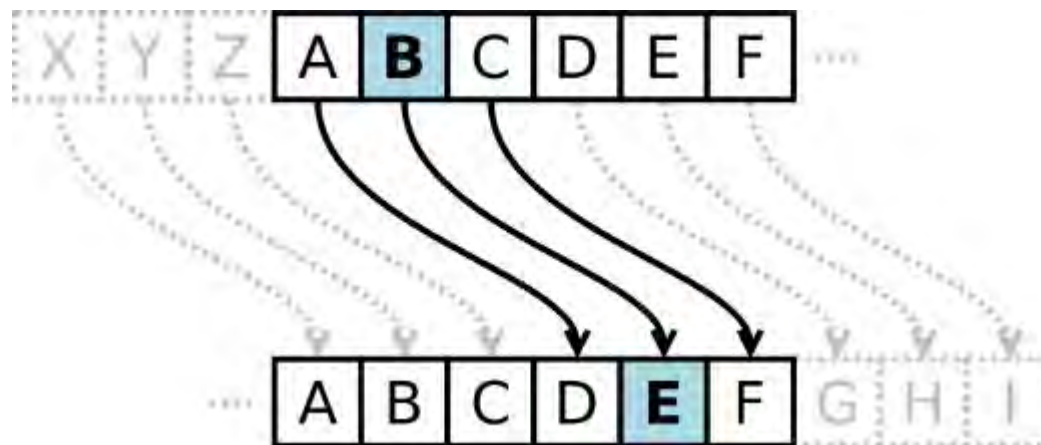
# Cifrario di Cesare, Ricerca all'interno di un array, Generazione numeri casuali

Dott. Emanuele Pio Barracchia

# Cifrario di Cesare



- Il cifrario di Cesare è uno dei più antichi algoritmi crittografici
- Il cifrario di Cesare prende il nome da Giulio Cesare, che lo utilizzava per proteggere i suoi messaggi segreti.
- È un cifrario a sostituzione monoalfabetica in cui ogni lettera del testo in chiaro è sostituita nel testo cifrato dalla lettera che si trova un certo numero di posizioni (*la chiave*) dopo nell'alfabeto. In particolare, Cesare utilizzava 3 come chiave



# Cifrario di Cesare

```
void encrypt(char message[], int key){
    char ch;
    for(int i = 0; message[i] != '\0'; ++i){
        ch = message[i];

        if(ch >= 'a' && ch <= 'z'){
            ch = ch + key;

            if(ch > 'z'){
                ch = ch - 'z' + 'a' - 1;
            }

            message[i] = ch;
        }
        else if(ch >= 'A' && ch <= 'Z'){
            ch = ch + key;

            if(ch > 'Z'){
                ch = ch - 'Z' + 'A' - 1;
            }

            message[i] = ch;
        }
    }
}
```



# Cifrario di Cesare - Esercizio

- Scrivere un metodo in linguaggio C che, ricevendo in input il messaggio da codificare e la chiave, restituisca il messaggio cifrato.
- Cercare di decifrare il seguente messaggio: **fliudulr gl fhvduh** con chiave 3



# Cifrario di Cesare - Esercizio

- Scrivere un metodo in linguaggio C che, ricevendo in input il messaggio da codificare e la chiave, restituisca il messaggio cifrato.

```
void decrypt(char message[], int key){
    char ch;
    for(int i = 0; message[i] != '\0'; ++i){
        ch = message[i];

        if(ch >= 'a' && ch <= 'z'){
            ch = ch - key;

            if(ch < 'a'){
                ch = ch + 'z' - 'a' + 1;
            }

            message[i] = ch;
        }
        else if(ch >= 'A' && ch <= 'Z'){
            ch = ch - key;

            if(ch < 'A'){
                ch = ch + 'Z' - 'A' + 1;
            }

            message[i] = ch;
        }
    }
}
```





# Ricerca all'interno di un array

Esistono due algoritmi principali di ricerca di un elemento all'interno di un array:

- Algoritmo di ricerca sequenziale
- Algoritmo di ricerca dicotomica



# Ricerca sequenziale

- L'algoritmo di ricerca sequenziale (o lineare) è un algoritmo utilizzabile per trovare un elemento in un insieme non ordinato
- Effettua una ricerca all'interno dell'array in modo sequenziale
- In particolare, controlla in sequenza gli elementi dell'array e verifica per ogni elemento controllato se è uguale all'elemento cercato
- L'algoritmo termina se:
  - L'elemento analizzato è l'elemento cercato
  - Ha controllato tutto l'array senza trovare l'elemento cercato



# Ricerca sequenziale

```
void linear_search(int array[], int search){
    int i = 0;
    int dim = sizeof(array)/sizeof(array[0]);

    for (i = 0; i < dim; i++)
    {
        if (array[i] == search)
        {
            printf("%d trovato in posizione %d.\n", search, i+1);
            break;
        }
    }
    if (i == dim)
        printf("%d non è presente nell'array.\n", search);
}
```



# Ricerca sequenziale - Versione ricorsiva

```
#include<stdio.h>

int recSearch(int arr[], int left, int right, int x)
{
    if (right < left)
        return -1;
    if (arr[left] == x)
        return left;
    return recSearch(arr, left+1, right, x);
}

int main()
{
    int search, n;

    printf("Inserire il numero di elementi dell'array: ");
    scanf("%d",&n);
    int array[n];

    printf("Inserire %d numeri: \n", n);

    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);

    printf("Inserire il numero da cercare: ");
    scanf("%d", &search);

    int index = recSearch(array, 0, n-1, search);

    if (index != -1)
        printf("L'elemento %d è presente in posizione %d. \n", search, index);
    else
        printf("L'elemento %d non è presente.", search);
    return 0;
}
```



# Ricerca dicotomica

- L'algoritmo di ricerca dicotomica (o ricerca binaria) è un algoritmo di ricerca che individua l'indice di un determinato valore presente all'interno di un array ordinato
- Effettua mediamente meno confronti rispetto ad una ricerca sequenziale
- Il metodo alla base della ricerca binaria è lo stesso che si utilizza quando si cerca una parola all'interno del dizionario. L'idea è quella di iniziare la ricerca dal primo elemento, ma da quello centrale:
  - Se l'elemento corrisponde a quello cercato, la ricerca termina
  - Se è superiore, la ricerca viene ripetuta sugli elementi precedenti
  - Se è inferiore, la ricerca viene ripetuta su quelli successivi



# Ricerca dicotomica

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```



# Ricerca dicotomica

First = 0

Search = 5

Middle = 2

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 0

Search = 5

Middle = 2

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----





# Ricerca dicotomica

First = 0

Search = 5

Middle = 2

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 0

Search = 5

Middle = 2

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 0

Search = 5

Middle = 2

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 0

Search = 5

Middle = 0

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 0

Search = 5

Middle = 0

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 0

Search = 5

Middle = 0

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 1

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 1

Search = 5

Middle = 1

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----





# Ricerca dicotomica

First = 1

Search = 5

Middle = 1

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 1

Search = 5

Middle = 1

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 1

Search = 5

Middle = 1

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----



# Ricerca dicotomica

First = 1

Search = 5

Middle = 1

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);
}
```

1	5	8	10	12
---	---	---	----	----

5 trovato in posizione 2.



# Ricerca dicotomica - Versione ricorsiva

```
void binary_search_rec(int array[], int low, int high, int search)
{
    int mid;

    if (low > high)
    {
        printf("%d non trovato.\n", search);
        return;
    }
    mid = (low + high) / 2;
    if (array[mid] == search)
    {
        printf("%d trovato in posizione %d.\n", search, mid);
    }
    else if (array[mid] > search)
    {
        binary_search_rec(array, low, mid - 1, search);
    }
    else if (array[mid] < search)
    {
        binary_search_rec(array, mid + 1, high, search);
    }
}
```



## Esercizio - Test primalità

Scrivere un programma in linguaggio C che, dato in input un numero naturale positivo, controlli se il numero è un numero primo.

**NB.** un numero si dice primo se è divisibile solo per 1 e per sé stesso.



## Esercizio - Test primalità

```
int main_prime()
{
    int n, i, flag = 0;

    printf("Inserire un numero positivo: ");
    scanf("%d",&n);

    for(i=2; i<=n/2; ++i)
    {
        // condition for nonprime number
        if(n%i==0)
        {
            flag=1;
            break;
        }
    }

    if (flag==0)
        printf("%d è un numero primo.\n",n);
    else
        printf("%d non è un numero primo.\n",n);

    return 0;
}
```



# Generazione numeri casuali

(Simplest ever random number generator)

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
              // guaranteed to be random.  
}
```





# Generazione numeri casuali

```
int main() {  
    int c, n;  
  
    srand((unsigned)time(NULL));  
  
    printf("10 numeri interi presenti in [1,100]\n");  
  
    for (c = 1; c <= 10; c++) {  
        n = rand() % 100 + 1;  
        printf("%d\n", n);  
    }  
  
    return 0;  
}
```



# Esercizio 1

- Scrivere un programma in linguaggio C che legga da tastiera una sequenza di numeri positivi e ad ogni numero letto ne stampi la somma progressiva.
- Il programma termina quando si introduce un numero minore o uguale a zero.



# Esercizio 1 - Soluzione

```
#include <stdio.h>

int main()
{
    int num;
    int somma= 0;

    printf ("Inserisci un numero: ");
    scanf("%d", &num);

    while(num>=0)
    {
        somma = somma + num;
        printf("La somma progressiva attuale è pari a: %d\n", somma);

        printf ("Inserisci un numero: ");
        scanf("%d", &num);
    }
    printf("La somma progressiva finale è pari a: %d\n", somma);
    return 0;
}
```



## Esercizio 2

Si chiedano 10 numeri in input. Verificare se i numeri inseriti sono pari o dispari.

Stampare a video prima tutti i numeri dispari, poi tutti i pari.



## Esercizio 2 - Soluzione

```
#include<stdio.h>

int main() {
    int array[10];
    int i;

    printf("Inserisci 10 numeri: \n");

    for (i = 0; i < 10; i++) {
        scanf("%d", &array[i]);
    }

    for (i = 0; i < 10; i++) {
        if (array[i] % 2 != 0) {
            printf("Numero dispari: %d \n", array[i]);
        }
    }

    for (i = 0; i < 10; i++) {
        if (array[i] % 2 == 0) {
            printf("Numero pari: %d \n", array[i]);
        }
    }

    return 0;
}
```



## Esercizio 3

- Generare un numero a caso nell'intervallo  $[1,100]$  e chiedere all'utente un numero fino a quando non e' uguale a quello generato casualmente.
- Dire ogni volta se il numero immesso è  $>$  o  $<$  di quello iniziale.



## Esercizio 3 - Soluzione

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

int main()
{
    int num;
    int random;

    srand((unsigned)time(NULL));
    random = rand()% 100+1;
    num = random -1; //evito di impostare num pari a random

    while (num != random){
        printf("Inserisci un numero: ");
        scanf("%d", &num);
        if(num < random){
            printf("Inserisci un numero maggiore. \n");
        } else if (num > random){
            printf("Inserisci un numero minore. \n");
        }
    }
    printf("Numero trovato!");
}
```



## Esercizio 4

Scrivere un programma in linguaggio C che:

- Riceve in input un numero ***n*** dall'utente
- Costruisce un array di numeri interi vuoto di dimensione ***n***
- Popola l'array con numeri casuali nell'intervallo **[-200, 99]**
- Stampa l'array popolato
- Cerca se è presente il numero **42** (utilizzare l'algoritmo di **ricerca binaria**)





# Esercizio 4 - Soluzione

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

void binary_search_es1(int [], int, int, int);
void bubble_sort_es1(int [], int);

int main()
{
    int search, size, i;

    printf("Enter size of a list: ");
    scanf("%d", &size);
    int array[size];
    printf("Generating random numbers\n");
    for(i = 0; i < size; i++)
    {
        array[i] = (rand() % 300) - 200;
        printf("%d ", array[i]);
    }
    bubble_sort_es1(array, size);
    printf("\n\n");
    printf("Enter key to search\n");
    scanf("%d", &search);
    binary_search_es1(array, 0, size, search);
}
```

```
void bubble_sort_es1(int list[], int size)
{
    int temp, i, j;
    for (i = 0; i < size; i++)
    {
        for (j = i; j < size; j++)
        {
            if (list[i] > list[j])
            {
                temp = list[i];
                list[i] = list[j];
                list[j] = temp;
            }
        }
    }
}

void binary_search_es1(int array[], int low, int high, int search)
{
    int mid;

    if (low > high)
    {
        printf("%d non trovato.\n", search);
        return;
    }
    mid = (low + high) / 2;
    if (array[mid] == search)
    {
        printf("%d trovato in posizione %d.\n", search, mid);
    }
    else if (array[mid] > search)
    {
        binary_search_es1(array, low, mid - 1, search);
    }
    else if (array[mid] < search)
    {
        binary_search_es1(array, mid + 1, high, search);
    }
}
```



# Domande?



# Somma di numeri binari, Trasposta di una matrice, File

Dott. Emanuele Pio Barracchia

# Somma di numeri binari

- Scrivere un programma in linguaggio C, che dati due numeri binari sotto forma di array, ne calcola la somma

1	1	0	0
---	---	---	---

+

1	0	1	0
---	---	---	---

=

1	0	1	1	0
---	---	---	---	---



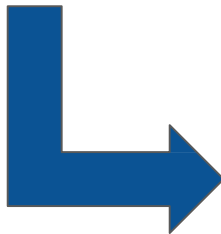
# Somma di numeri binari

```
int main(){
    int num1[] = {0, 0, 1, 1};
    int num2[] = {0, 1, 0, 1};

    int size1 = sizeof(num1)/sizeof(num1[0]);
    int size2 = sizeof(num2)/sizeof(num2[0]);

    //Stampo a video i due numeri binari
    printf("Primo numero binario: ");
    for (int i = size1-1; i >= 0; i--){
        printf("%d", num1[i]);
    }
    printf("\n");

    printf("Secondo numero binario: ");
    for (int i = size2-1; i >= 0; i--){
        printf("%d", num2[i]);
    }
    printf("\n");
}
```



```
int maxSize;
if (size1>size2)
    maxSize = size1;
else
    maxSize = size2;

int result[maxSize+1];
int carry = 0;

for (int i = 0; i < maxSize; i++){
    int somma = num1[i] + num2[i] + carry;

    if (somma > 1){
        somma = somma - 2;
        carry = 1;
    } else {
        carry = 0;
    }

    result[i] = somma;
}
result[maxSize] = carry;

//Stampo a video il risultato
printf("La somma è pari a : ");
for (int i = maxSize; i >= 0; i--){
    printf("%d", result[i]);
}
printf("\n");

return 0;
}
```



# Matrici

- Una matrice è un array bidimensionale
- Es. Definire una matrice di numeri interi in linguaggio C composta di N righe e M colonne

```
int matrix[n][m];
```



# Matrici - Esercizi

- Scrivere un programma in linguaggio C che, data una matrice, genera la sua trasposta
  - NB. La matrice trasposta è la matrice ottenuta scambiando le righe con le colonne

$$\begin{bmatrix} 9 & 0 & 1 \\ 0 & 11 & 1 \\ 1 & 1 & 4 \\ 1 & 0 & 1 \end{bmatrix} \xrightarrow{\text{Transpose}} \begin{bmatrix} 9 & 0 & 1 & 1 \\ 0 & 11 & 1 & 0 \\ 1 & 1 & 4 & 1 \end{bmatrix}$$



# Matrici - Esercizi

```
int main(){
    int matrix[4][3] = {
        {9,0,1},
        {0,11,1},
        {1,1,4},
        {1,0,1}
    };

    int numRows = sizeof(matrix)/sizeof(matrix[0]);
    int numCol = (sizeof(matrix[0])/sizeof(matrix[0][0]));

    //stampo a video la matrice
    printf("Matrice di partenza: \n");
    for (int i = 0; i < numRows; i++){
        for (int j = 0; j < numCol; j++){
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}
```

```
printf("-----\n");

//genero la traposta della matrice
int transMatrix[numCol][numRows];
for (int i = 0; i < numRows; i++){
    for (int j = 0; j < numCol; j++){
        transMatrix[j][i] = matrix[i][j];
    }
}

printf("Matrice trasposta: \n");
for (int i = 0; i < numCol; i++){
    for (int j = 0; j < numRows; j++){
        printf("%d ", transMatrix[i][j]);
    }
    printf("\n");
}

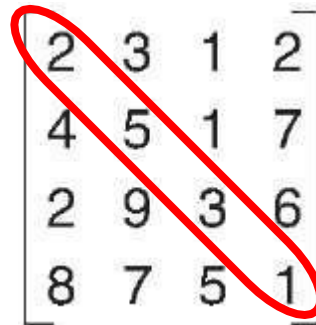
return 0;
}
```





## Matrici - Esercizi

- Scrivere un programma in linguaggio C che, data una matrice quadrata, somma gli elementi presenti sulla diagonale



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



# Matrici - Esercizi

```
int main(){
    int matrix[4][4] = {
        {2,3,1,2},
        {4,5,1,7},
        {2,9,3,6},
        {8,7,5,1}
    };

    int numRows = sizeof(matrix)/sizeof(matrix[0]);
    int numCol = (sizeof(matrix[0])/sizeof(matrix[0][0]));

    //stampo a video la matrice
    printf("Matrice di partenza: \n");

    for (int i = 0; i < numRows; i++){
        for (int j = 0; j < numCol; j++){
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    printf("-----\n");

    int sum = 0;

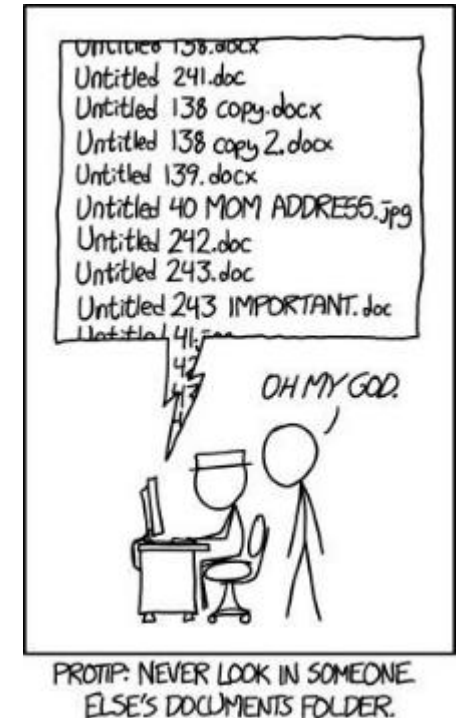
    for (int i = 0; i < numRows; i++){
        sum = sum + matrix[i][i];
    }

    printf("La somma degli elementi sulla diagonale è pari a: %d \n", sum);
    return 0;
}
```



# File

- In C le operazioni di input/output avvengono mediante l'uso di **stream**, ovvero astrazioni rappresentative di un file o di un dispositivo fisico che sono manipolabili mediante l'uso di puntatori
- Operazioni frequenti su stream sono: **apertura, accesso e chiusura**
- Esistono i buffer per gli stream in modo tale da evitare ritardi o interruzioni nella fase di lettura e scrittura.
  - NB: il contenuto di un buffer non viene mandato al file o al dispositivo finchè il buffer non è svuotato o chiuso



# Stream

Gli stream predefiniti nel linguaggio C sono:

- **stdin**, per i flussi in input. Il dispositivo di default è la tastiera
- **stdout**, per i flussi in output. Il dispositivo di default è la console
- **stderr**, per i messaggi di errore. Il dispositivo di default è la console



# Apertura di un file

La funzione utilizzata per aprire un file è la seguente:

**FILE \*fopen(char \*nome\_file, char \*modalità\_di\_apertura)**

La funzione **fopen** accetta le seguenti modalità di apertura di un file:

Parametro	Azione	Descrizione
r	(read) lettura	Lettura da un file esistente
w	(write) scrittura	Scrive un nuovo file o sovrascrive un file esistente
a	(append) aggiungere dati	Aggiunge dati partendo dalla fine del file



# Accesso al file

Una volta aperto un file, è possibile accedervi mediante due funzioni:

- **int** fprintf(**FILE** \*stream, **char** \*formato, argomenti ...)
- **int** fscanf(**FILE** \*stream, **char** \*formato, argomenti ...)



# Chiusura di un file

Infine, gli stream, una volta utilizzati, devono essere prima “puliti” e poi chiusi.

È possibile eseguire queste due operazioni, rispettivamente, con le funzioni `fflush` e `fclose`.

**`fflush(FILE *stream)`**

**`fclose(FILE *stream)`**



## File - Esercizi

- Scrivere un programma in linguaggio C che crea un nuovo file di nome “provaFile.txt” e scrive una stringa data in input dall’utente





# File - Esercizi

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

int main()
{
    char str[1000];

    char path[40]="provaFile.txt";

    //Creazione del file
    FILE *file = fopen(path,"w");
    if(file == NULL)
    {
        printf("Errore durante l'apertura del file");
        exit(1);
    }
    printf("Inserire frase da inserire nel file : ");
    fgets(str, sizeof(str), stdin);
    fprintf(file,"%s",str);

    fflush(file);
    fclose(file);
    printf("\n Il file %s è stato creato con successo \n",path);
    return 0;
}
```



# File - Esercizi

- Scrivere un programma in linguaggio C che legge il contenuto di un file e conta il numero di righe presenti
  - NB. EOF è un carattere speciale che indica la fine del file



# File - Esercizi

```
#include <stdio.h>

int main()
{
    FILE *fptr;
    int numRows = 0;
    char path[100];
    char c;
    printf("Inserire il path del file da aprire : \n");
    scanf("%s", path);

    fptr = fopen(path, "r");
    if (fptr == NULL)
    {
        printf("Impossibile aprire il file %s", path);
        return 0;
    }
    // Lettura del file
    printf("-----\n");
    printf("Contenuto del file: \n");

    while(fscanf(fptr, "%c" , &c) != EOF){
        //Stampo a video il carattere
        printf("%c", c);

        if (c == '\n'){ //Se trovo \n sta iniziando una nuova riga
            numRows = numRows + 1;
        }
    }
    printf("\n-----\n");
    fflush(fptr);
    fclose(fptr);
    printf("Il numero di righe presenti nel file %s è pari a: %d \n", path, numRows+1);
    return 0;
}
```



# File - Esercizi

```
#include <stdio.h>

int main()
{
    FILE *fptr;
    int numRows = 0;
    char path[100];
    char c;
    printf("Inserire il path del file da aprire : \n");
    scanf("%s", path);

    fptr = fopen(path, "r");
    if (fptr == NULL)
    {
        printf("Impossibile aprire il file %s", path);
        return 0;
    }
    // Lettura del file
    printf("-----\n");
    printf("Contenuto del file: \n");
    for (c = getc(fptr); c != EOF; c = getc(fptr)){
        //Stampo a video il carattere
        printf("%c", c);

        if (c == '\n'){ //Se trovo \n sta iniziando una nuova riga
            numRows = numRows + 1;
        }
    }
    printf("\n-----\n");
    fflush(fptr);
    fclose(fptr);
    printf("Il numero di righe presenti nel file %s è pari a: %d ", path, numRows+1);
    return 0;
}
```



# Domande?

