Corso di Informatica A.A. 2009-2010

Lezione 15


Liste concatenate

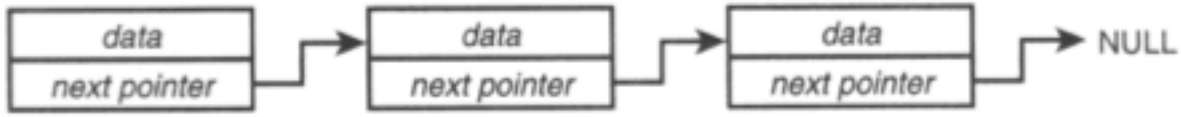
Liste concatenate

Una lista concatenata costituisce un metodo molto utile per immagazzinare dati che può essere implementato facilmente in C

Ogni blocco di dati in una lista concatenata è dato da una struttura. Tale struttura contiene oltre che le informazioni caratterizzanti l'elemento anche un puntatore che fornisce il *link* ad un altro elemento della lista.

Liste concatenate

```
struct person {
char name[20];
struct person *next;
};
```



L'ultima struttura non punta a nessun elemento

Le strutture che costituiscono una lista concatenata prendono il nome di elementi o nodi

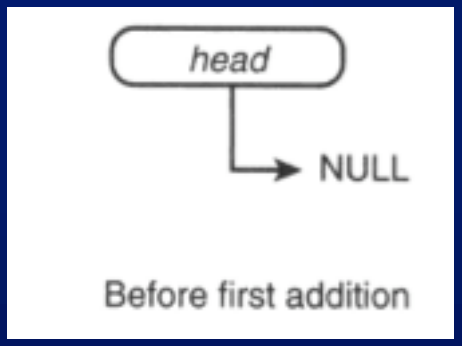
Liste concatenate

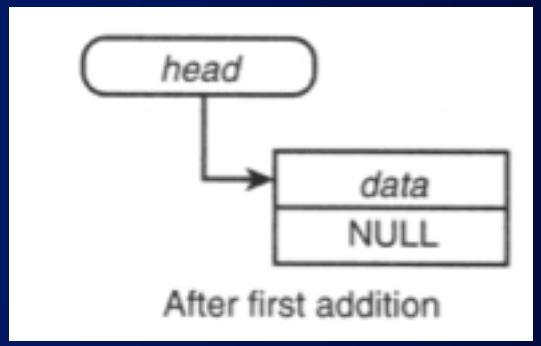
L'ultimo nodo di una lista possiede un puntatore a `NULL`

Il primo nodo è invece identificato da un puntatore speciale (non una struttura) che prende il nome di *head pointer*.

L'head pointer punta sempre al primo elemento di una lista. Il primo elemento contiene un puntatore che punta al secondo elemento; il secondo elemento contiene un puntatore che punta al terzo elemento e così via sino all'ultimo elemento il cui puntatore punta a `NULL`

Liste concatenate





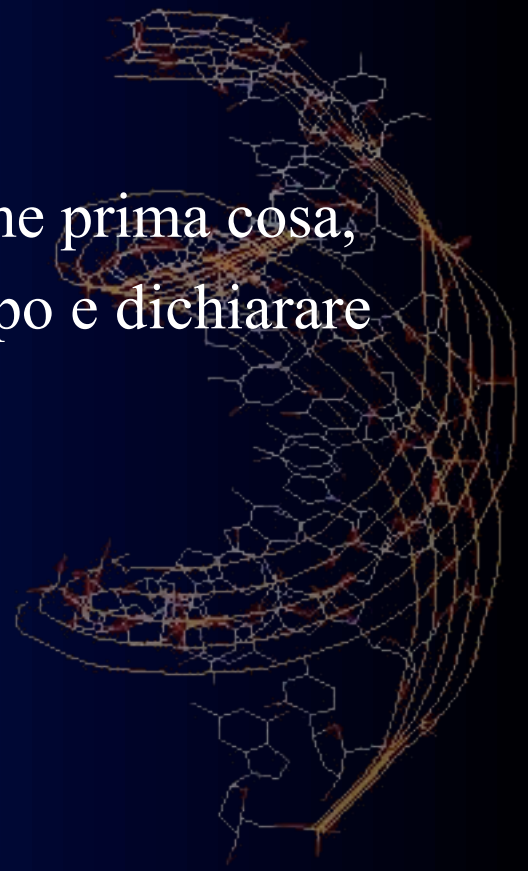
Quando si lavora con una lista concatenata è possibile aggiungere, eliminare o modificare elementi .

La modifica di un elemento non costituisce un compito gravoso mentre l'eliminazione o l'aggiunta di un elemento richiede una certa attenzione

Preliminari

Per introdurre una lista concatenata bisogna, come prima cosa, definire la struttura che sarà utilizzata a tale scopo e dichiarare l'head pointer

```
struct person {
  char name[20];
  struct person *next;
};
struct person *new;
struct person *head;
head = NULL;
```

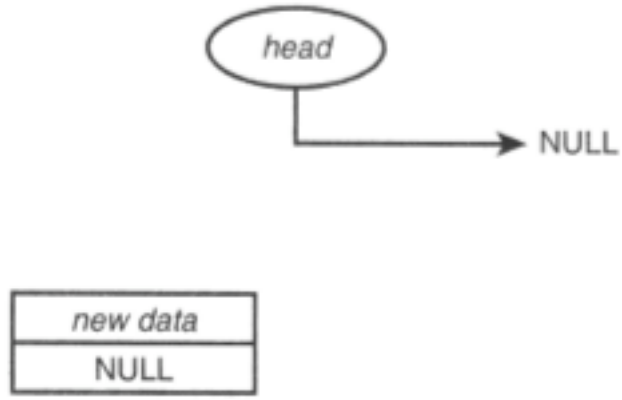


Aggiunta di un nodo all'inizio della lista

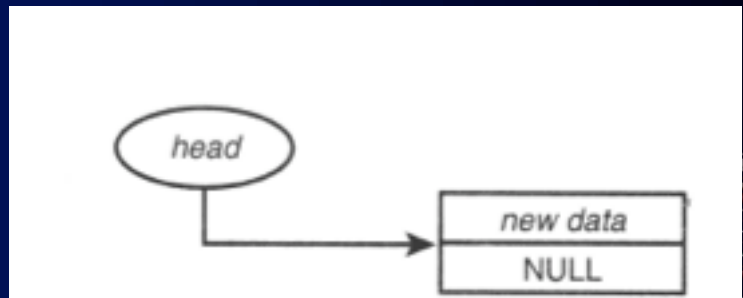
La procedura è la stessa sia che la lista sia inizialmente vuota che in caso contrario

1. Si alloca la memoria necessaria per un nuovo elemento con la funzione `malloc()`
2. Si pone il puntatore next pari al valore dell'head pointer
3. Si fa puntare l'head pointer al nuovo elemento

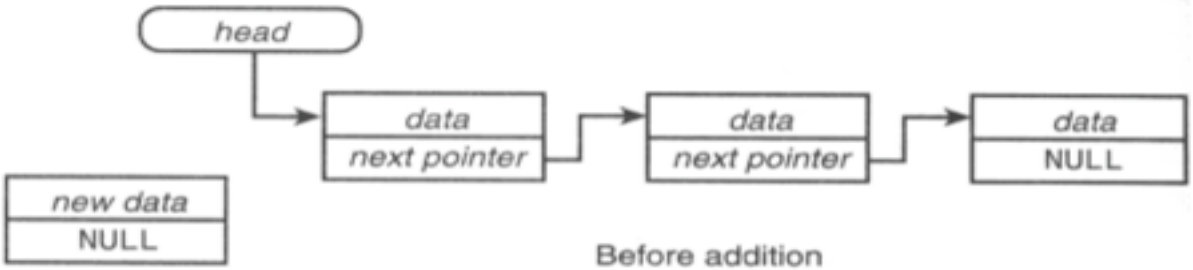
```
new = (person*)malloc(sizeof(struct person));
new->next = head;
head = new
```



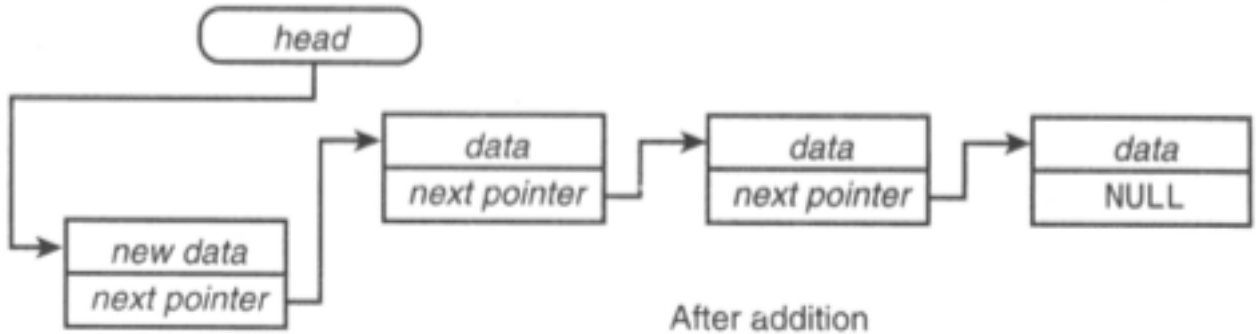
Before addition



After addition



Before addition



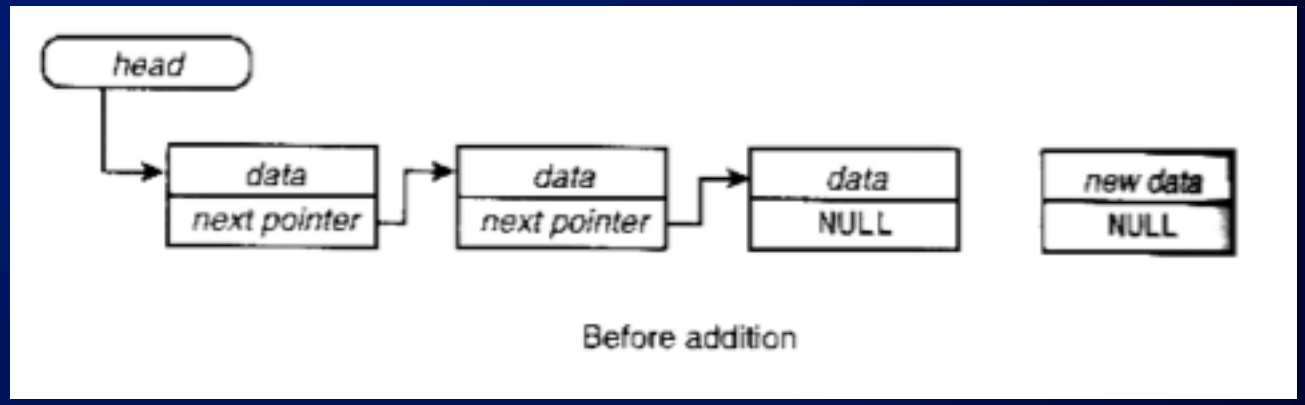
After addition

Aggiunta di un nodo alla fine della lista

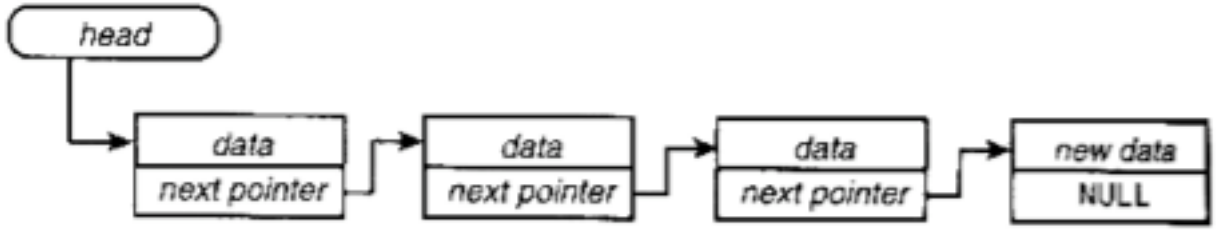
Si comincia dall'head pointer e si scorre la lista sino a raggiungere l'ultimo elemento. Si seguono poi i seguenti passi

1. Si alloca la memoria necessaria per un nuovo elemento con la funzione `malloc()`
2. Si fa puntare il puntatore next dell'ultimo elemento al nuovo elemento
2. Si fa puntare a NULL il puntatore next del nuovo elemento

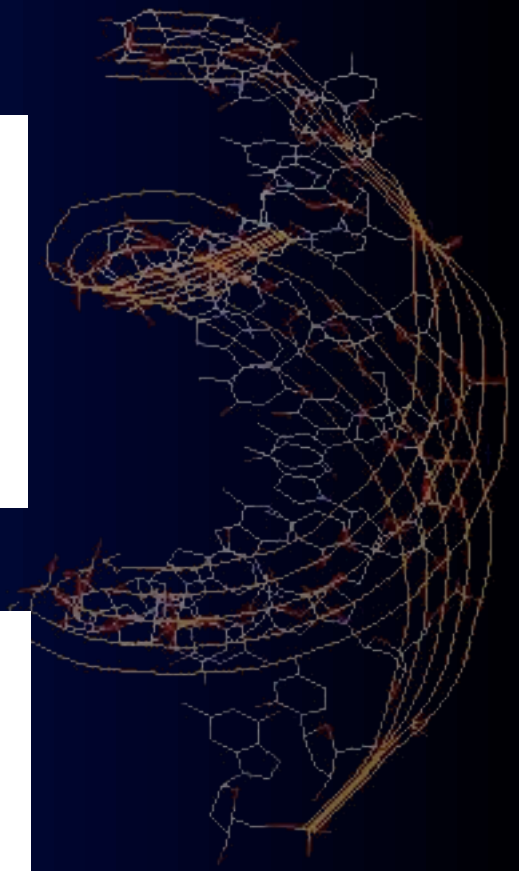
```
person *current;
 . . .
current = head;
while (current->next != NULL)
    current = current->next;
new = (person*)malloc(sizeof(struct person));
current->next = new;
new->next = NULL;
```



Before addition



After addition

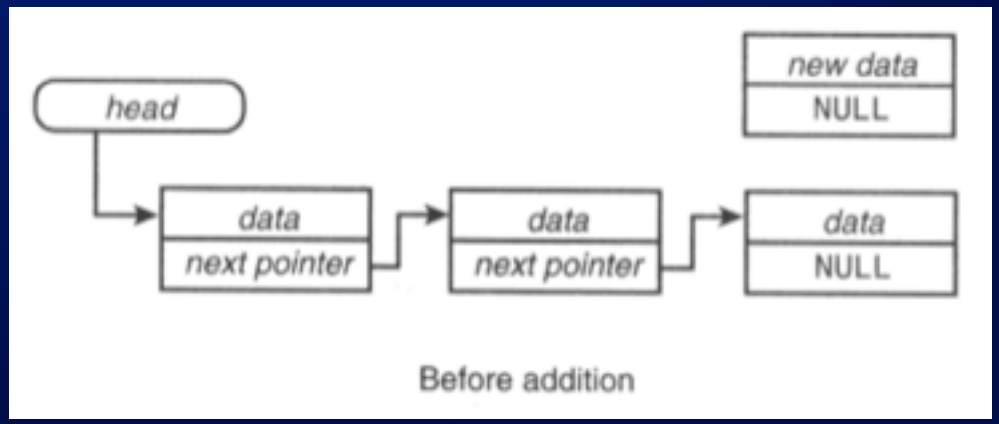


Aggiunta di un nodo nel mezzo della lista

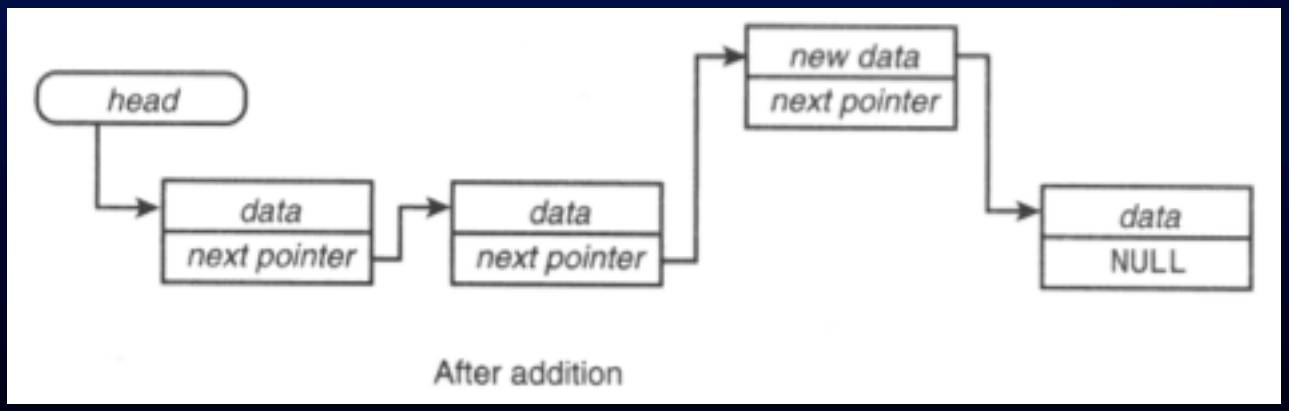
In realtà nella maggioranza dei casi si deve aggiungere un elemento nel mezzo della lista.

1. Si identifica nella lista il nodo subito dopo il quale va inserito il nuovo elemento (marker)
2. Si alloca la memoria necessaria per un nuovo elemento con la funzione `malloc()`
3. Si fa puntare il puntatore next del marker al nuovo elemento
4. Si fa puntare il puntatore next del nuovo elemento al nodo cui puntava il marker

```
person *marker;
/* Code here to set marker to point to the desired list location. */
. . .
new = (person *)malloc(sizeof(PERSON));
new->next = marker->next;
marker->next = new;
```

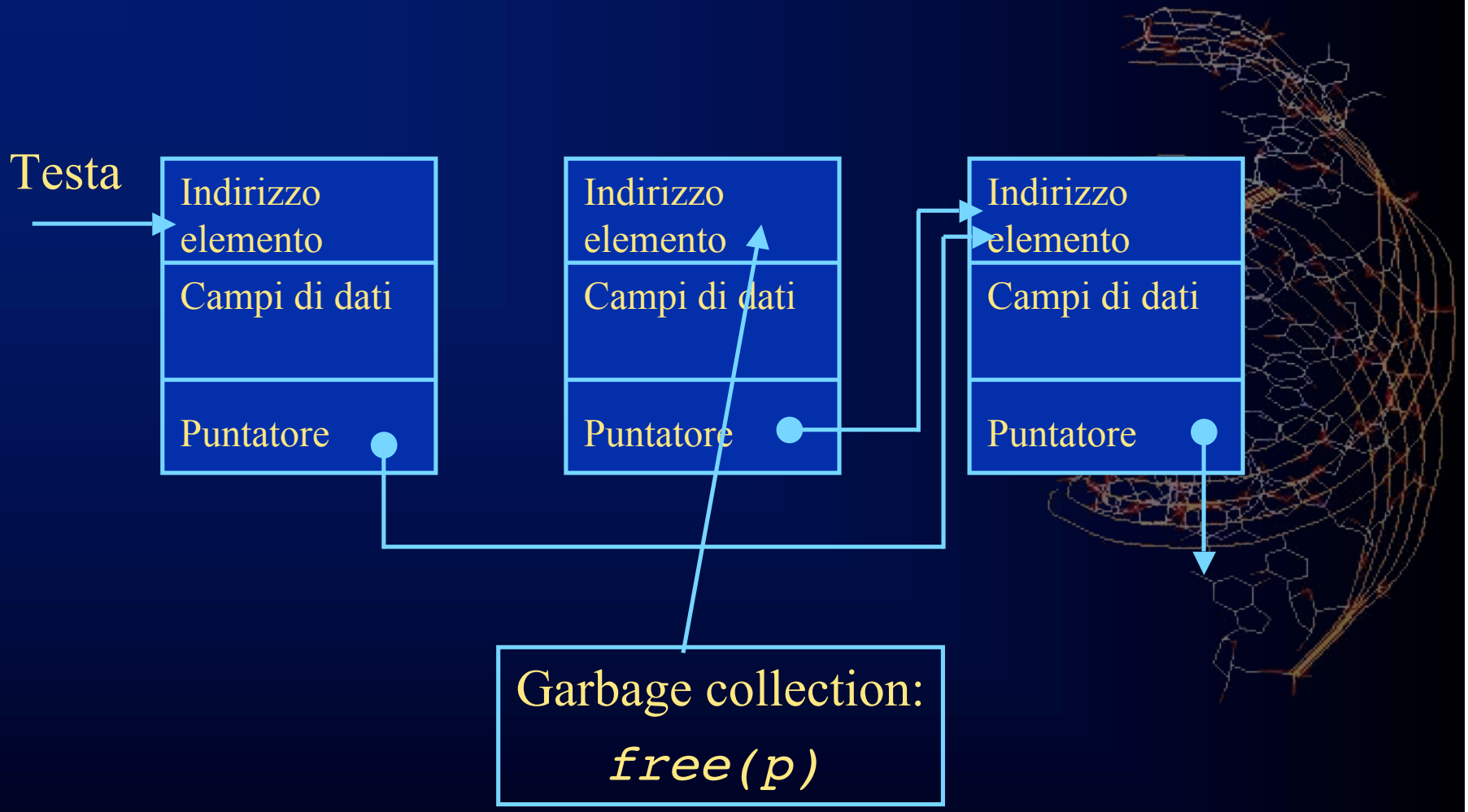


Before addition

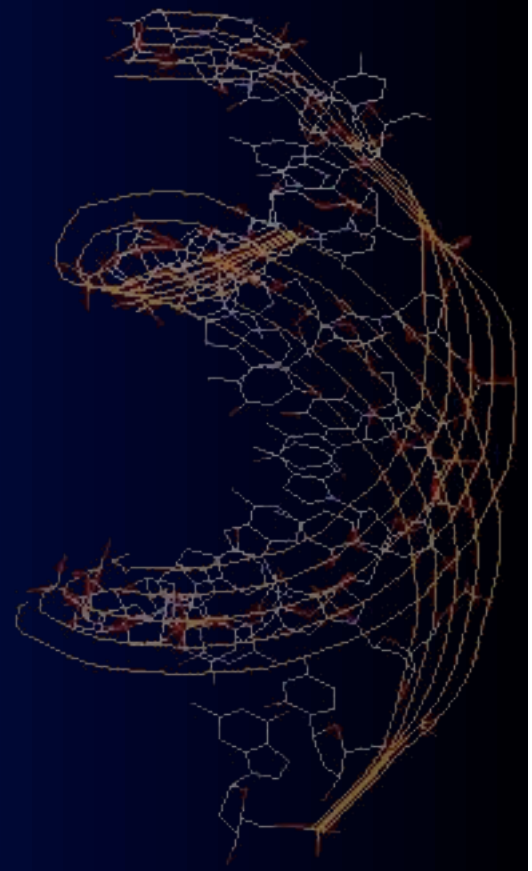


After addition

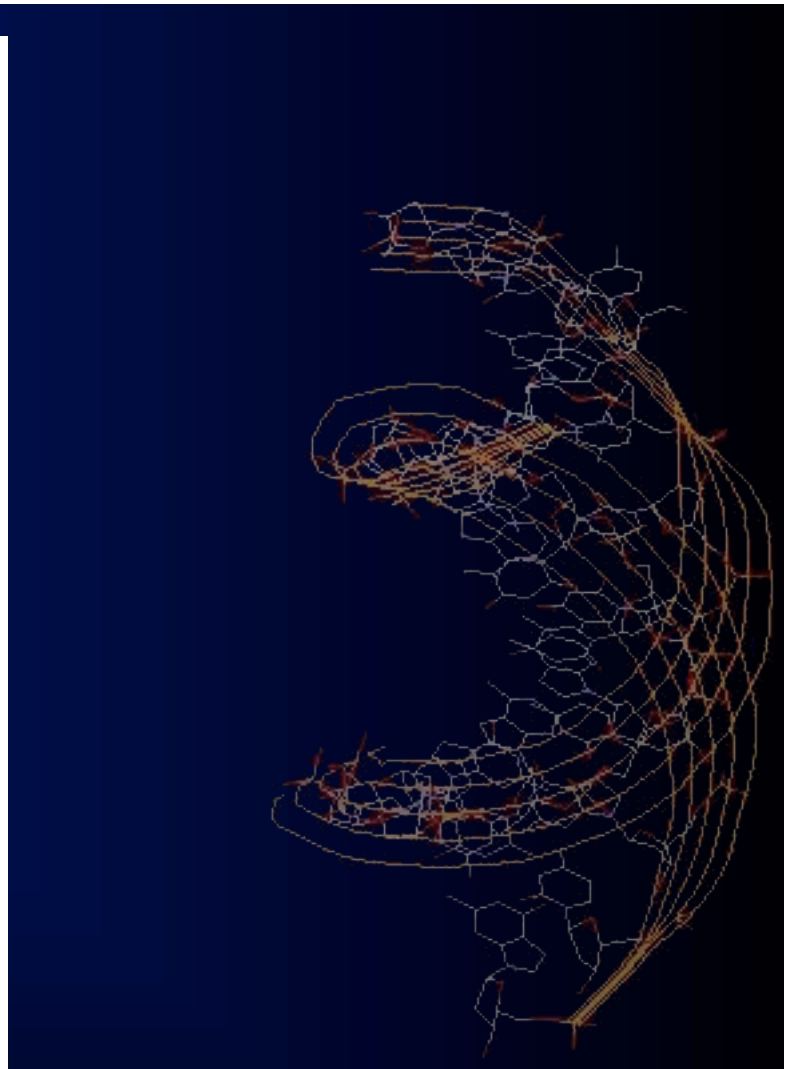
Cancellazione da una lista



```
/* Demonstrates the fundamentals of using */
/* a linked list. */

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

/* The list data structure. */
struct data {
     char name[20];
     struct data *next;
     };

/* Define typedefs for the structure */
/* and a pointer to it. */
typedef struct data PERSON;
typedef PERSON *LINK;
```



```
main()
{
    /* Head, and current element pointers. */
    LINK head = NULL;
    LINK new = NULL;
    LINK current = NULL;

    /* Add the first list element. We do not */
    /* assume the list is empty, although in */
    /* this demo program it always will be. */

    new = (LINK)malloc(sizeof(PERSON));
    new->next = head;
    head = new;
    strcpy(new->name, "Abigail");

    /* Add an element to the end of the list. */
    /* We assume the list contains at least one element. */

    current = head;
    while (current->next != NULL)
    {
        current = current->next;
    }

    new = (LINK)malloc(sizeof(PERSON));
    current->next = new;
    new->next = NULL;
    strcpy(new->name, "Catherine");

    /* Add a new element at the second position in the list. */
    new = (LINK)malloc(sizeof(PERSON));
    new->next = head->next;
    head->next = new;
    strcpy(new->name, "Beatrice");
```



```
    /* Print all data items in order. */
    current = head;
    while (current != NULL)
    {
        printf("\n%s", current->name);
        current = current->next;
    }

    printf("\n");
    return(0);
}
```

Un esempio: una lista di voli

Supponiamo di voler tenere una lista in tempo reale dei voli in arrivo in un aeroporto. La struttura base sarà:

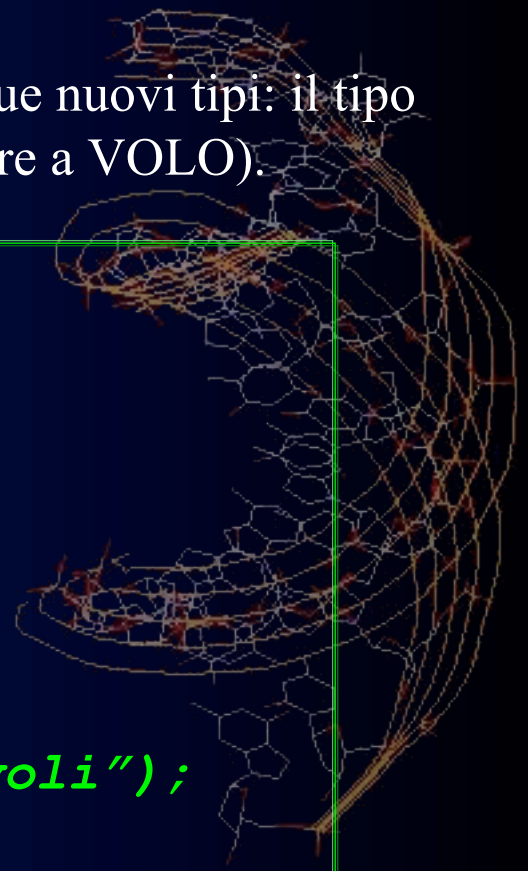
File= volodef.h

```
struct volo {
    int numero_volo;
    int num_passeggeri;
    int Stato;
    float tempo_arrivo_previsto;
    float tempo_arrivo_effettivo;
    int numero_gate;
    struct volo *next;
};
typedef struct volo VOLO;
typedef VOLO *PVOLO;
```

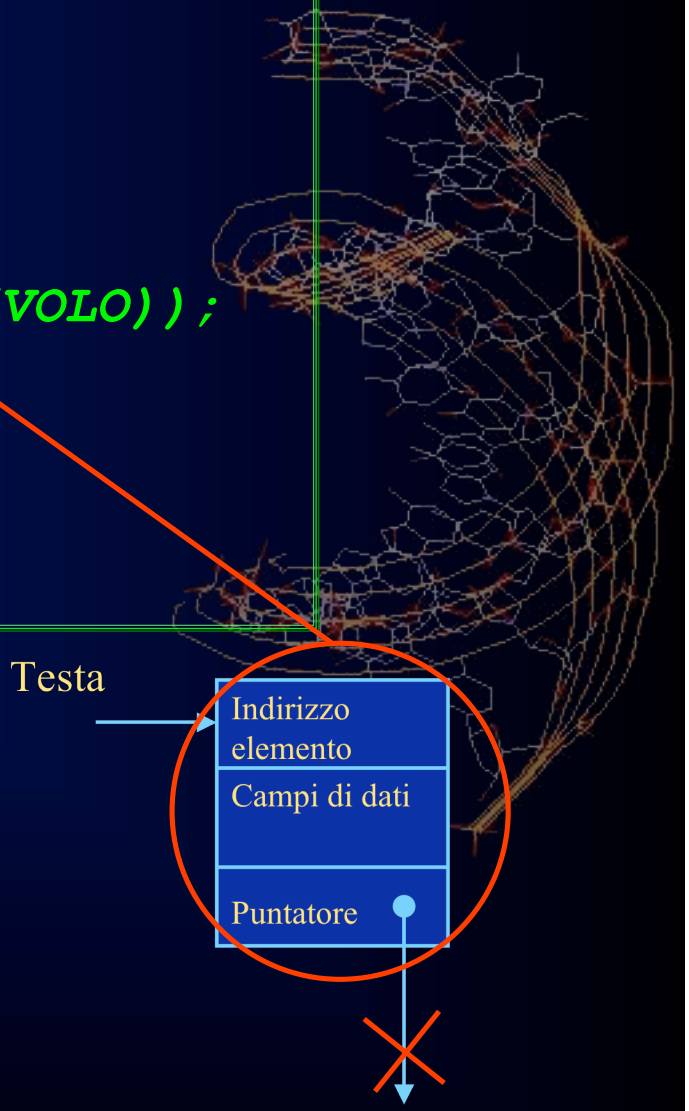
Creazione della lista di voli

Una volta inserito l'header volodef.h avremo a disposizione due nuovi tipi: il tipo VOLO (elemento della lista di voli) e il tipo PVOLO (puntatore a VOLO).

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

PVOLO crea_lista(void);
int main(){
 PVOLO testa_della_lista;
 printf("Sto generando la lista di voli");
 testa_della_lista = crea_lista();
 return 0;
}
/* segue… */
```



```
/* …segue */
PVOLO crea_lista(void){
 PVOLO testa;

   testa = (PVOLO)malloc(sizeof(VOLO));
    testa-> next = NULL;

 return (testa);
}
```



Inserimento e ….

```
void inserisci(PVOLO elemento1, PVOLO elemento_new){
 PVOLO temp;
 temp = elemento1->next;
 elemento1->next = elemento_new;
 elemento_new->next = temp;
}
```

..... cancellazione

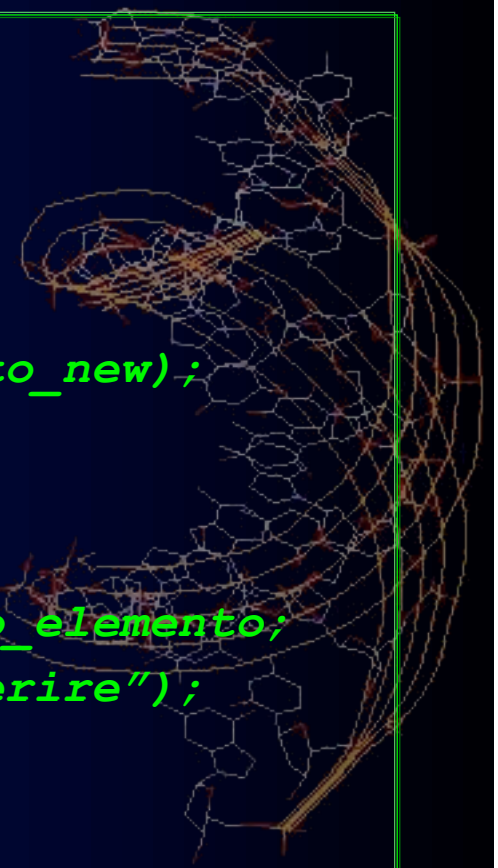

```
void cancella_successivo(PVOLO elemento){
 PVOLO garbage=elemento->next;
 elemento->next = garbage->next;
 free(garbage);
}
```

Riempimento della lista

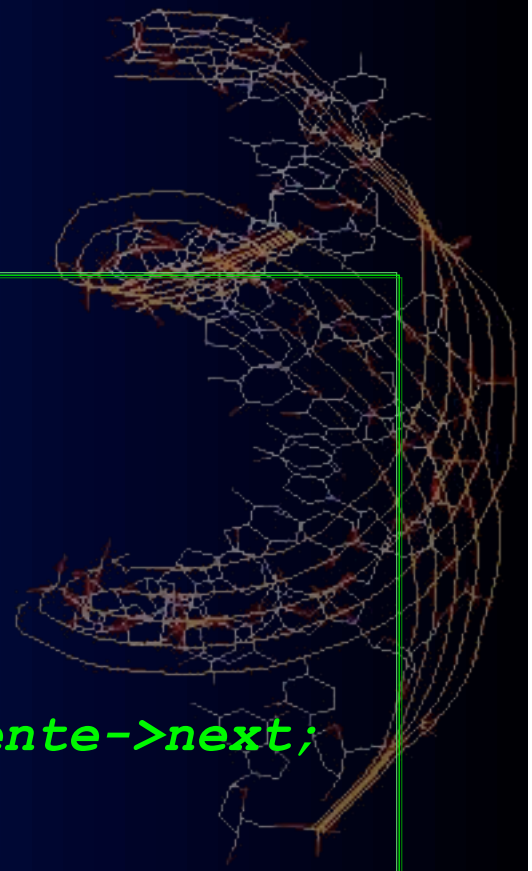
```
void riempi(PVOLO elemento){
 if(elemento!=NULL){
printf("\nInserisci il numero di volo, il numero
di passeggeri etc\n") ;
   scanf("%d %d",
   &elemento->numero_volo,
   &elemento->num_passeggeri);
 }
}
```

Riempimento della lista (II)

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

PVOLO crea_lista(void);
void inserisci(PVOLO elemento, PVOLO elemento_new);
void riempi(PVOLO elemento);
int main(){
 int n,i;
 PVOLO testa_lista, elemento_corrente, nuovo_elemento;
 printf("Inserisci il numero di voli da inserire");
 scanf("%d",&n);
 testa_lista = crea_lista();
for(i=1; i<n; i++) {
 nuovo_elemento = crea_lista();
 inserisci(testa_lista, nuovo_elemento);
}  /* ….segue….*/
```



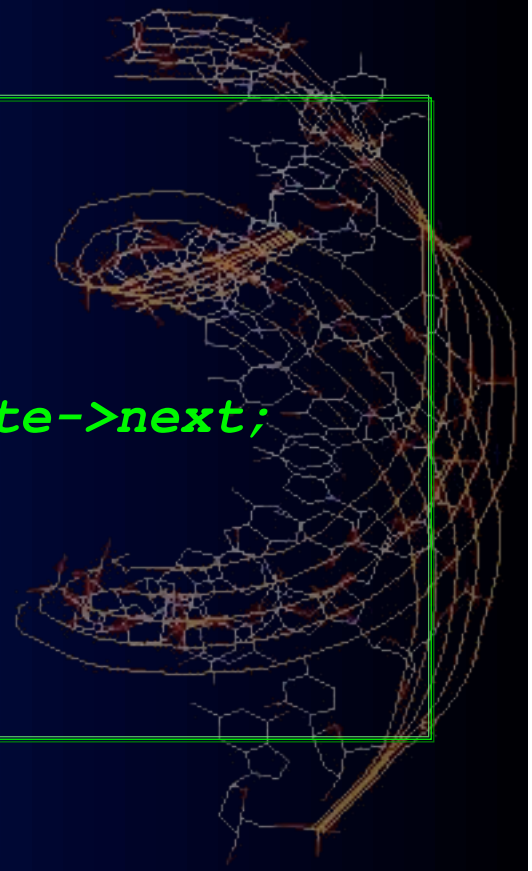
Riempimento della lista (III)



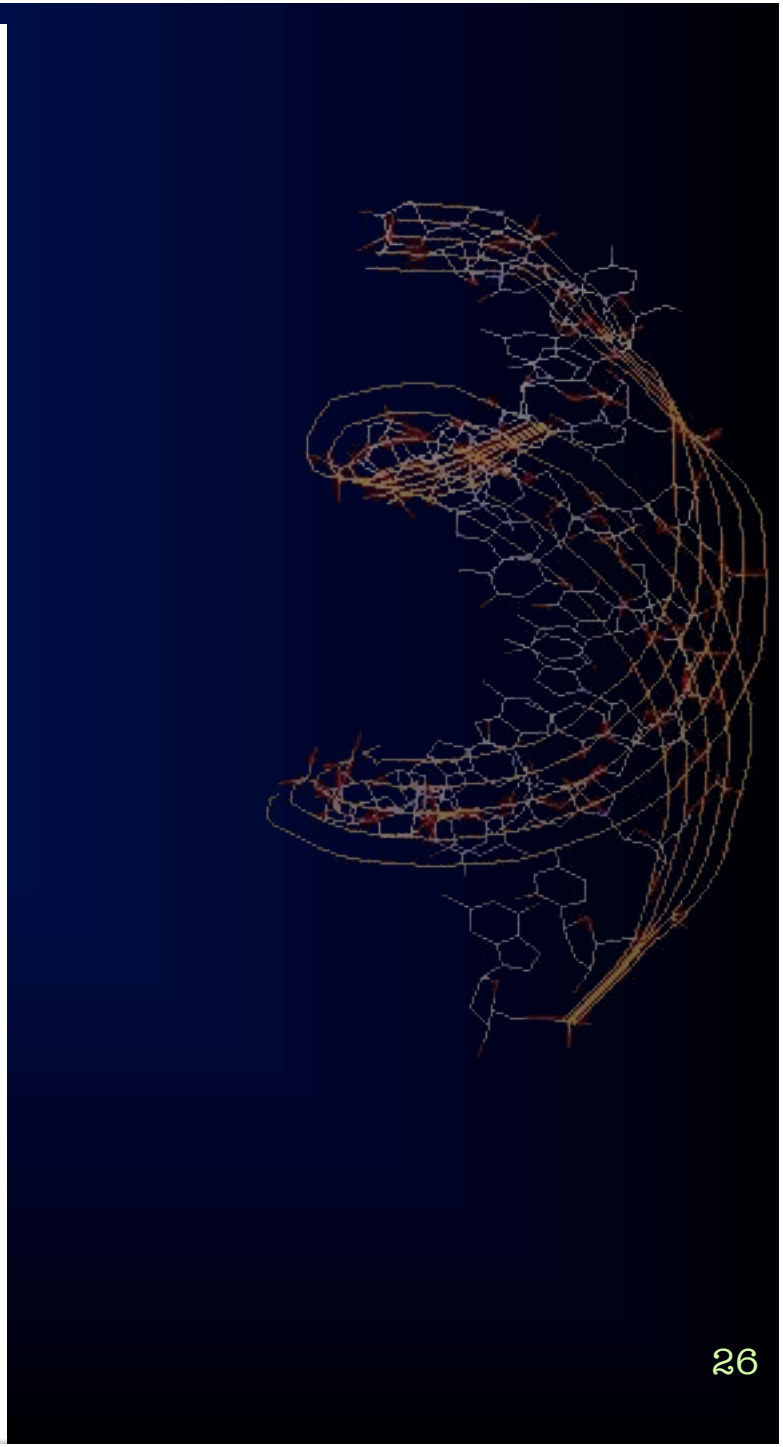
```
/* …segue….*/

elemento_corrente = testa_lista;

 while(elemento_corrente !=NULL){
  riempi(elemento_corrente);
  elemento_corrente = elemento_corrente->next;
 }
return 0;
}
```

Contare gli elementi…

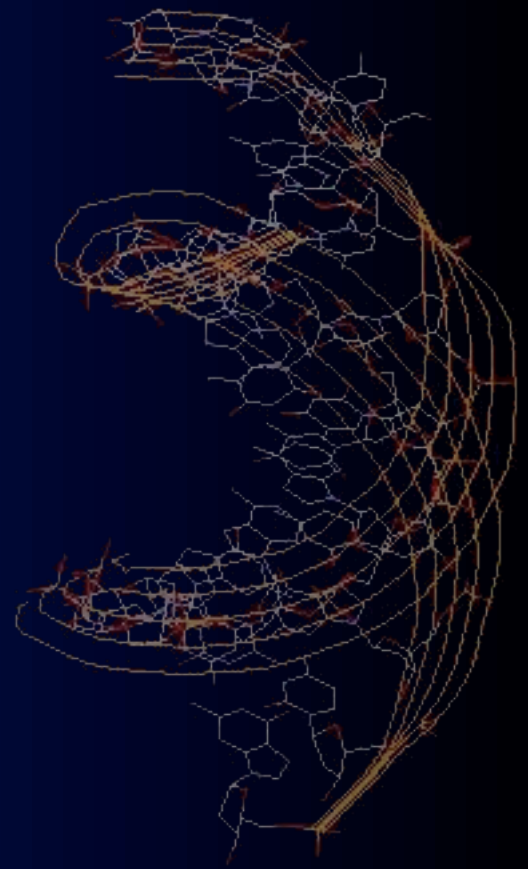
```
int conta (PVOLO testa){
 PVOLO elemento_corrente = testa;
 int count=0;
 while(elemento_corrente !=NULL){
  elemento_corrente = elemento_corrente->next;
  count++
 }
 return count;
}
```



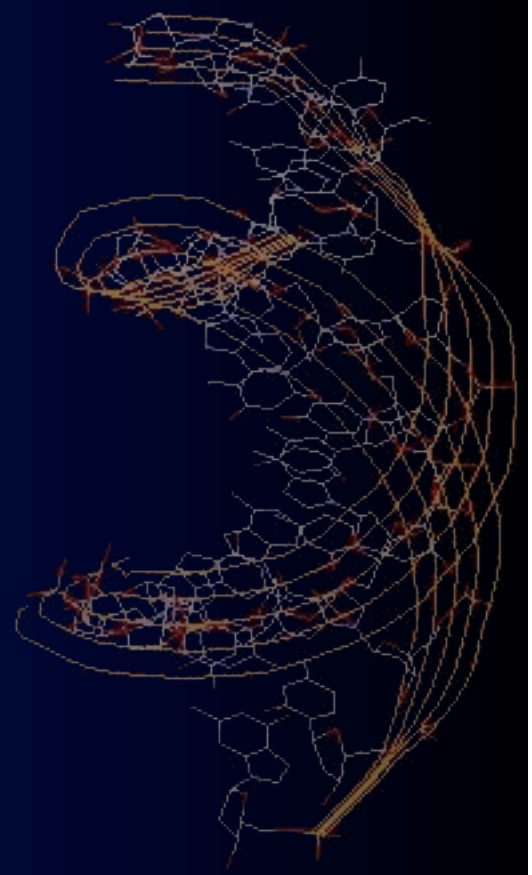
```c
# include <stdio.h>
# include <stdlib.h>

double media(double *vec, int n);
void minmax(double *vec, int n, double *m);


void  main( )
{
double *vettore , elemento , m[2];
int  i , j , fine ;
vettore = NULL ;
/*  inserimento  dati */
i = 0 ;
fine = 0 ;
while (!fine)
{
printf( " Inserisci un  elemento ( 0 per finire ): " ) ;
scanf( "%lf",&elemento) ;
if(elemento == 0.0 )
fine = 1 ;
else
{
  /* aumento di 1 la dimensione del vettore  */
  vettore = ( double *) realloc (vettore , (i+1)*sizeof(double) ) ;
vettore[i] = elemento ;
i++ ;
}
}
/* calcolo e stampa della media */
printf ( " La  media e' : %lf \n " , media(vettore,i)) ;
minmax(vettore,i , m);
printf ( " Il minimo e' %lf \n " , m[0]) ;
printf ( " Il massimo e' %lf \n " , m[1]) ;

/* deallocazione del vettore */
free(vettore) ;
}
```



```c
double media(double *vec, int n)
{
  int  j;
  double somma=0.0,ave;

for ( j = 0 ; j < n ; j++)
somma+= vec[j] ;
ave = somma/n ;
 return ave;
}

void minmax(double *vec, int n, double *m)
{
  int j;
  if(vec[0] >= vec[1])
    {
      m[0]=vec[1];
      m[1]=vec[0];
    }
  else
    {
      m[0]=vec[0];
      m[1]=vec[1];
    }
  for(j=2; j < n; j++)
    {
    if(vec[j] < m[0]) m[0]=vec[j];
    else if(vec[j] > m[1]) m[1]=vec[j];
    }
}
```



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

void main()
{
  int **matrice, **trasposta;
  int righe, colonne, r, c;

  printf("Quante righe ha la matrice: ");
  scanf("%d", &righe);
  printf("Quante colonne ha la matrice: ");
  scanf("%d", &colonne);

  /* allocazione della matrice */
  matrice = (int **)malloc(righe * sizeof(int *));
  for (r=0; r<righe; r++)
    matrice[r] = (int *)malloc(colonne * sizeof(int));

  /* inserimento dati */
  for (r=0; r<righe; r++)
    for (c=0; c<colonne; c++)
      {
        printf("Inserisci elemento di riga %d e colonna %d: ", r, c);
        scanf("%d", &matrice[r][c]);
      }
  /* alloco la matrice trasposta */
  trasposta = (int **)malloc(colonne * sizeof(int *));
  for (c=0; c<colonne; c++)
    trasposta[c] = (int *)malloc(righe * sizeof(int));
```



```
  /* calcolo della trasposta */
  for (r=0; r<righe; r++)
    for (c=0; c<colonne; c++)
      trasposta[c][r] = matrice[r][c];

  /* stampo la trasposta */
  printf("La trasposta e':\n");
  for (c=0; c<colonne; c++)
    {
      for (r=0; r<righe; r++)
        printf("%d ", trasposta[c][r]);
      printf("\n");
    }
    /* dealloco la matrice */
  for (r=0; r<righe; r++)
    free(matrice[r]);
  free(matrice);

  /* dealloco la trasposta */
  for (c=0; c<colonne; c++)
    free(trasposta[c]);
  free(trasposta);
}
```