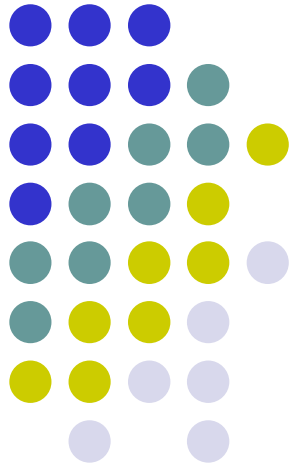
# Laboratorio di Calcolatori 1

## Corso di Laurea in Fisica

## A.A. 2007/2008



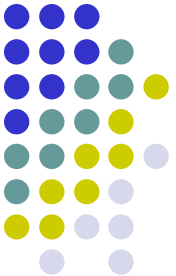
Dott.Davide Di Ruscio

Dipartimento di Informatica

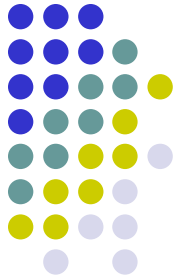
Università degli Studi di L'Aquila

Lezione del 29/05/08

# Nota



Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele



# Sommario (II parte)

Il Linguaggio C

- Caratteristiche generali
- Un linguaggio C semplificato ed esempi di semplici programmi
- Struttura di un programma C
- Direttive del pre-processore
- Parte dichiarativa:
  - tipi
  - definizioni di tipi
  - definizioni di variabili
- Parte esecutiiva
  - istruzione di assegnamento
  - istruzioni (funzioni) di input-output
  - istruzioni di selezione
  - istruzioni iterative
- Vettori mono e multidimensionali
- Funzioni e procedure
- File
- Allocazione dinamica di memoria
- Suddivisione dei programmi in piu' file e compilazione separata
- Algoritmi elementari
  - ricerca sequenziale e binaria
  - ordinamento di un vettore: per selezione, per inserimento, per fusione e a bolle
- Aspetti avanzati di programmazione
  - ricorsione
  - strutture dati dinamiche

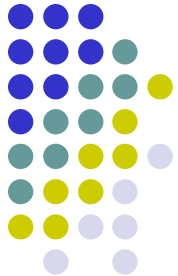
RIFERIMENTI

Ceri, Mandrioli, Sbattella

Informatica arte e mestiere

McGraw-Hill

# Allocazione e cancellazione di memoria



- `malloc (sizeof (TipoDato));`
  - Crea in memoria una variabile di tipo TipoDato, e restituisce come risultato l'indirizzo della variabile creata
  - Se P è una variabile di tipo puntatore a TipoDato, l'istruzione:

    ```
    P = malloc(sizeof(TipoDato));
    ```

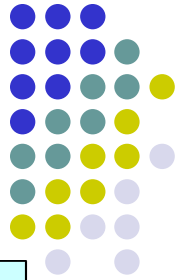
    assegna l'indirizzo restituito dalla funzione malloc a P

- `free(P)`
  - Rilascia lo spazio di memoria puntato da P

- Richiedono l'inclusione del file header stdlib.h tramite la direttiva

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

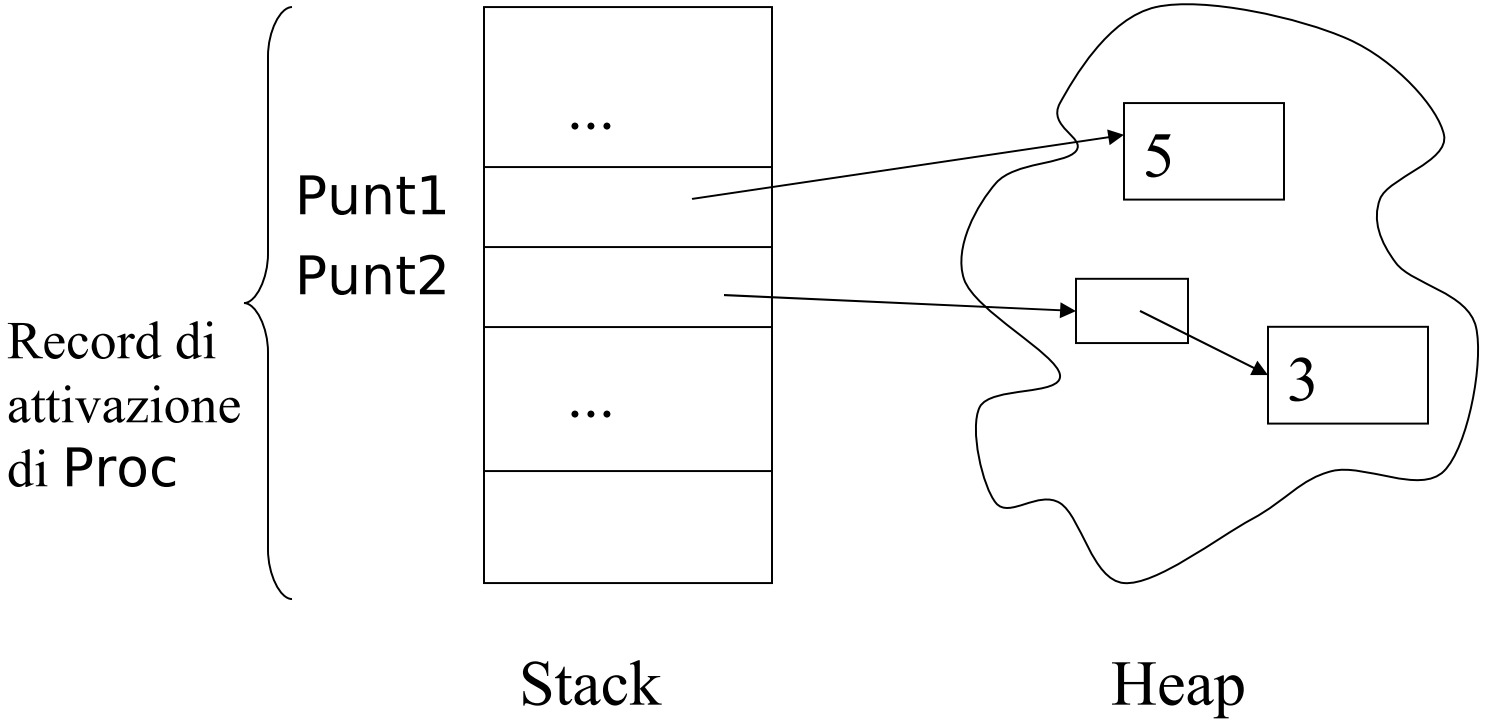
# **Esempio**: allocazione dinamica di vettori



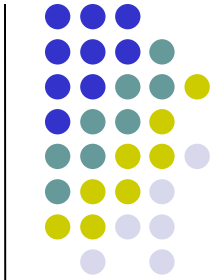
```
#include <stdio.h>
#include <stdlib.h>

main() {

   int *a,n,i,imin;

   scanf("%d",&n);
   if (n > 0)
   {
    a=(int *) malloc(n*sizeof(int));
    for (i=0; i<n; i++)
        scanf("%d",&a[i]);
    imin = 0;
    for (i=0; i<n; i++)
        if (a[i]<a[imin]) imin=i;

    printf("%d",imin);
   }
   else
    printf("Vettore vuoto");
}
```

# Stack e Heap

```
...        Proc ( ...)
..
int        *Punt1;
int        **Punt2;
...
```



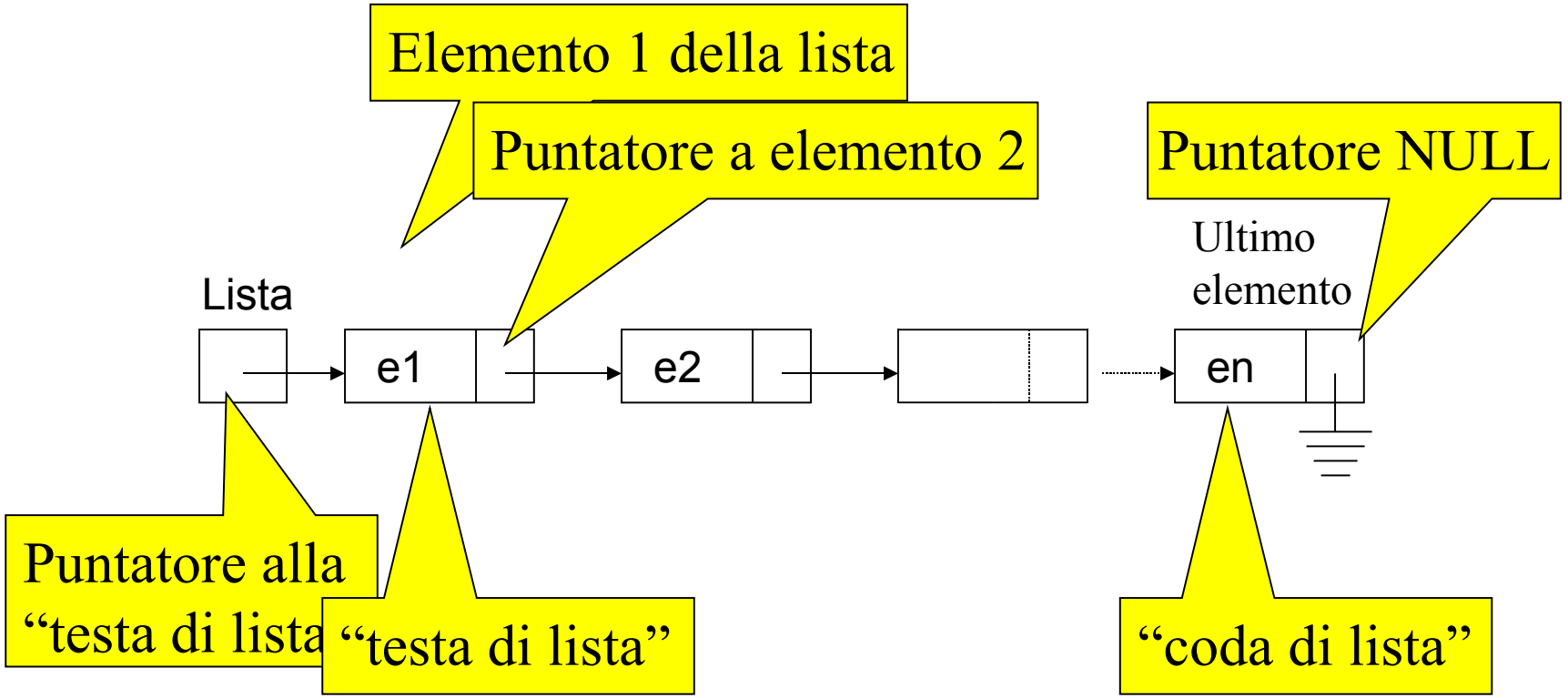
# Rischi della gestione dinamica della memoria



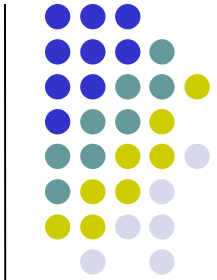
- produzione di *garbage* :
  - `P = malloc(sizeof(TipoDato));`
  - `P = Q;`
- “riferimenti fluttuanti” (*dangling references*):
  - `P = Q;`
  - `free(Q);`

# Lista dinamica

- Costruzione e gestione della struttura dinamica (pseudotipo astratto) *lista* mediante puntatori:



# Dichiarazione lista dinamica (1)



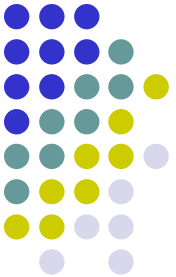
- **Invece di dichiarare il tipo lista, si dichiarano i suoi elementi**:

```
struct  EL   {
             TipoInfo     Info;
             struct EL        *Prox;
         };

typedef      struct EL        TipoElemLista;

typedef      TipoElemLista    *TipoLista;
```

- **Oppure**

```
typedef struct  EL {
                     TipoInfo      Info;
                     struct EL     *Prox;
                  } TipoElemLista, *TipoLista;
```

# Dichiarazione lista dinamica (2)



- Dichiarazione standard; mette in evidenza il tipo della lista:

```
TipoLista  Lista1, Lista2, Lista3;
```

- Dichiarazioni abbreviate:

```
TipoElemLista  *Lista1;
```

se non interessa mettere in evidenza il tipo della lista, ma si vuole indicare esplicitamente il tipo dei suoi elementi.

```
struct EL  *Lista1;
```

può sostituire entrambe le **typedef** se non è necessario nominare esplicitamente né il tipo della lista né il tipo dei suoi elementi.

# Operazioni sulle liste: Inizializzazione

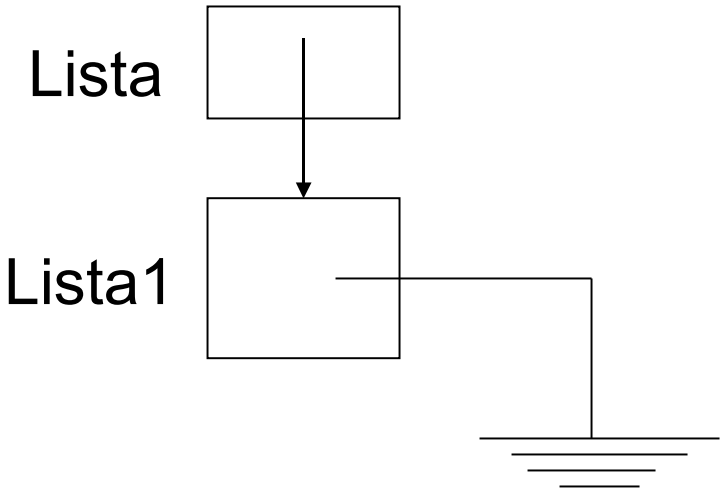
```
#include <stdlib.h>

/* Lista è la variabile locale che punta alla "testa di
   lista". La funzione assegna alla "testa di lista" il
   valore NULL corrispondente al valore di lista vuota */
void   Inizializza (TipoLista *Lista)
{
   *Lista = NULL;
}
```

- L'istruzione

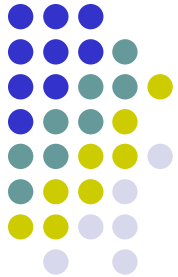
  ```
  Inizializza(&Lista1);
  ```

  produce:

- Al termine dell'esecuzione, il parametro formale Lista viene eliminato



# Controllo di lista vuota

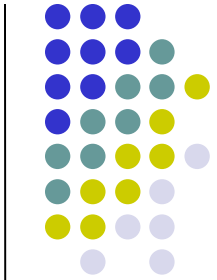


```
int ListaVuota (TipoLista Lista)
/* Produce il valore vero (1) se la lista passata come
   parametro è vuota, falso (0) in caso contrario, a Lista
   viene passato il valore contenuto nella variabile testa di
   lista. Lista punta pertanto al primo elemento della lista
   considerata */
{
   if (Lista == NULL)
    return 1;
   else
    return 0;
}
```

La chiamata sarà:

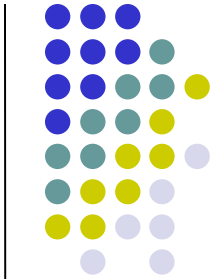
ListaVuota (Lista1)

# Ricerca di un elemento nella lista



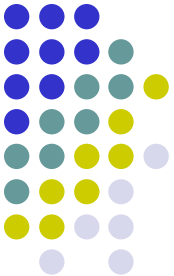
```
int Ricerca (TipoLista Lista, TipoInfo Info)
{
   TipoLista   Cursore;

   Cursore=Lista;

   while (Cursore != NULL && Cursore->Info != Info)
     Cursore = Cursore->Prox;
                /* In questa maniera Cursore viene fatto puntare
           all'elemento successivo della lista */

   if (Cursore== NULL) return 0;
        else return 1;
}
```

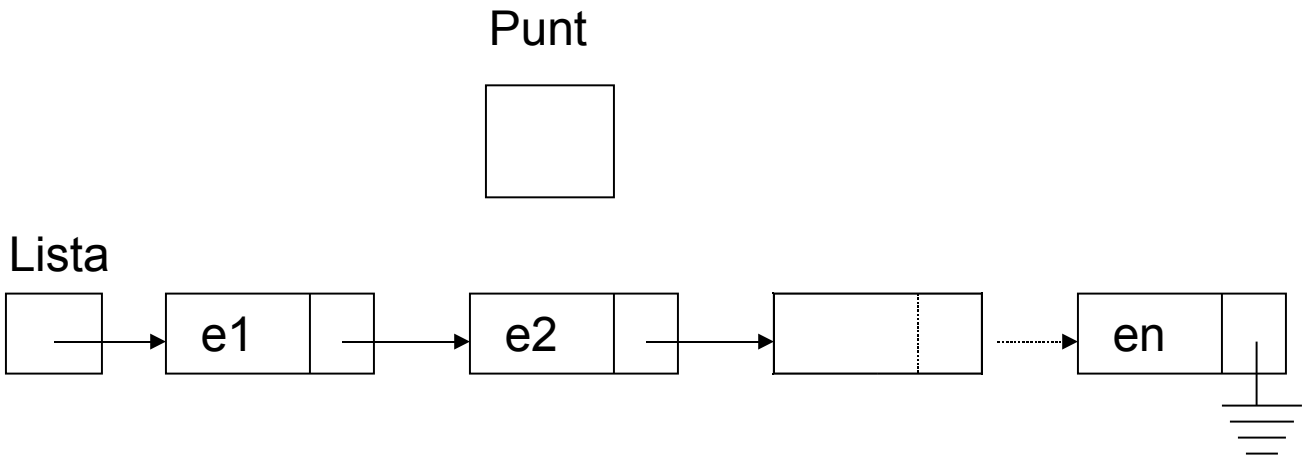
# Ricerca di un elemento nella lista, versione *ricorsiva*



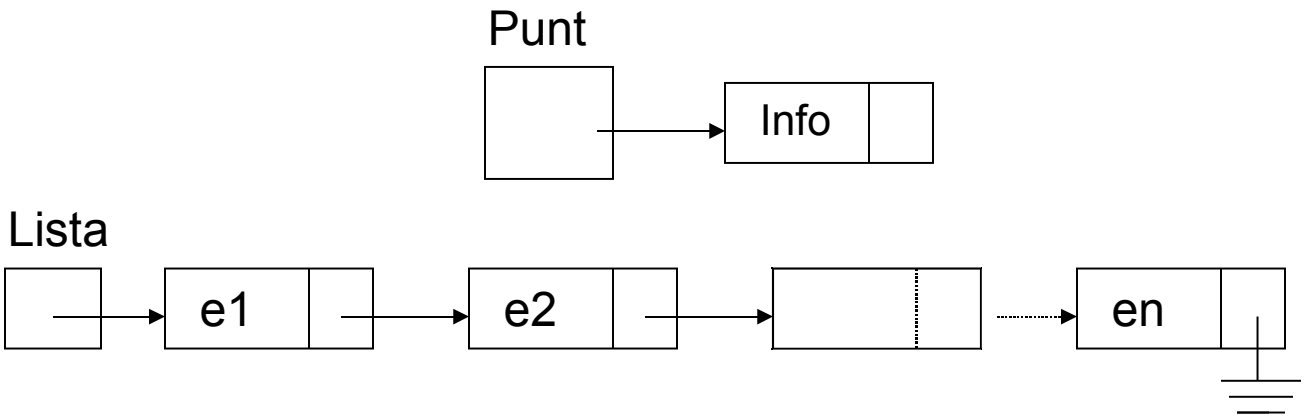
```
int  Ricerca (TipoLista Lista, TipoInfo   Info)
{
   if (Lista == NULL)
    return 0;
   else
    if (Lista->Info == Info)
        return 1;
    else
        return Ricerca(Lista->Prox, Info);
}
```

# Inserimento di un nuovo elemento in testa alla lista (1)

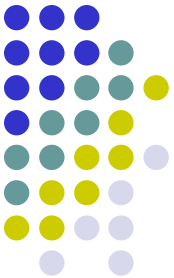




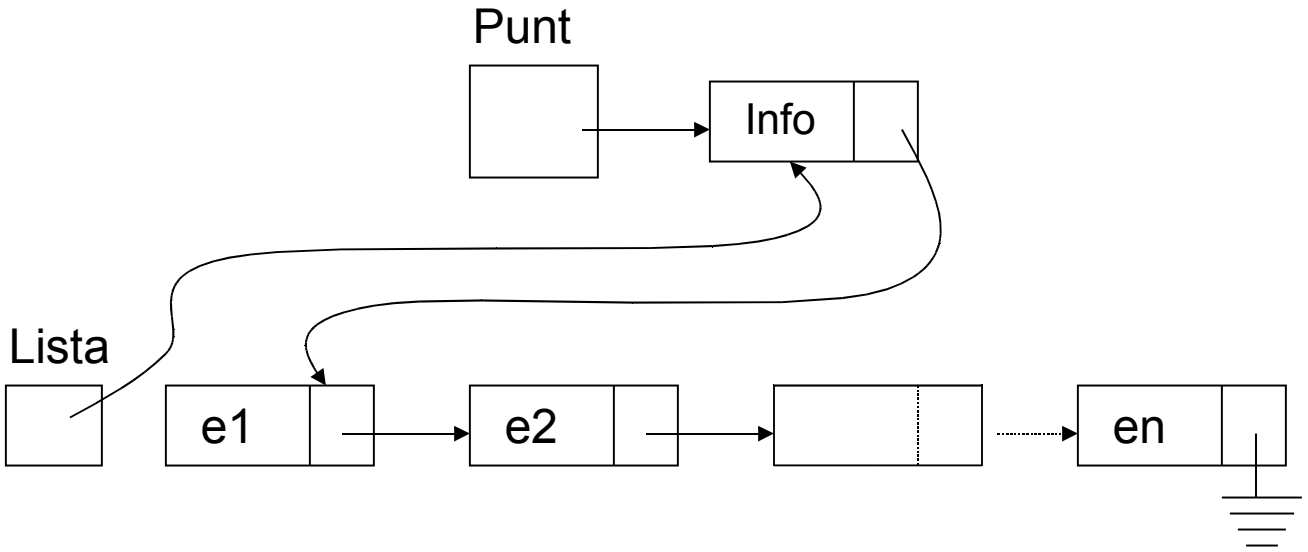

```
Punt = malloc(sizeof(TipoElemLista));
Punt->Info = Info;
```



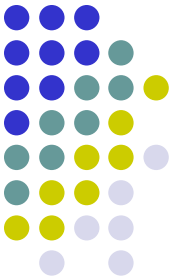
# Inserimento di un nuovo elemento in testa alla lista (2)



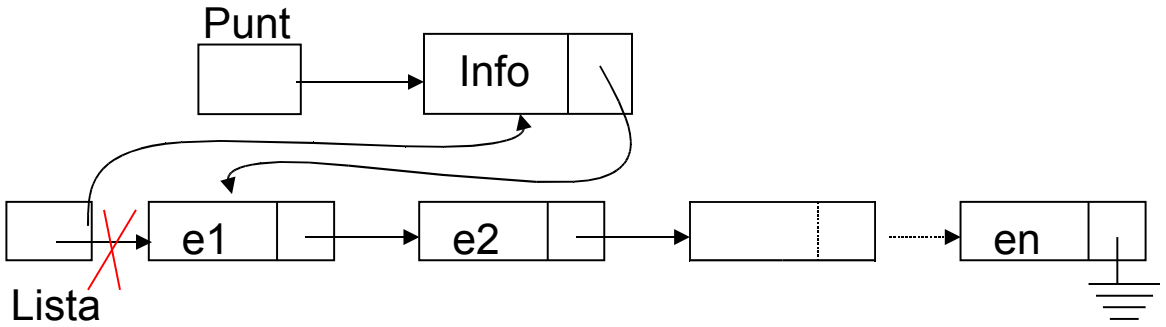
- Infine si collega il nuovo elemento al precedente primo elemento della lista e la testa della lista viene fatta puntare al nuovo elemento:



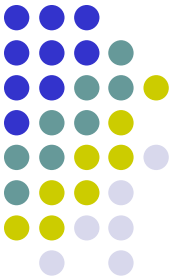
# Inserimento di un nuovo elemento in testa alla lista (3)



```
void InserisciInTesta (TipoLista    *Lista,    TipoInfo   Info)
{
   TipoLista   Punt;

   /* Allocazione dello spazio necessario per la memorizzazione
   del nuovo elemento e inizializzazione del puntatore */

   Punt = malloc(sizeof(TipoElemLista));
   Punt->Info = Info;
   Punt->Prox = *Lista;
   *Lista = Punt;
}
```

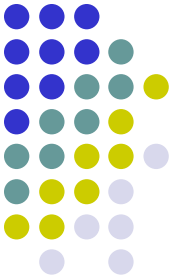


# Inserimento di un nuovo elemento in coda alla lista



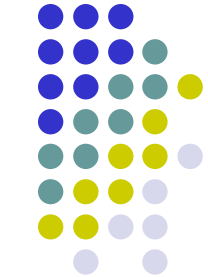
```
void   InserisciInCoda (TipoLista   *Lista,  TipoInfo Info)
{
   TipoLista   Cursore, Punt;

   Punt = malloc(sizeof(TipoElemLista));
   Punt->Prox = NULL;
   Punt->Info = Info;

   if (ListaVuota (*Lista))
    *Lista = Punt;
   else
   {
        Cursore=Lista;
        while (Cursore->prox != NULL)
            Cursore=Cursore->prox;
        Cursore->prox= Punt;
   }
}
```

# Inserimento di un nuovo elemento in coda alla lista, versione ricorsiva



```
void   InserisciInCoda (TipoLista  *Lista, TipoInfo Info)
{
   Tipolista   Punt;
   if (ListaVuota (*Lista))
   {
    Punt = malloc(sizeof(TipoElemLista));
    Punt->Prox = NULL;
    Punt->Info = Info;
    *Lista = Punt;
   }
   else InserisciIncoda(&((*Lista)->Prox), Info);
}
```

# Stampa Lista



```
void StampaLista(TipoLista Lista){
   TipoLista Cursore;

   Cursore=Lista;
   while (Cursore!=NULL){

    printf("|%d|->",Cursore->Info);
    Cursore=Cursore->Prox;
   }

   printf("NULL\n");

}
```