

```

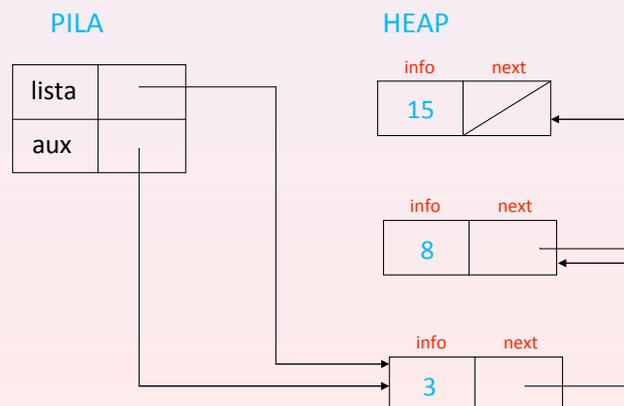
ListaDiElementi aux, lista = NULL;

aux = malloc(sizeof(ElementoLista));
aux->info = 15;    aux->next = lista;
lista = aux;

aux = malloc(sizeof(ElementoLista));
aux->info = 8;    aux->next = lista;
lista = aux;

aux = malloc(sizeof(ElementoLista));
aux->info = 3;    aux->next = lista;
lista = aux;

```



## Operazioni sulle liste

- ▶ Definiamo una serie di procedure e funzioni per **operare** sulle liste.
- ▶ Usiamo liste di interi per semplicità, ma tutte le operazioni sono realizzabili in modo del tutto analogo su liste di altro tipo (salvo rare eccezioni)
- ▶ Facciamo riferimento alle dichiarazioni dei tipi **ElementoLista** e **ListaDiElementi** viste in precedenza

### Inizializzazione

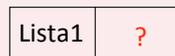
- ▶ Definiamo una procedura che inizializza una lista assegnando il valore **NULL** alla variabile **testa della lista**.
- ▶ Tale variabile deve essere modificata e quindi passata per **indirizzo**.
- ▶ Ciò provoca, nell'intestazione della procedura, la presenza di un puntatore a puntatore.

```
void Inizializza(ListaDiElementi *lista)
{
    *lista=NULL;
}
```

- Supponiamo ora che **Inizializza** sia chiamata passando come parametro l'indirizzo della variabile **Lista1** di tipo **ListaDiElementi**, ad esempio:

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

PILA

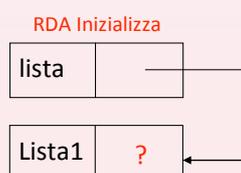


```
void Inizializza(ListaDiElementi *lista)
{
    *lista=NULL;
}
```

- Supponiamo ora che **Inizializza** sia chiamata passando come parametro l'indirizzo della variabile **Lista1** di tipo **ListaDiElementi**, ad esempio:

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

PILA

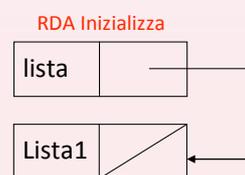


```
void Inizializza(ListaDiElementi *lista)
{
    *lista=NULL;
}
```

- Supponiamo ora che **Inizializza** sia chiamata passando come parametro l'indirizzo della variabile **Lista1** di tipo **ListaDiElementi**, ad esempio:

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

PILA

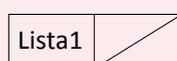


```
void Inizializza(ListaDiElementi *lista)
{
    *lista=NULL;
}
```

- Supponiamo ora che **Inizializza** sia chiamata passando come parametro l'indirizzo della variabile **Lista1** di tipo **ListaDiElementi**, ad esempio:

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

PILA



Cosa succederebbe se passassimo il parametro **per valore**?

```
void Inizializza(ListaDiElementi lista)
{
    lista=NULL;
}

main() {
    ListaDiElementi Lista1;
    Inizializza(Lista1);
    ...
}
```

PILA

|        |   |
|--------|---|
| Lista1 | ? |
|--------|---|

Cosa succederebbe se passassimo il parametro **per valore**?

```
void Inizializza(ListaDiElementi lista)
{
    lista=NULL;
}

main() {
    ListaDiElementi Lista1;
    Inizializza(Lista1);
    ...
}
```

PILA

RDA Inizializza

|       |   |
|-------|---|
| lista | ? |
|-------|---|

|        |   |
|--------|---|
| Lista1 | ? |
|--------|---|

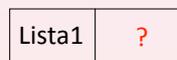
Cosa succederebbe se passassimo il parametro **per valore**?

```
void Inizializza(ListaDiElementi lista)
{
    lista=NULL;
}

main() {
    ListaDiElementi Lista1;
    Inizializza(Lista1);
    ...
}
```

PILA

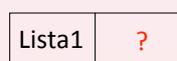
RDA Inizializza

Cosa succederebbe se passassimo il parametro **per valore**?

```
void Inizializza(ListaDiElementi lista)
{
    lista=NULL;
}

main() {
    ListaDiElementi Lista1;
    Inizializza(Lista1);
    ...
}
```

PILA



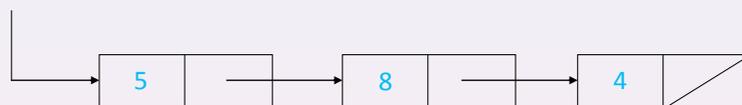
## Controllo lista vuota

```
boolean ListaVuota(ListaDiElementi lista)
{
    return (lista==NULL);
}
```

A `lista` viene passato il valore contenuto nella variabile testa di lista e quindi punta al primo elemento della lista considerata.

## Stampa degli elementi di una lista

- Data la lista



vogliamo che venga stampato:

5 -> 8 -> 4 -> //

## Versione iterativa:

```
void StampaLista(ListaDiElementi lis)
{
    while (lis != NULL)
    {
        printf("%d -->", lis->info);
        lis = lis->next;
    }
    printf("//");
}
```

- ▶ **N.B.:** `lis = lis->next` fa puntare `lis` all'elemento successivo della lista
- ▶ **Attenzione:** Possiamo usare `lis` per scorrere la lista perché, avendo utilizzato il passaggio per **valore**, le modifiche a `lis` non si ripercuotono sul parametro attuale.

```
void StampaLista(ListaDiElementi lis)
{
    while (lis != NULL)
    {
        printf("%d -->", lis->info);
        lis = lis->next;
    }
    printf("//");
}

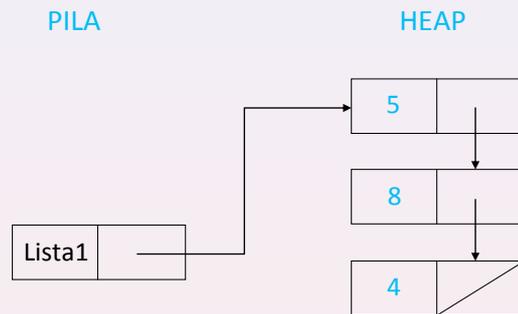
main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(Lista1);
    ...
}
```

```

void StampaLista(ListaDiElementi lis)
{
    while (lis != NULL)
    {
        printf("%d -->", lis->info);
        lis = lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(Lista1);
    ...
}

```

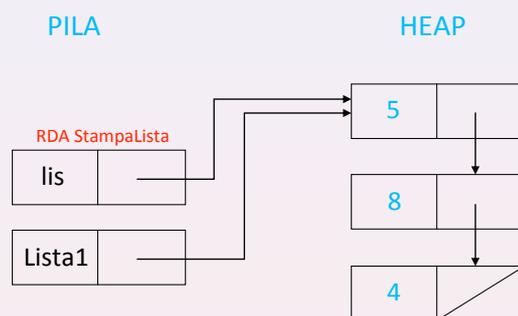


```

void StampaLista(ListaDiElementi lis)
{
    while (lis != NULL)
    {
        printf("%d -->", lis->info);
        lis = lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(Lista1);
    ...
}

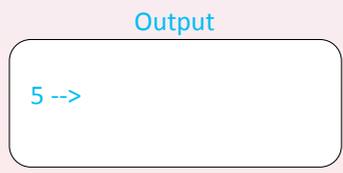
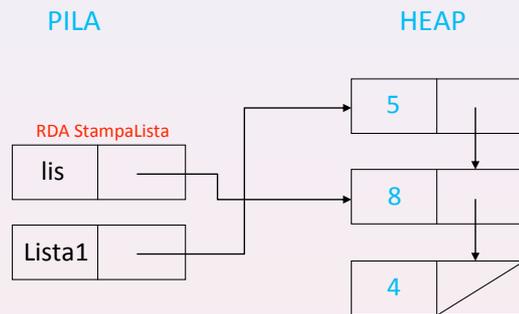
```



```

void StampaLista(ListaDiElementi lis)
{
    while (lis != NULL)
    {
        printf("%d -->", lis->info);
        lis = lis->next;
    }
    printf("//");
}

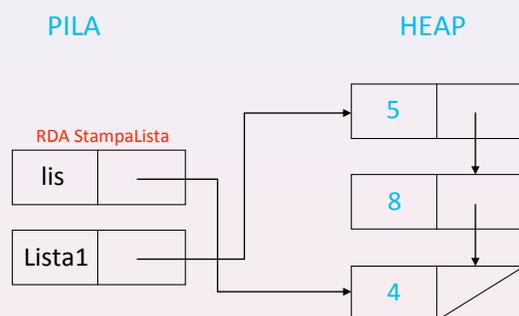
main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(Lista1);
    ...
}
    
```



```

void StampaLista(ListaDiElementi lis)
{
    while (lis != NULL)
    {
        printf("%d -->", lis->info);
        lis = lis->next;
    }
    printf("//");
}

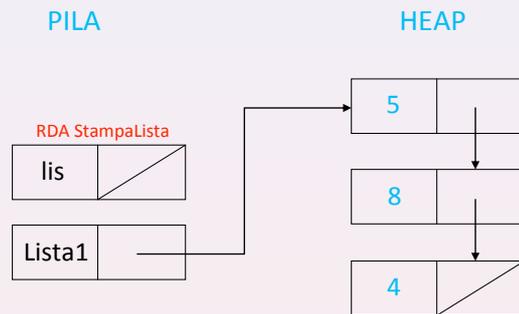
main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(Lista1);
    ...
}
    
```



```

void StampaLista(ListaDiElementi lis)
{
    while (lis != NULL)
    {
        printf("%d -->", lis->info);
        lis = lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(Lista1);
    ...
}
    
```



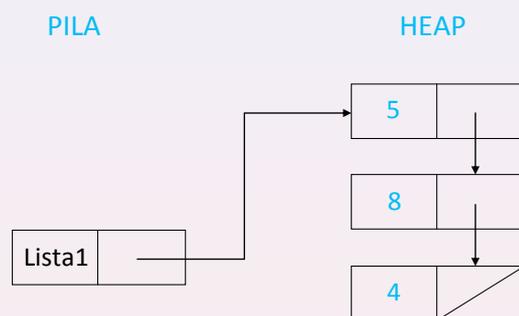
Output

5 --> 8 --> 4 --> //

```

void StampaLista(ListaDiElementi lis)
{
    while (lis != NULL)
    {
        printf("%d -->", lis->info);
        lis = lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(Lista1);
    ...
}
    
```



Output

5 --> 8 --> 4 --> //

Cosa sarebbe successo passando il parametro per indirizzo?

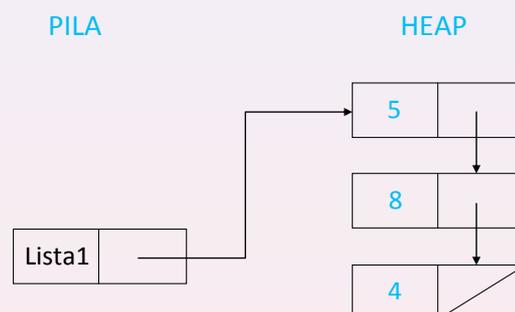
```
void StampaLista(ListaDiElementi *lis)
{
    while (*lis != NULL)
    {
        printf("%d -->", *lis->info);
        *lis = *lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(&Lista1);
    ...
}
```

Cosa sarebbe successo passando il parametro per indirizzo?

```
void StampaLista(ListaDiElementi *lis)
{
    while (*lis != NULL)
    {
        printf("%d -->", *lis->info);
        *lis = *lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(&Lista1);
    ...
}
```

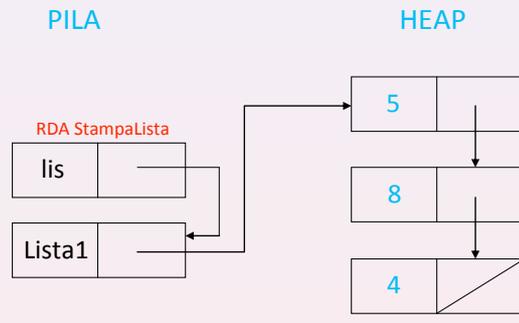


### Cosa sarebbe successo passando il parametro per indirizzo?

```

void StampaLista(ListaDiElementi *lis)
{
    while (*lis != NULL)
    {
        printf("%d -->", *lis->info);
        *lis = *lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(&Lista1);
    ...
}
    
```

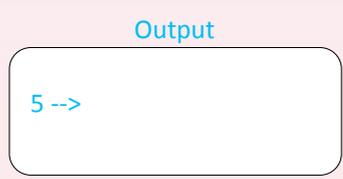
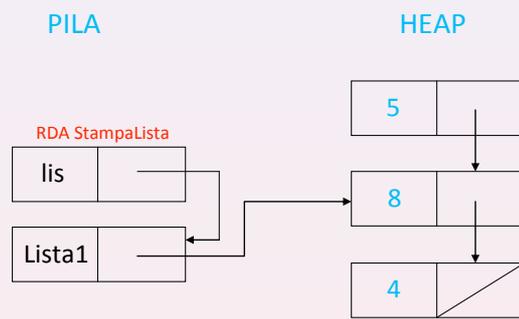


### Cosa sarebbe successo passando il parametro per indirizzo?

```

void StampaLista(ListaDiElementi *lis)
{
    while (*lis != NULL)
    {
        printf("%d -->", *lis->info);
        *lis = *lis->next;
    }
    printf("//");
}

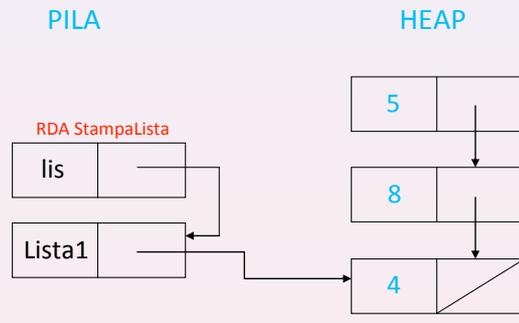
main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(&Lista1);
    ...
}
    
```



### Cosa sarebbe successo passando il parametro per indirizzo?

```
void StampaLista(ListaDiElementi *lis)
{
    while (*lis != NULL)
    {
        printf("%d -->", *lis->info);
        *lis = *lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(&Lista1);
    ...
}
```

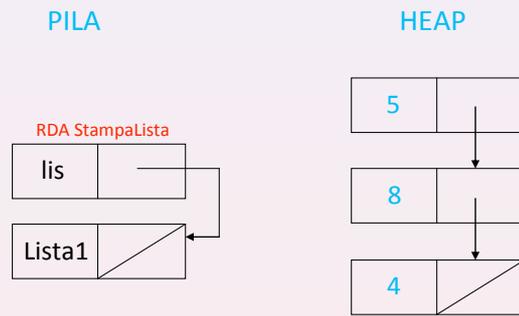


Output  
5 --> 8 -->

### Cosa sarebbe successo passando il parametro per indirizzo?

```
void StampaLista(ListaDiElementi *lis)
{
    while (*lis != NULL)
    {
        printf("%d -->", *lis->info);
        *lis = *lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(&Lista1);
    ...
}
```

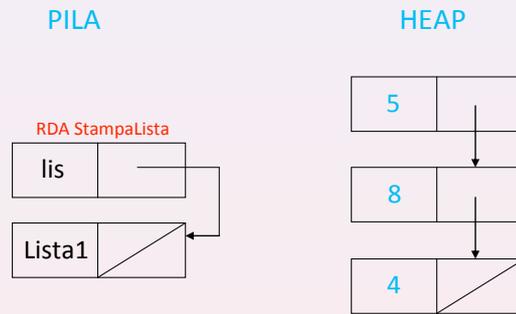


Output  
5 --> 8 --> 4 -->

### Cosa sarebbe successo passando il parametro per indirizzo?

```
void StampaLista(ListaDiElementi *lis)
{
    while (*lis != NULL)
    {
        printf("%d -->", *lis->info);
        *lis = *lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(&Lista1);
    ...
}
```

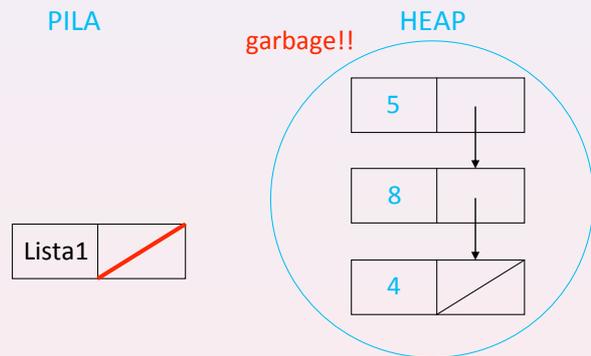


Output  
5 --> 8 --> 4 --> //

### Cosa sarebbe successo passando il parametro per indirizzo?

```
void StampaLista(ListaDiElementi *lis)
{
    while (*lis != NULL)
    {
        printf("%d -->", *lis->info);
        *lis = *lis->next;
    }
    printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaLista(&Lista1);
    ...
}
```



Output  
5 --> 8 --> 4 --> //

## Versione ricorsiva

Se la lista non è vuota stampa il primo elemento, quindi stampa la lista rimanente

```
void StampaListaRic(ListaDiElementi lis)
{
    if (lis != NULL)
    {
        printf("%d ", lis->info);
        StampaListaRic(lis->next);
    }
    else
        printf("//");
}

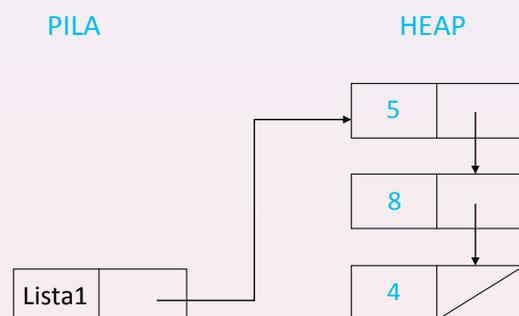
main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaListaRic(Lista1);
    ...
}
```

## Versione ricorsiva

Se la lista non è vuota stampa il primo elemento, quindi stampa la lista rimanente

```
void StampaListaRic(ListaDiElementi lis)
{
    if (lis != NULL)
    {
        printf("%d ", lis->info);
        StampaListaRic(lis->next);
    }
    else
        printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaListaRic(Lista1);
    ...
}
```



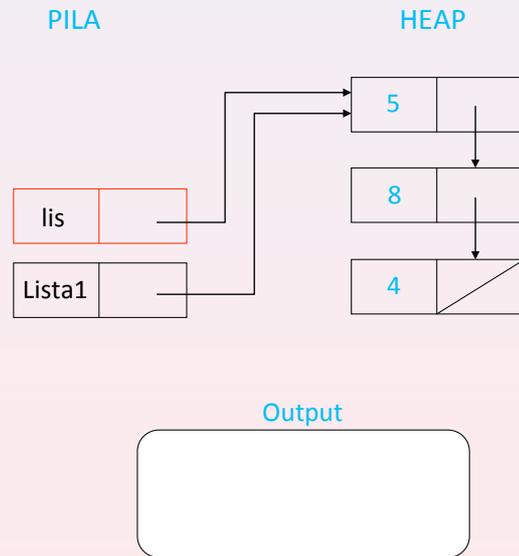
Output

## Versione ricorsiva

Se la lista non è vuota stampa il primo elemento, quindi stampa la lista rimanente

```
void StampaListaRic(ListaDiElementi lis)
{
    if (lis != NULL)
    {
        printf("%d ", lis->info);
        StampaListaRic(lis->next);
    }
    else
        printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaListaRic(Lista1);
    ...
}
```

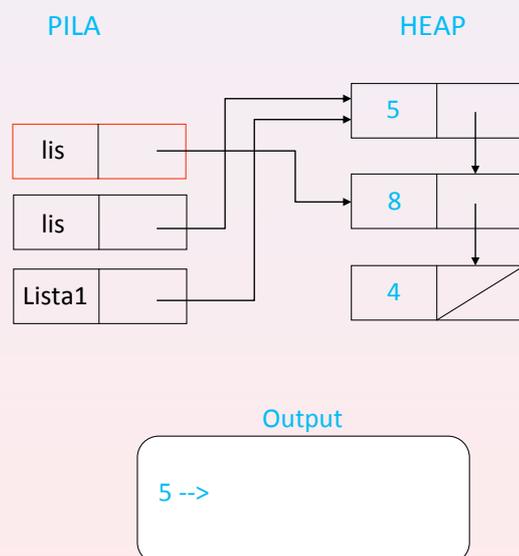


## Versione ricorsiva

Se la lista non è vuota stampa il primo elemento, quindi stampa la lista rimanente

```
void StampaListaRic(ListaDiElementi lis)
{
    if (lis != NULL)
    {
        printf("%d ", lis->info);
        StampaListaRic(lis->next);
    }
    else
        printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaListaRic(Lista1);
    ...
}
```

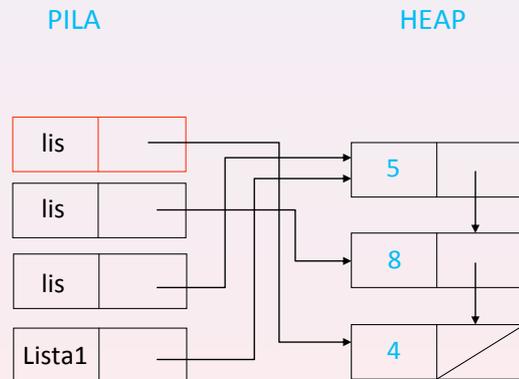


## Versione ricorsiva

Se la lista non è vuota stampa il primo elemento, quindi stampa la lista rimanente

```
void StampaListaRic(ListaDiElementi lis)
{
    if (lis != NULL)
    {
        printf("%d ", lis->info);
        StampaListaRic(lis->next);
    }
    else
        printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaListaRic(Lista1);
    ...
}
```



Output

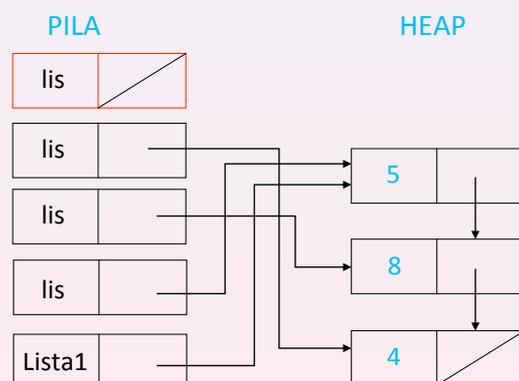
5 --&gt; 8 --&gt;

## Versione ricorsiva

Se la lista non è vuota stampa il primo elemento, quindi stampa la lista rimanente

```
void StampaListaRic(ListaDiElementi lis)
{
    if (lis != NULL)
    {
        printf("%d ", lis->info);
        StampaListaRic(lis->next);
    }
    else
        printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaListaRic(Lista1);
    ...
}
```



Output

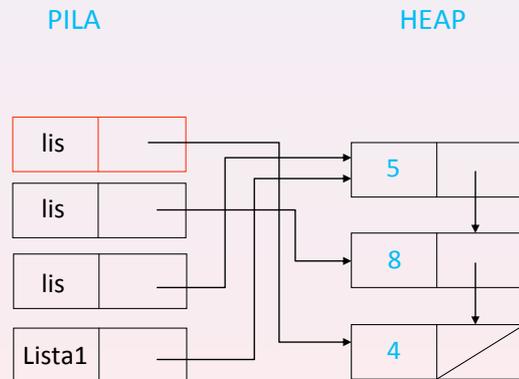
5 --&gt; 8 --&gt; 4 --&gt;

## Versione ricorsiva

Se la lista non è vuota stampa il primo elemento, quindi stampa la lista rimanente

```
void StampaListaRic(ListaDiElementi lis)
{
    if (lis != NULL)
    {
        printf("%d ", lis->info);
        StampaListaRic(lis->next);
    }
    else
        printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaListaRic(Lista1);
    ...
}
```



Output

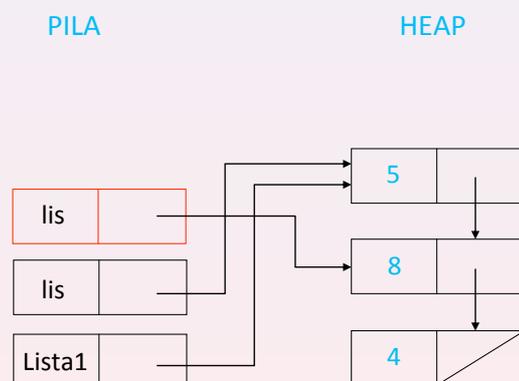
```
5 --> 8 --> 4 --> //
```

## Versione ricorsiva

Se la lista non è vuota stampa il primo elemento, quindi stampa la lista rimanente

```
void StampaListaRic(ListaDiElementi lis)
{
    if (lis != NULL)
    {
        printf("%d ", lis->info);
        StampaListaRic(lis->next);
    }
    else
        printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaListaRic(Lista1);
    ...
}
```



Output

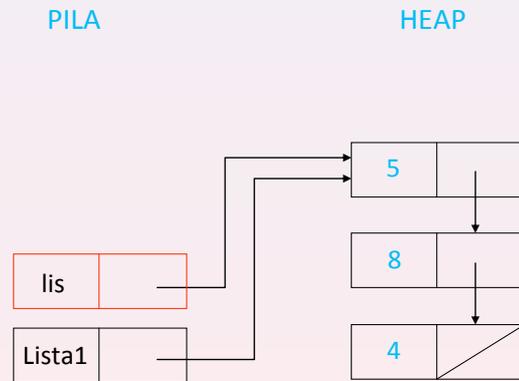
```
5 --> 8 --> 4 --> //
```

## Versione ricorsiva

Se la lista non è vuota stampa il primo elemento, quindi stampa la lista rimanente

```
void StampaListaRic(ListaDiElementi lis)
{
    if (lis != NULL)
    {
        printf("%d ", lis->info);
        StampaListaRic(lis->next);
    }
    else
        printf("//");
}

main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaListaRic(Lista1);
    ...
}
```



Output

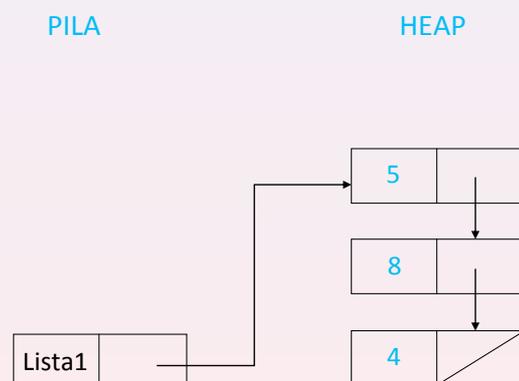
5 --&gt; 8 --&gt; 4 --&gt; //

## Versione ricorsiva

Se la lista non è vuota stampa il primo elemento, quindi stampa la lista rimanente

```
void StampaListaRic(ListaDiElementi lis)
{
    if (lis != NULL)
    {
        printf("%d ", lis->info);
        StampaListaRic(lis->next);
    }
    else
        printf("//");
}

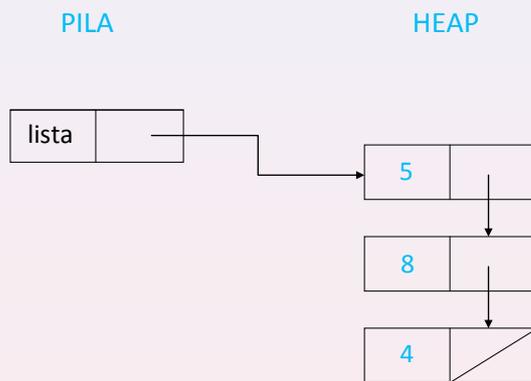
main()
{
    ListaDiElementi Lista1;
    ...
    /* costruzione lista 5 --> 8 --> 4 */
    ...
    StampaListaRic(Lista1);
    ...
}
```



Output

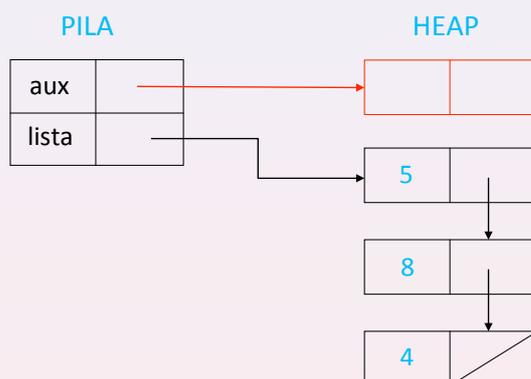
5 --&gt; 8 --&gt; 4 --&gt; //

## Inserimento di un nuovo elemento in testa



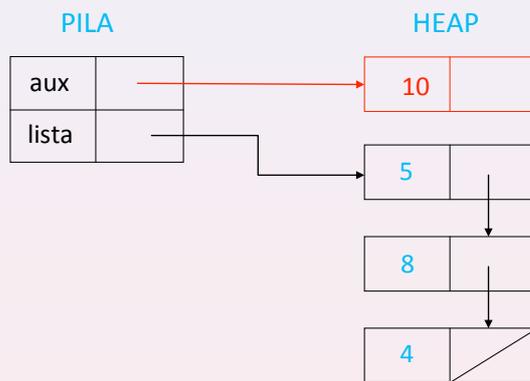
1. allochiamo una nuova struttura per l'elemento (**malloc**)
2. assegniamo il valore da inserire al campo **info** della struttura
3. concateniamo la nuova struttura con la vecchia lista
4. il puntatore iniziale della lista punta alla nuova struttura  
 ⇒ la lista da modificare deve essere passata per **indirizzo**

## Inserimento di un nuovo elemento in testa



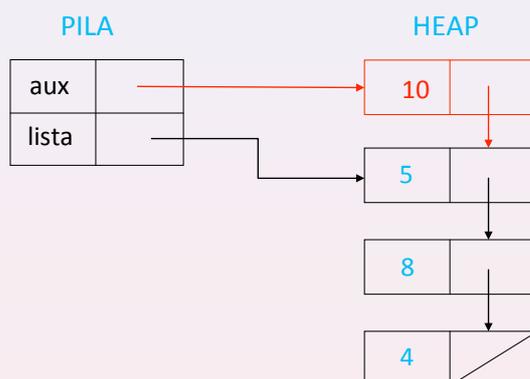
1. allochiamo una nuova struttura per l'elemento (**malloc**)
2. assegniamo il valore da inserire al campo **info** della struttura
3. concateniamo la nuova struttura con la vecchia lista
4. il puntatore iniziale della lista punta alla nuova struttura  
 ⇒ la lista da modificare deve essere passata per **indirizzo**

## Inserimento di un nuovo elemento in testa



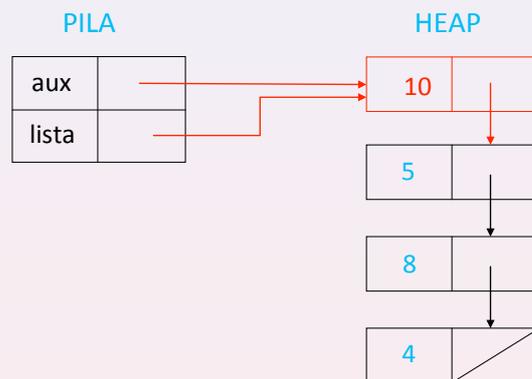
1. allochiamo una nuova struttura per l'elemento (`malloc`)
2. assegniamo il valore da inserire al campo `info` della struttura
3. concateniamo la nuova struttura con la vecchia lista
4. il puntatore iniziale della lista punta alla nuova struttura  
 ⇒ la lista da modificare deve essere passata per `indirizzo`

## Inserimento di un nuovo elemento in testa



1. allochiamo una nuova struttura per l'elemento (`malloc`)
2. assegniamo il valore da inserire al campo `info` della struttura
3. concateniamo la nuova struttura con la vecchia lista
4. il puntatore iniziale della lista punta alla nuova struttura  
 ⇒ la lista da modificare deve essere passata per `indirizzo`

## Inserimento di un nuovo elemento in testa



1. allochiamo una nuova struttura per l'elemento (`malloc`)
2. assegniamo il valore da inserire al campo `info` della struttura
3. concateniamo la nuova struttura con la vecchia lista
4. il puntatore iniziale della lista punta alla nuova struttura  
 ⇒ la lista da modificare deve essere passata per `indirizzo`

```
void InserisciTestaLista(ListaDiElementi *lista, int elem)
{
    ListaDiElementi aux;

    aux = malloc(sizeof(ElementoLista));
    aux->info = elem;
    aux->next = *lista;
    *lista = aux;
}
```

- ▶ il primo parametro è la lista da modificare (passata per indirizzo)
- ▶ il secondo parametro è l'elemento da inserire (passato per indirizzo)
  - ▶ Attenzione: nel caso di liste di tipo `TipoElemLista` la procedura può essere generalizzata se su tale tipo è definito l'assegnamento

### Esercizio

Impostare una chiamata alla procedura e tracciare l'evoluzione di pila e heap

```

void InserisciTestaLista(ListaDiElementi *lista, TipoElemLista elem)
{
    ListaDiElementi aux;

    aux = malloc(sizeof(ElementoLista));
    aux->info = elem;
    aux->next = *lista;
    *lista = aux;
}

```

- ▶ il primo parametro è la lista da modificare (passata per indirizzo)
- ▶ il secondo parametro è l'elemento da inserire (passato per indirizzo)
  - ▶ Attenzione: nel caso di liste di tipo `TipoElemLista` la procedura può essere generalizzata se su tale tipo è definito l'assegnamento

### Esercizio

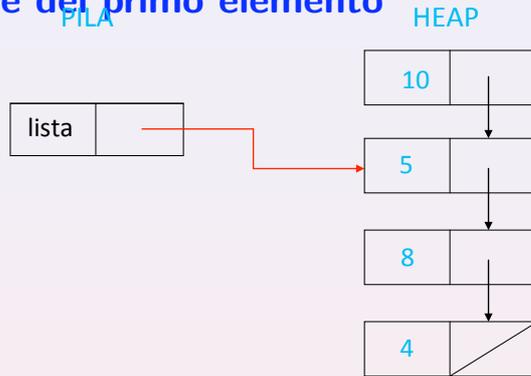
Impostare una chiamata alla procedura e tracciare l'evoluzione di pila e heap

## Cancellazione del primo elemento



- ▶ se la lista è vuota non facciamo nulla
- ▶ altrimenti eliminiamo il primo elemento
  - ⇒ la lista deve essere passata per indirizzo
- ▶ **cancellare** significa anche **deallocare** la memoria occupata dall'elemento
  - ⇒ dobbiamo invocare `free` passando il puntatore all'elemento da cancellare
  - ⇒ è necessario un puntatore ausiliario

## Cancellazione del primo elemento



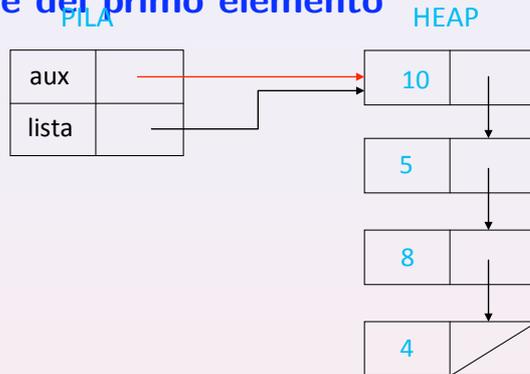
- ▶ se la lista è vuota non facciamo nulla
- ▶ altrimenti eliminiamo il primo elemento
  - ⇒ la lista deve essere passata per indirizzo
- ▶ **cancellare** significa anche **deallocare** la memoria occupata dall'elemento
  - ⇒ dobbiamo invocare **free** passando il puntatore all'elemento da cancellare
  - ⇒ è necessario un puntatore ausiliario

## Cancellazione del primo elemento



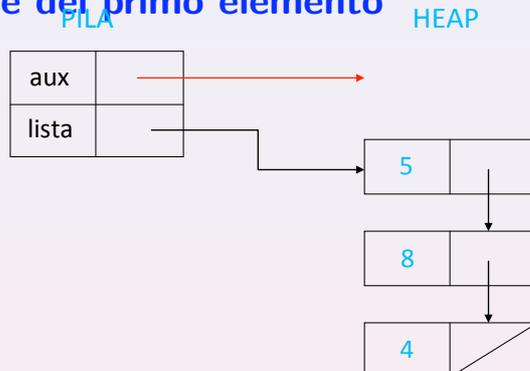
- ▶ se la lista è vuota non facciamo nulla
- ▶ altrimenti eliminiamo il primo elemento
  - ⇒ la lista deve essere passata per indirizzo
- ▶ **cancellare** significa anche **deallocare** la memoria occupata dall'elemento
  - ⇒ dobbiamo invocare **free** passando il puntatore all'elemento da cancellare
  - ⇒ è necessario un puntatore ausiliario

## Cancellazione del primo elemento



- ▶ se la lista è vuota non facciamo nulla
- ▶ altrimenti eliminiamo il primo elemento
  - ⇒ la lista deve essere passata per indirizzo
- ▶ **cancellare** significa anche **deallocare** la memoria occupata dall'elemento
  - ⇒ dobbiamo invocare **free** passando il puntatore all'elemento da cancellare
  - ⇒ è necessario un puntatore ausiliario

## Cancellazione del primo elemento



- ▶ se la lista è vuota non facciamo nulla
- ▶ altrimenti eliminiamo il primo elemento
  - ⇒ la lista deve essere passata per indirizzo
- ▶ **cancellare** significa anche **deallocare** la memoria occupata dall'elemento
  - ⇒ dobbiamo invocare **free** passando il puntatore all'elemento da cancellare
  - ⇒ è necessario un puntatore ausiliario

```
void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
        {
            aux = *lista;
            *lista = (*lista)->next;
            free(aux);
        }
}
```

## Cancellazione di tutta una lista

```
void CancellaLista(ListaDiElementi *lista)
{
    ListaDiElementi aux;

    while (*lista != NULL) {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}
```

- Osserviamo che il corpo del ciclo corrisponde alle azioni della procedura `CancellaPrimo`. Possiamo allora scrivere:

```
void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}
```

- Si noti il parametro attuale della chiamata a `CancellaPrimo`, che è `lista` (di tipo `ListaDiElementi *`) e non `&lista`

```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

```

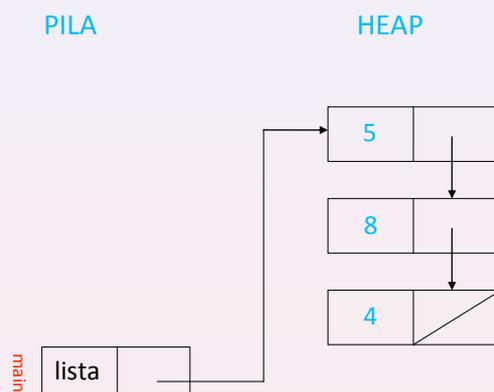
```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

```



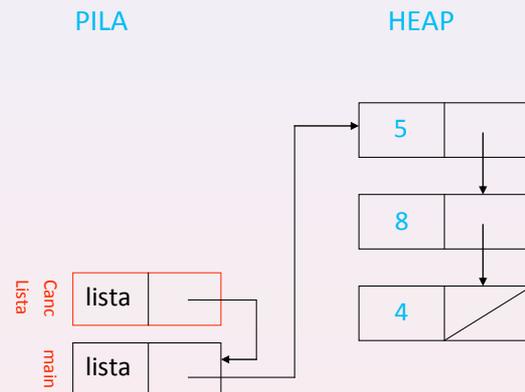
```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

```



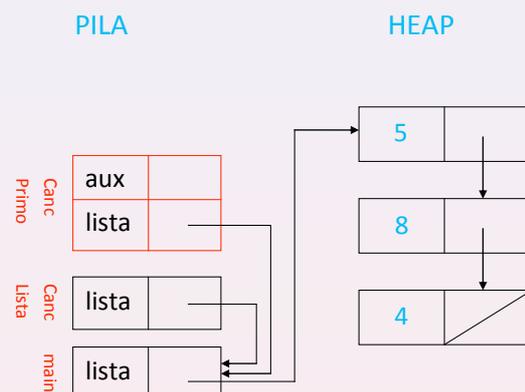
```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

```



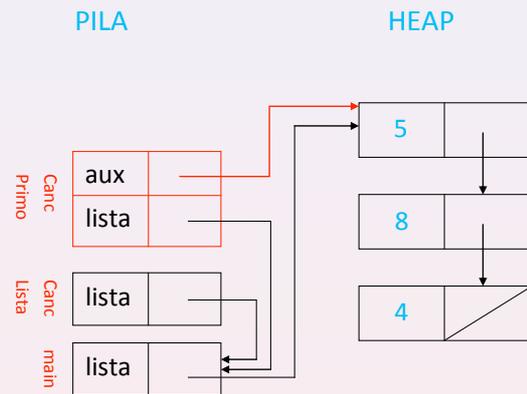
```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

```



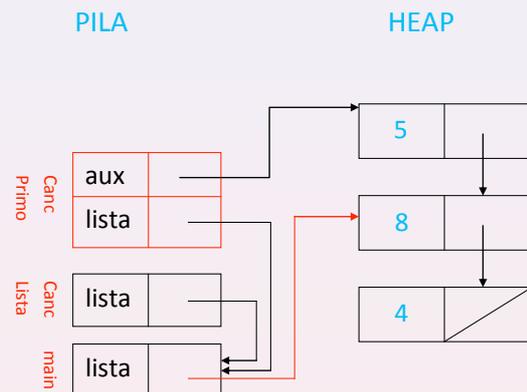
```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

```



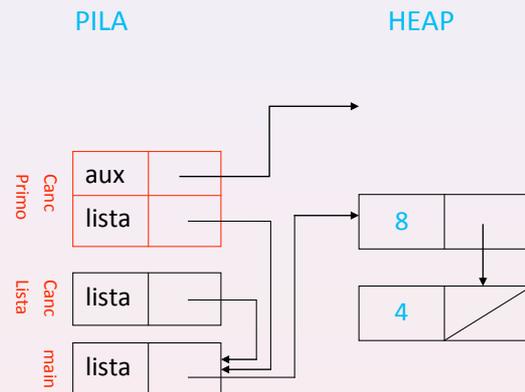
```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

```



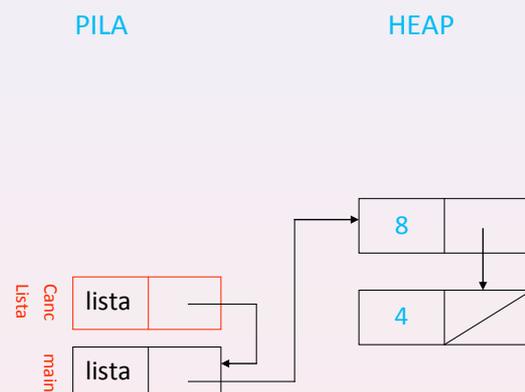
```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

```

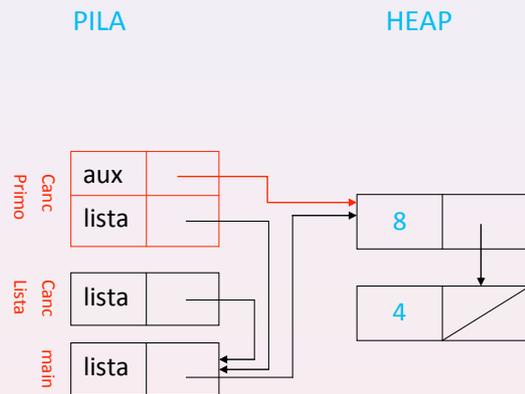


```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}
    
```

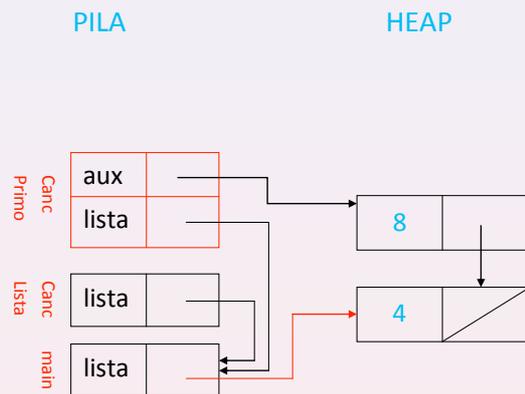


```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}
    
```



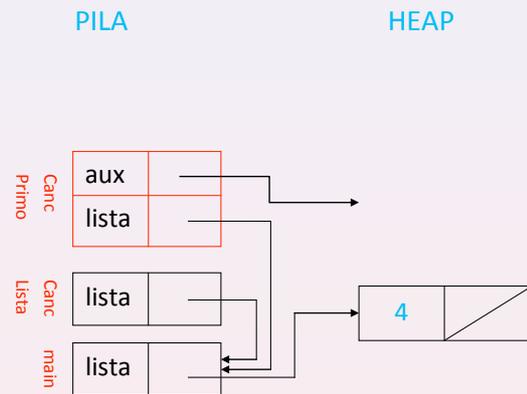
```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

```



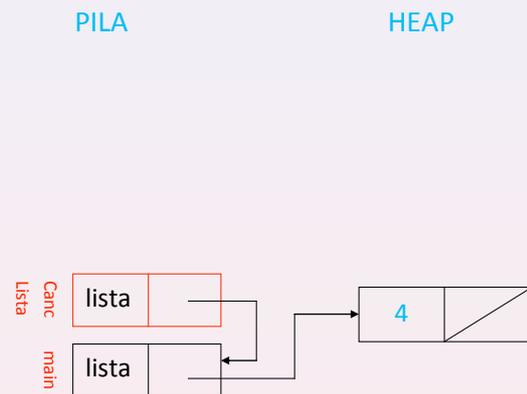
```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

```

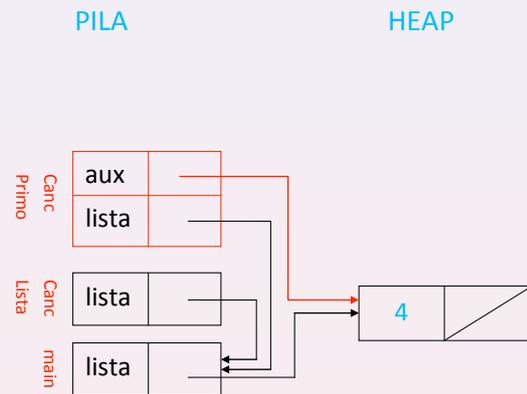


```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}
    
```

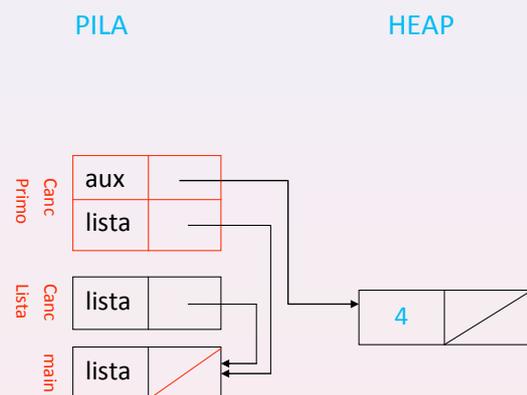


```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}
    
```

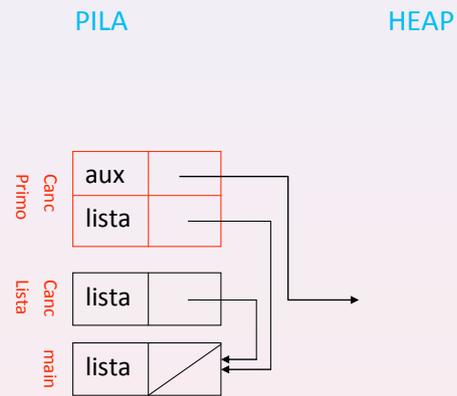


```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}
    
```

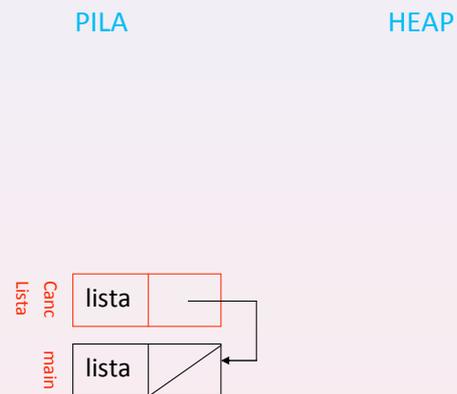


```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}
    
```



```

void CancellaPrimo(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
    }
}

```

PILA

HEAP

```

void CancellaLista(ListaDiElementi *lista)
{
    while (*lista != NULL)
        CancellaPrimo(lista);
}

```



```

main()
{
    ListaDiElementi lista;
    ...
    CancellaLista(&lista);
    ...
}

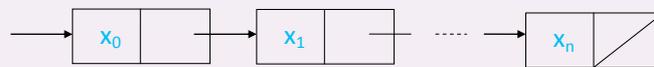
```

## Visione ricorsiva delle liste



- ▶ Una lista di elementi è una struttura dati ricorsiva per sua natura
  1. data una lista  $L$  di elementi  $x_1, \dots, x_n$
  2. dato un ulteriore elemento  $x_0$
  3. anche la **concatenazione** di  $x_0$  e  $L$  è una lista
- ▶ Si noti che in 1.  $L$  può anche essere la lista vuota

## Visione ricorsiva delle liste



- ▶ Una lista di elementi è una struttura dati ricorsiva per sua natura
  1. data una lista  $L$  di elementi  $x_1, \dots, x_n$
  2. dato un ulteriore elemento  $x_0$
  3. anche la **concatenazione** di  $x_0$  e  $L$  è una lista
- ▶ Si noti che in 1.  $L$  può anche essere la lista vuota

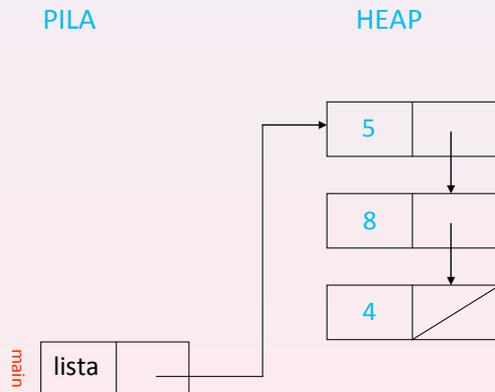
## Cancellazione lista: versione ricorsiva

- ▶ Sfruttiamo la visione ricorsiva della struttura dati lista per realizzare la cancellazione in modo **ricorsivo**
  1. la cancellazione della lista vuota non richiede alcuna azione
  2. la cancellazione della lista ottenuta come concatenazione dell'elemento  $x$  e della lista  $L$  richiede l'eliminazione di  $x$  e la cancellazione di  $L$
- ▶ la traduzione in C è immediata

```
void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
```

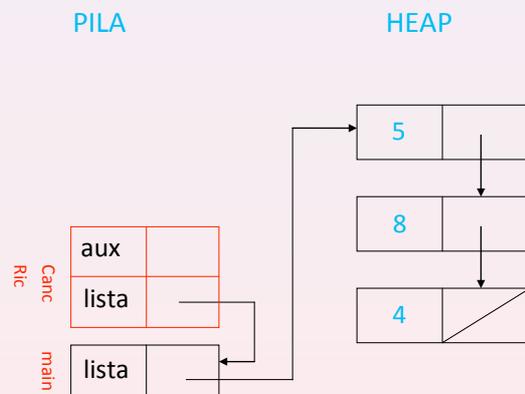
```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
    
```



```

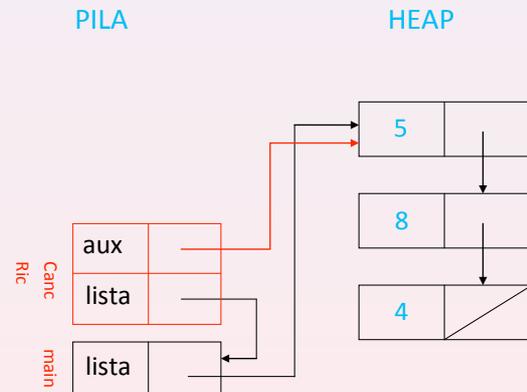
void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
    
```



```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}

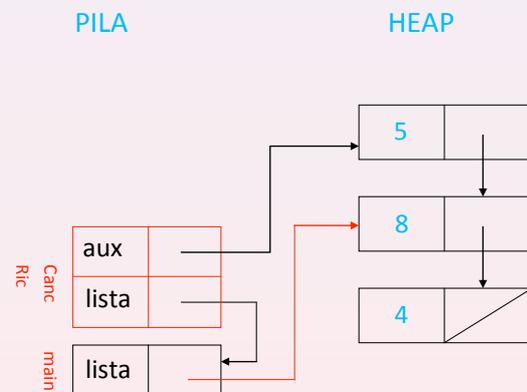
```



```

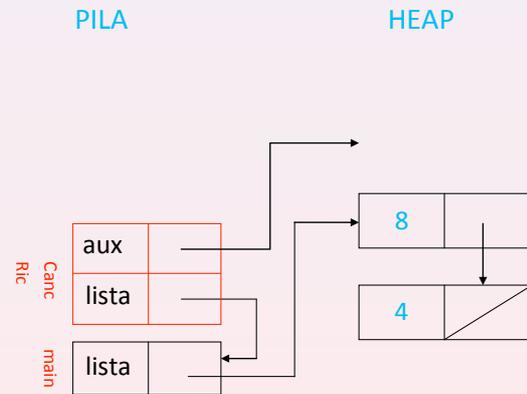
void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}

```



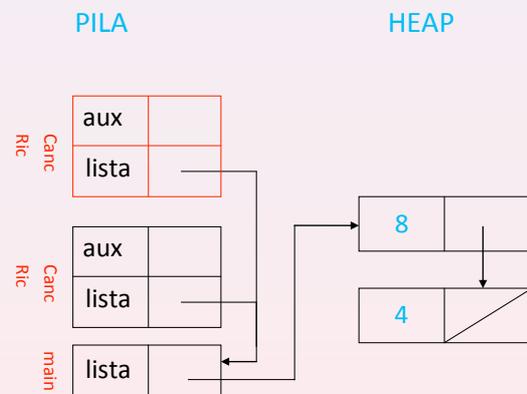
```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
    
```



```

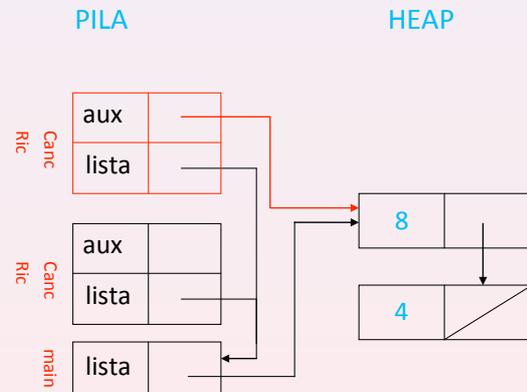
void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
    
```



```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}

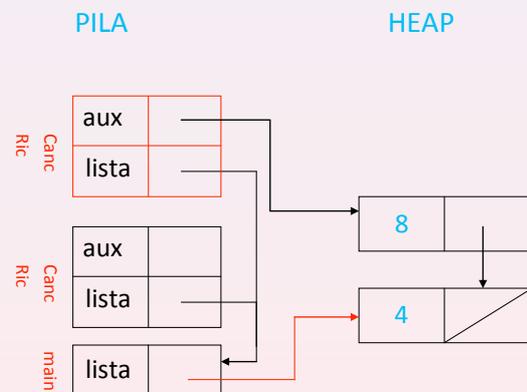
```



```

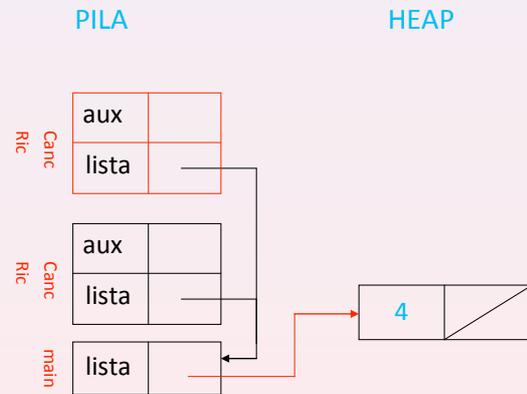
void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}

```



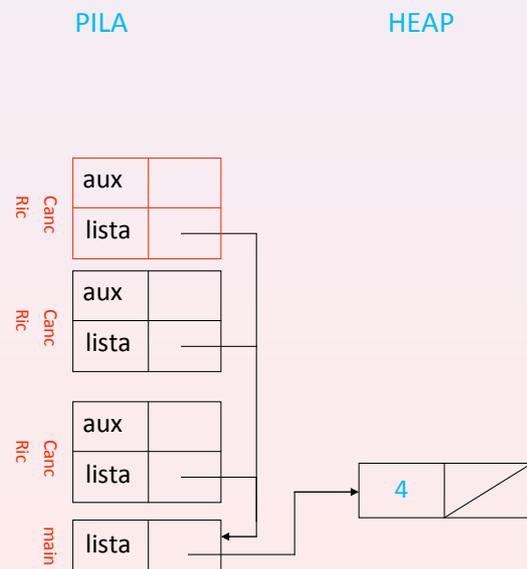
```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
    
```



```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
    
```

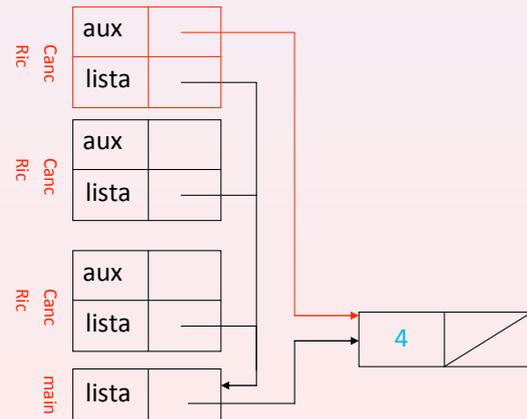


```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
    
```

PILA

HEAP

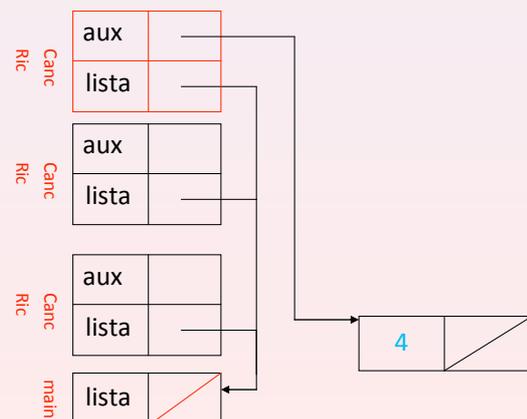


```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
    
```

PILA

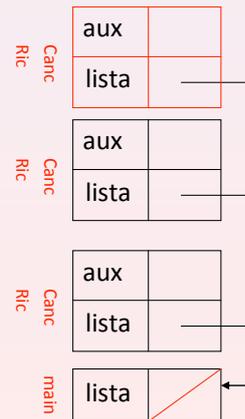
HEAP



```
void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
```

PILA

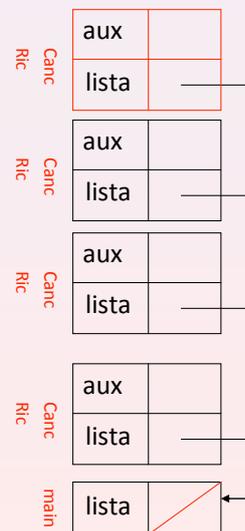
HEAP



```
void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
```

PILA

HEAP

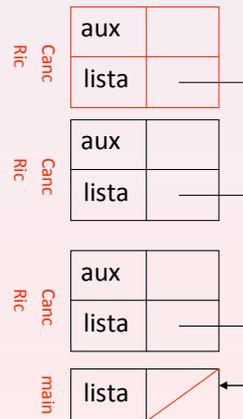


```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
    
```

PILA

HEAP

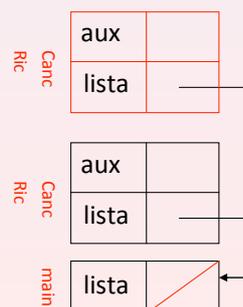


```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}
    
```

PILA

HEAP



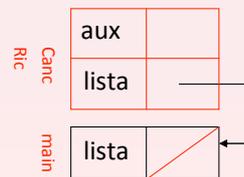
```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}

```

PILA

HEAP



```

void CancellaListaRic(ListaDiElementi *lista)
{
    ListaDiElementi aux;
    if (*lista != NULL)
    {
        aux = *lista;
        *lista = (*lista)->next;
        free(aux);
        CancellaListaRic(lista);
    }
}

```

PILA

HEAP

