

# Parte 1 - Liste

## Liste doppia



R. Magritte – Transfer,  
1966

# Liste doppie

- Lista *singolarmente concatenata* o *semplice (singly linked list)*: ciascun elemento contiene solo un puntatore al prossimo elemento
- Lista *doppiamente concatenata* o *doppia*: ciascun elemento contiene due puntatori
  - un puntatore al prossimo elemento
  - un puntatore all'elemento precedente
- Con le liste doppie è possibile **scorrere** la lista degli elementi nelle due direzioni

# Schema lista doppia

Ciascun elemento punta sia al precedente che al successivo



A partire da un elemento è possibile accedere a tutti gli altri elementi della lista attraverso

- La sequenza di puntatori all'elemento precedente
- La sequenza di puntatori all'elemento successivo

# Liste semplici vs. liste doppie

Vogliamo implementare il tipo lista doppia.

Cosa cambia rispetto all'implementazione delle liste semplici?


- Il tipo di dato?

*Si, dobbiamo aggiungere un puntatore*

- Le primitive?

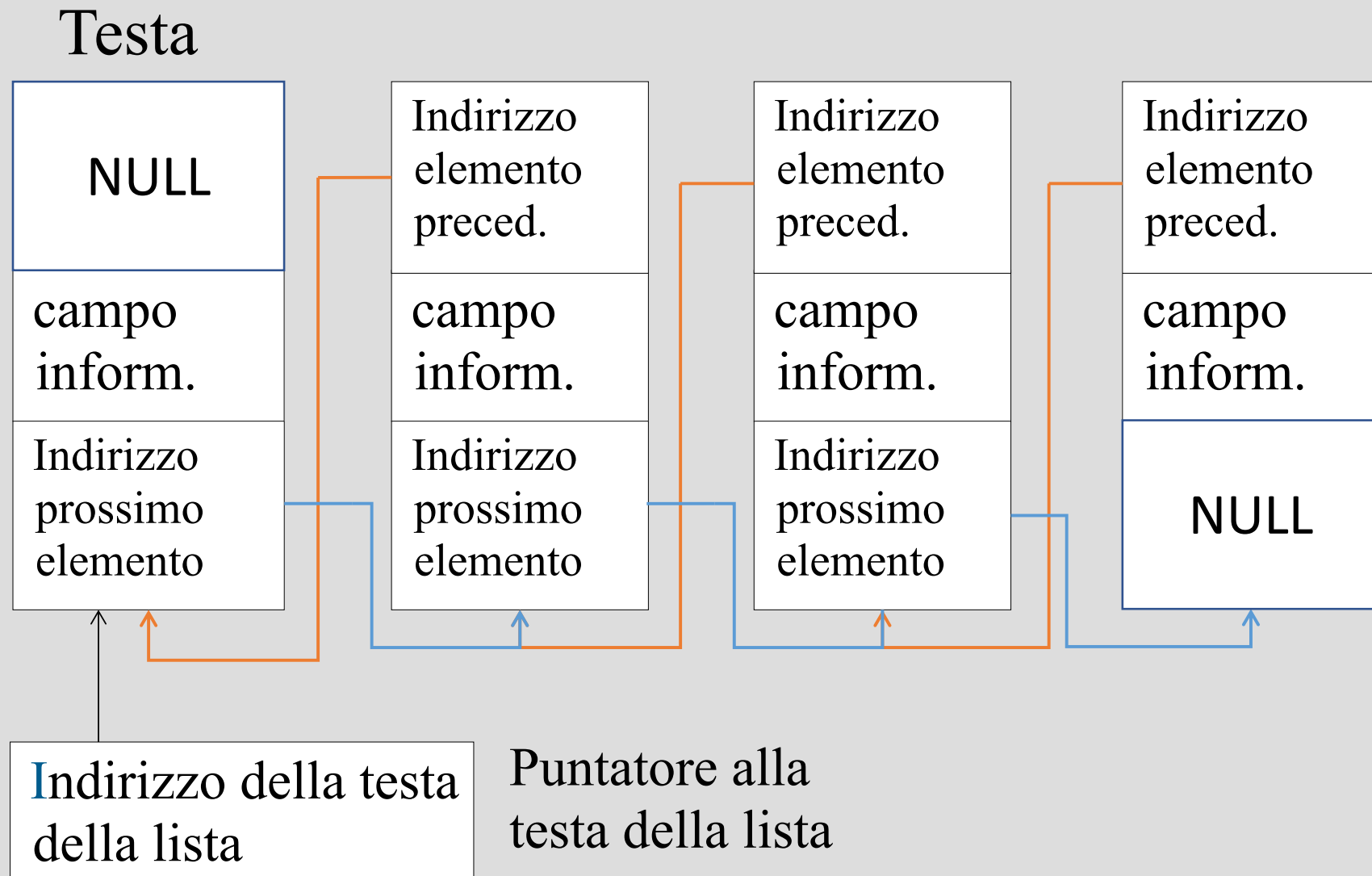
*L'interfaccia delle primitive non cambia. Cambia l'implementazione di alcune primitive. Quali?*

- Le primitive che agiscono sui puntatori: **insert\_elem**  
**delete\_elem** **copy**...

 Vediamo il caso delle primitive **insert\_elem** e **delete\_elem**

- Aggiungiamo la primitive **lista\_prev(lista)** che restituisce la lista corrispondente all'elemento che precede l'element in input

# Schema lista doppia

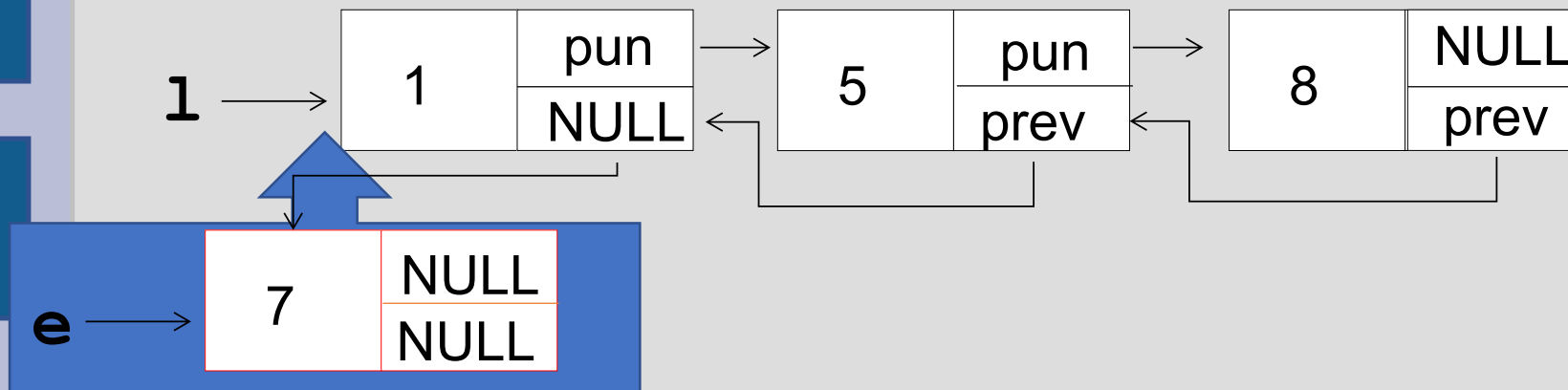


# Struttura dati

```
struct elem {  
    int inf;  
    elem* pun; // punt. al prossimo elem  
    elem* prec; // punt. al precedente elem  
} ;  
  
typedef elem* lista;
```

# insert\_elem

## Cosa cambia



Cosa cambia in caso di liste doppie?

```
lista insert_elem(lista l, elem* e){  
    e->pun=l;  
    if(l!=NULL)  
        l->prev=e;  
    e->prev=NULL;  
    return e;  
}
```

Dobbiamo aggiornare anche i puntatori  
prev

# delete\_elem:

**ATTENZIONE:** `delete_elem` può essere implementato in modo più efficiente perché, dato l'elemento da cancellare, è possibile accedere direttamente al suo predecessore nella lista!!



# delete\_elem: implementazioni a confronto

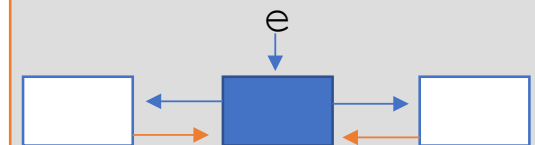
```
lista delete_elem(lista l, elem* e){  
  if (l==e)  
    l=tail(l);  
  else{  
    lista l1=l;  
    while (tail(l1)!=e)  
      l1=tail(l1);  
    l1->pun=tail(e);  
  }  
  delete e;  
  return l;}
```

## LISTA SEMPLICE

Ciclo per localizzare l'elemento  
che mi consente di tenere  
traccia del precedente (l1):  
Costo  $O(n)$

```
lista delete_elem(lista l, elem* e){  
  if(l==e)  
    l=tail(l); // e è la testa della lista  
  else // e non è la testa della lista  
    (e->prev)->pun = e->pun;  
  if(e->pun!=NULL)  
    (e->pun)->prev=e->prev;  
  delete e;  
  return l;}
```

## LISTA DOPPIA



Accedo direttamente a e  
e posso modificare  
precedente e successivo:  
Costo  $O(1)$

# Perché usare le liste doppie?

A partire da un qualsiasi element della lista, le liste doppie consentono di accedere a tutti gli elementi la lista

Alcune applicazioni delle linked list:

- Buona soluzione per rappresentare un mazzo di carte nei giochi on-line
- La cache del browser che consente di cliccare il bottone BACK e FORWARD (lista di URLs)
- Applicazioni che mantengono la liste dei Most Recently Used (MRU) (lista di nomi di file)
- Quasi tutte le applicazioni come Word e Photoshop consentono di eseguire UNDO e REDO (lista degli stati successivi)
- Implementazione di altre strutture dati come gli alberi di ricerca

# Codice per la gestione di liste doppie

Completare *gestione\_lista\_doppia.cc* per la gestione di liste doppie.

In particolare, il programma consente all'utente di accedere ad una lista di URL.

Il programma presenta un menù all'utente che può

- creare una lista di  $n$  URL— ogni URL ha al più 50 caratteri - attraverso la funzione `insert_elem()`
- cancellare un URL dato dalla lista attraverso la funzione `delete_elem()`
- stampare la lista attraverso `head()` e `tail()`
- cercare un URL nella lista e quindi consentire all'utente di muoversi a piacere avanti (F-forward) e indietro (B-backward) nella lista a partire dalla posizione del sito cercato fino a quando non digita S-stop, utilizzando `search()`

Per l'implementazione delle liste doppie a partire dalle liste semplici, è necessario re-implementare le *funzioni* **`insert_elem`** e **`delete_elem`**

**Vedi programma** *gestione\_lista\_doppia\_sol.cc*

# Altri tipi di liste

**Liste circolari:** il puntatore al prossimo elemento della coda della lista punta alla testa della lista stessa

**ESEMPIO** Lista circolare semplice

