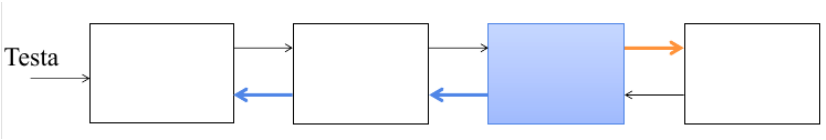
LISTE DOPPIE

A differenza di una lista singolarmente concatena o semplice (singly linked list) che contiene solo un puntatore al prossimo elemento una lista doppiamente concatenata o doppia contiene due puntatori, uno per il prossimo elemento e uno per il precedente. Ciò rende possibile lo scorrimento in due direzioni.

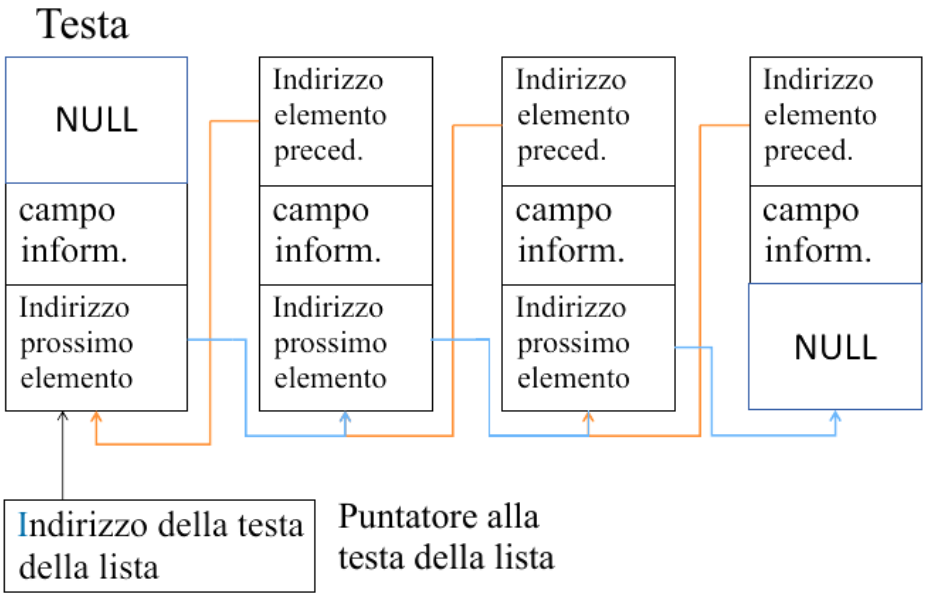
A partire da un elemento è quindi possibile accedere a tutti gli altri elementi della lista.



Rispetto all'implementazione delle liste semplici:

- nel tipo di dato bisogna aggiungere un puntatore.
- le primitive sono le stesse, cambia solo le implementazioni di alcune di esse (quelle su cui agiscono i puntatori).
    - insert:
    - cancellazione di un elemento
    - copia

SCHEMA LISTA DOPPIA



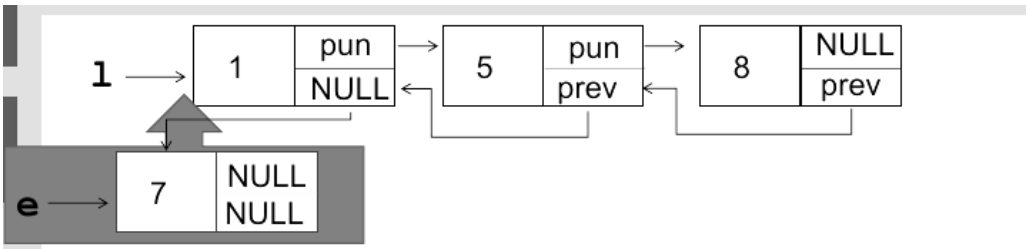
Le liste doppie e le liste semplici implementano lo stesso tipo di dato; cambiano il tipo di funzionalità date da ognuna di esse.

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

typedef elem* lista;
```

Inserimento in una lista doppia.

Va gestito il caso in cui l sia uguale a NULL (ossia la lista è vuota).



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`

L'implementazione della delete_elem è sicuramente più efficiente perchè, dato l'elemento da cancellare è possibile accedere direttamente al suo predecessore.

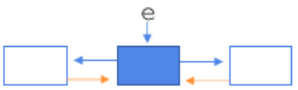
LISTA SEMPLICE

```
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;}
```

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

LISTA DOPPIA

```
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;}
```



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

Perchè usare le liste doppie? A partire da un qualsiasi elemento della lista, le liste doppie consentono di accedere a tutti gli elementi la lista

Vengono utilizzate per l'implementazione di altre strutture dati come gli alberi di ricerca.