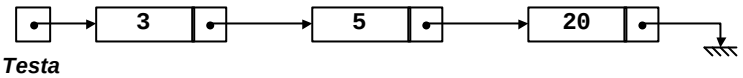
Liste Doppiamente Collegate

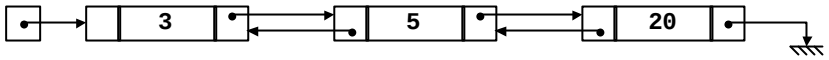
Consideriamo una lista semplicemente collegata…



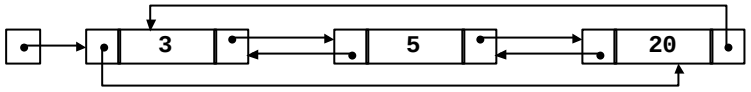
{3, 5, 20}

Testa della lista

Aggiungiamo ad ogni elemento un **puntatore all'elemento precedente** …

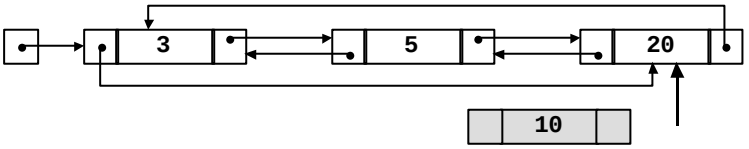


Chiudiamo, in modo **circolare**, la lista …

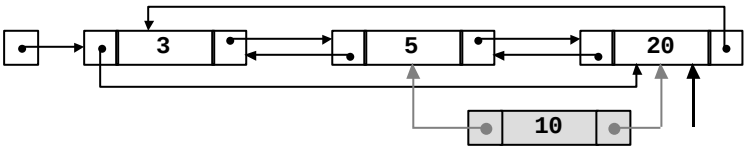


(da notare che non abbiamo più riferimenti a NULL)

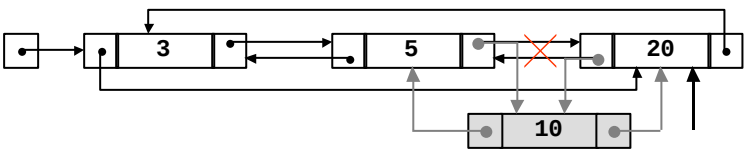
Liste Doppiamente Collegate: Inserimento in ordine



Ricerca del primo elemento in lista **maggiore** dell'elemento da inserire

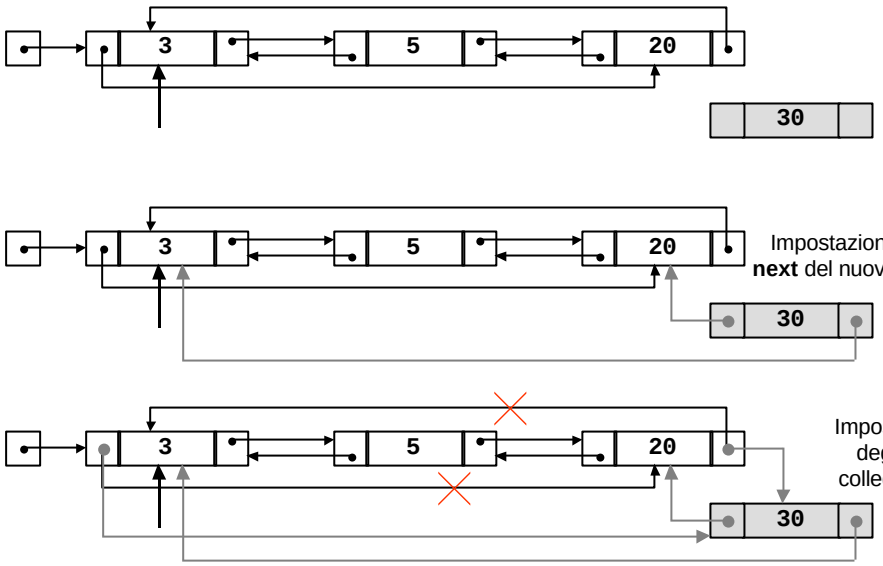


Impostazione **prev** e **next** del nuovo elemento



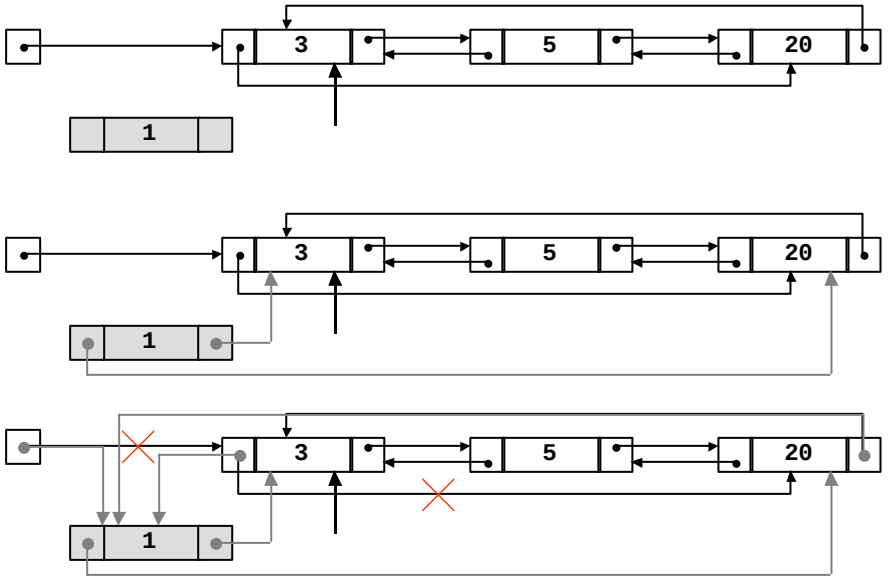
Impostazione degli altri collegamenti

Inserimento in ordine (ultimo elemento)



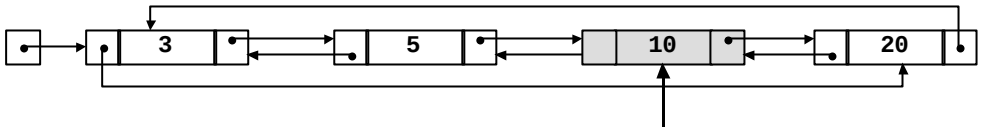
3

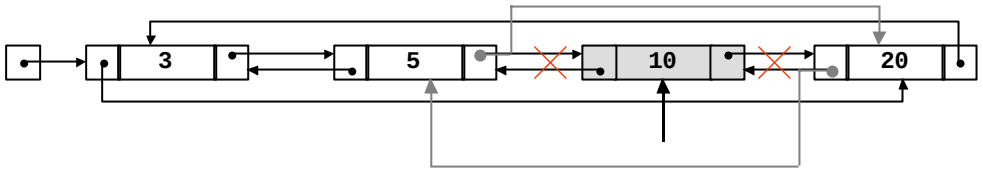
Inserimento in ordine (primo elemento)



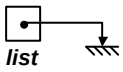
4

Liste Doppiamente Collegate: Cancellazione

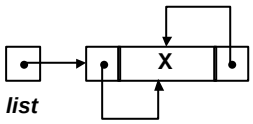




Liste Doppiamente Collegate: Lista vuota



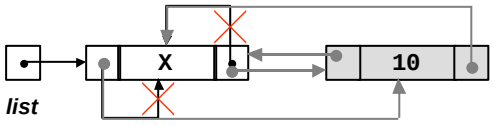
La testa punta a NULL: tuttavia dovremmo gestire questo caso particolare negli inserimenti e nelle cancellazioni



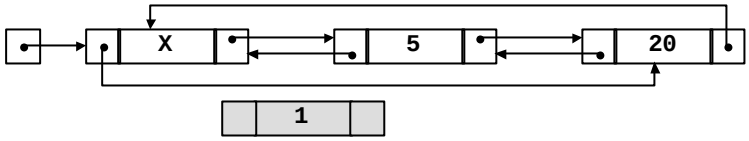
Mettiamo un "elemento fittizio"

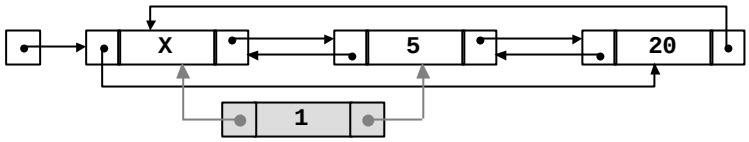
Lista vuota → `list->next == list->prev`

Inseriamo in testa …

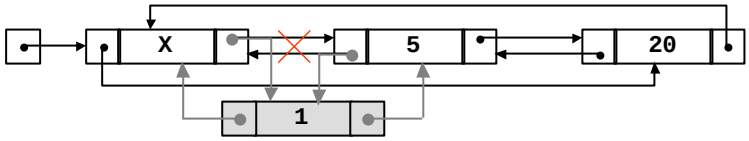


Inserimento in testa (con elemento fittizio)



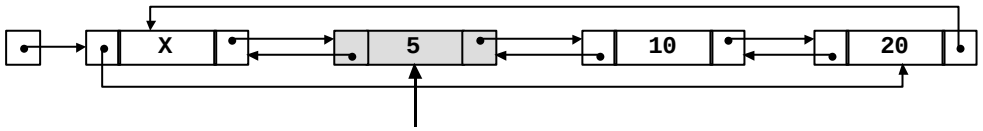


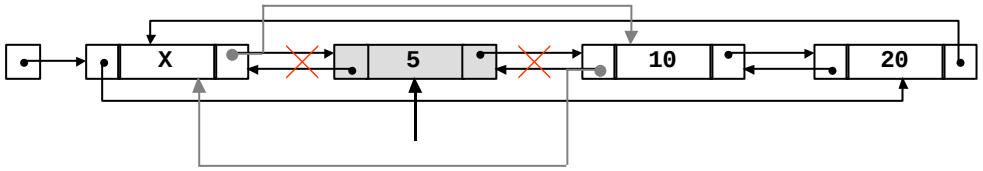
Impostazione **prev** e **next** del nuovo elemento



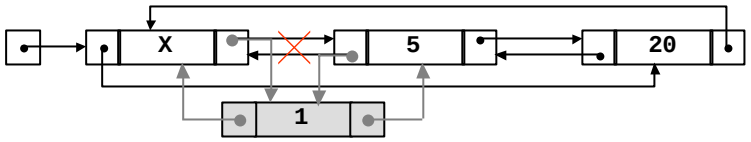
Impostazione degli altri collegamenti

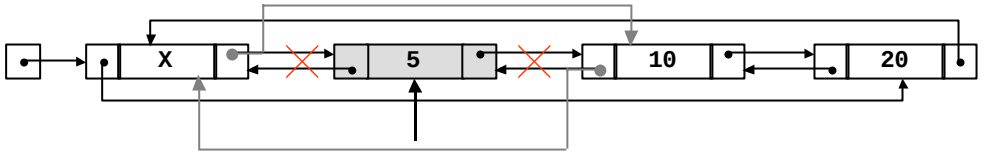
Cancellazione dalla testa (con elemento fittizio)

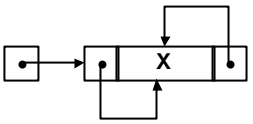




ESERCIZIO

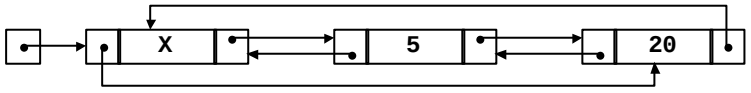




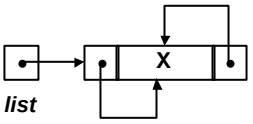


IMPLEMENTARE UNO STACK DI INTERI:
- Struttura dati
- Inizializzazione
- Inserimento in testa (push)
- Estrazione dalla testa (pop)
- Visita

Liste Doppiamente Collegate: Implementazione

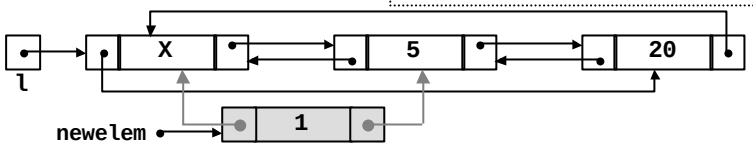



```
typedef struct t_list_element {
  int data;                                        // lista di interi
  struct t_list_element * next, * prev;
} t_list_element;

typedef t_list_element * t_list;

t_list init_list (void)
{
  t_list_element * p;
  p = (t_list_element *)malloc (sizeof (t_list_element));
  p->next = p;
  p->prev = p;
  return p;
}
...

t_list mylist;
mylist = init_list ();
```

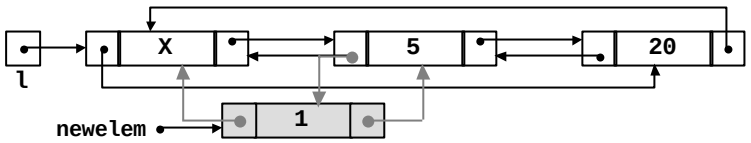


insert_at_head (t_list l, int data);

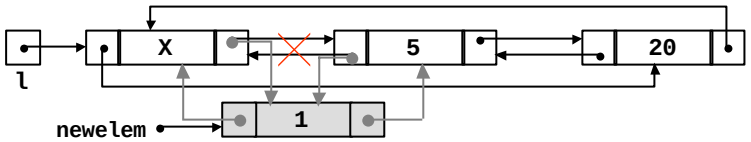
```
newelem->prev = l;
newelem->next = l->next;
```




```
l->next->prev = newelem;
```



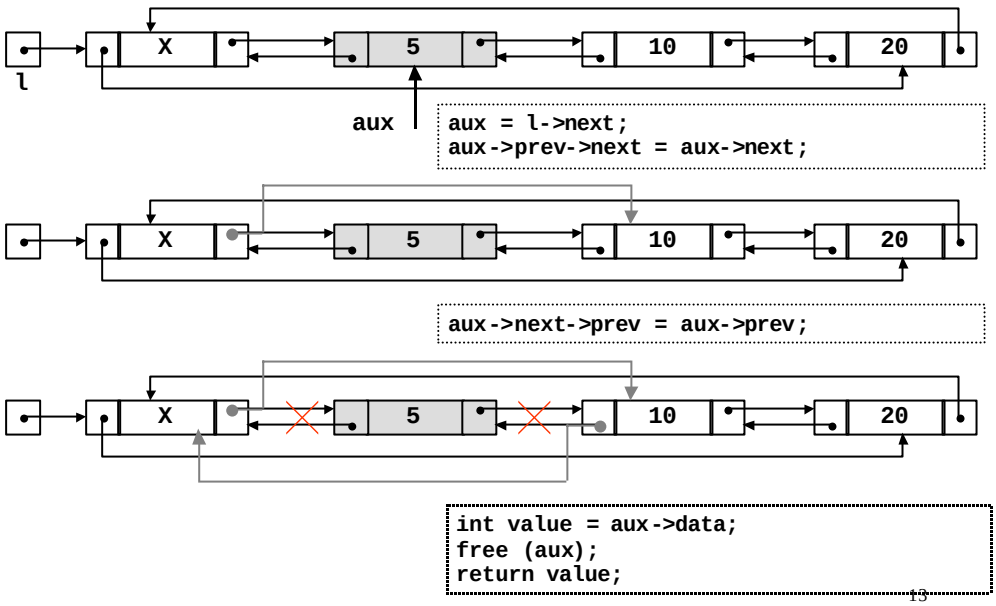

```
l->next = newelem;
```



INSERIMENTO IN TESTA

```
typedef struct t_list_element {
  int data;                                    // lista di interi
  struct t_list_element * next, * prev;
} t_list_element;

typedef t_list_element * t_list;

void insert_at_head (t_list l, int data)
{
  t_list_element * newelem;
  newelem = (t_list_element *) malloc (sizeof (t_list_element));
  if (newelem != NULL) {
    newelem->data = data;
    newelem->prev = l;
    newelem->next = l->next;
    l->next->prev = newelem;
    l->next = newelem;
  }
}
...
t_list mylist;
insert_at_head (mylist, 10);
```

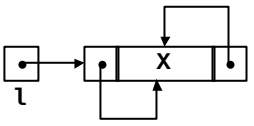
int remove_from_head (t_list l);



CANCELLAZIONE DALLA TESTA

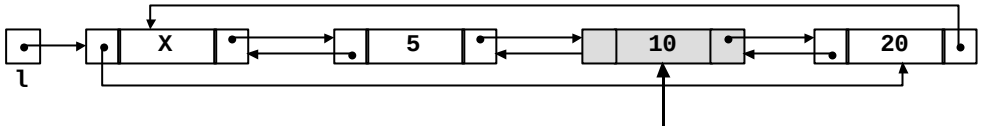
```
typedef struct t_list_element {
  int data;
  struct t_list_element * next, * prev;
} t_list_element;

typedef t_list_element * t_list;

int remove_from_head (t_list l, int data)
{
  t_list_element * aux;
  int value;
  aux = l->next;
  aux->prev->next = aux->next;
  aux->next->prev = aux->prev;
  value = aux->data;
  free (aux);
  return value;
}
```

```
int is_list_empty (t_list l)
{
  return l->next == l->prev;
}
```



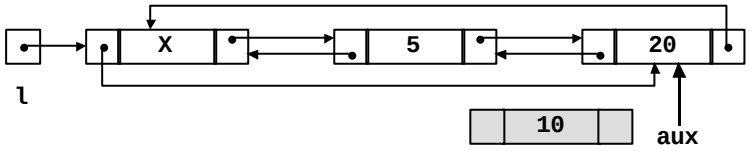

```
t_list mylist;
if (is_list_empty (mylist) == 0)
  printf ("Elemento estratto dalla lista = %d\n",
                              remove_from_head (mylist));
```

VISITA




```
typedef struct t_list_element {
  int data;
  struct t_list_element * next, * prev;
} t_list_element;

typedef t_list_element * t_list;

void dump_list (t_list l)
{
  t_list_element * aux;
  aux = l->next;
  while (aux != l) {
    printf ("Dato %d\n", aux->data);
    aux = aux->next;
  }
}
```

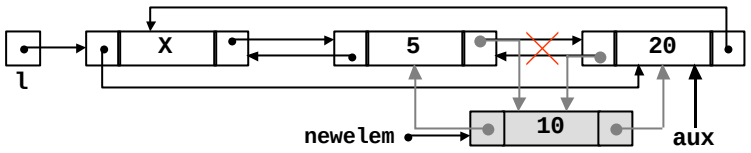

Inserimento in ordine



Ricerca del primo elemento in lista **maggiore** dell'elemento da inserire

```
typedef struct t_list_element {
  int data;
  struct t_list_element * next, * prev;
} t_list_element;

typedef t_list_element * t_list;

void insert_in_order (t_list l, int value)
{
  t_list_element * aux, * newelem;
  aux = l->next;
  while ((aux != l) && (aux->data < value)) {
    aux = aux->next;
  }
...
```

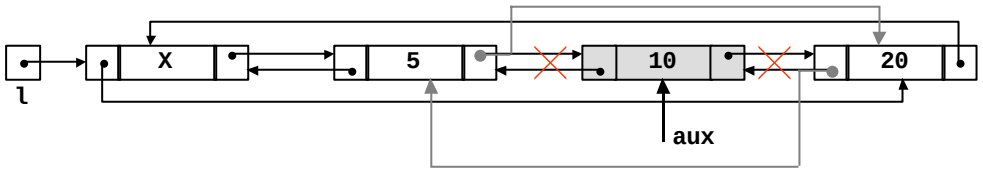
Inserimento in ordine (2)



Inserimento prima dell'elemento trovato

```
...
  newelem = (t_list_element *) malloc (sizeof (t_list_element));
  if (newelem != NULL) {
    newelem->data = value;
    newelem->prev = aux->prev;
    newelem->next = aux;
    aux->prev->next = newelem;
    aux->prev = newelem;
  }
}
```


Ricerca ed Estrazione




```
void extract (t_list l, int value)
{
  t_list_element * aux, * newelem;
  aux = l->next;
  while ((aux != l) && (aux->data != value)) {
    aux = aux->next;
  }
  if (aux != l) {
    aux->next->prev = aux->prev;
    aux->prev->next = aux->next;
    free (aux);
  }
}
```