Allocazione dinamica e vettori

Usando la “malloc” è possibile allocare un vettore le cui dimensioni sono note a runtime.

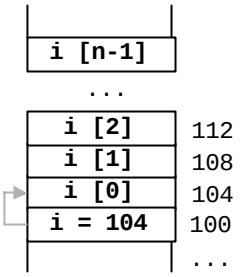
Allocazione di un vettore di interi di dimensione “n”

```
int * i;
i = (int *) malloc (n * sizeof (int) );
```

“Conversione di tipo” da “void *” *a* “int *”

Numero di elementi

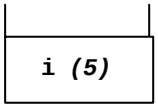
Dimensione di ogni elemento



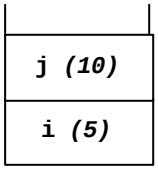
Ma cosa accade se abbiamo la necessità di cambiare, a runtime, le dimensioni del vettore allocato??

Esiste la “realloc” che permette di ri-allocare una zona di memoria modificando le dimensioni, ma …

Allocazione dinamica e vettori



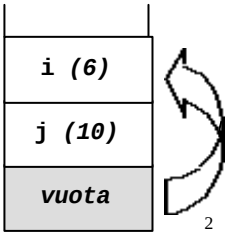

```
i = (int *) malloc (5 * sizeof (int) );
```




```
j = (int *) malloc (10 * sizeof (int) );
```

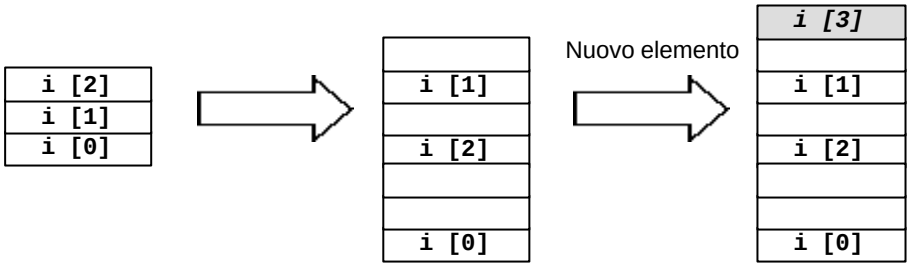
`i = (int *) realloc (i, 6 * sizeof (int) );`

Gli elementi da 0 a 4 devono essere ricopiati →
→ alto costo in termini di tempo di CPU



Soluzione: non usare l'allocazione contigua

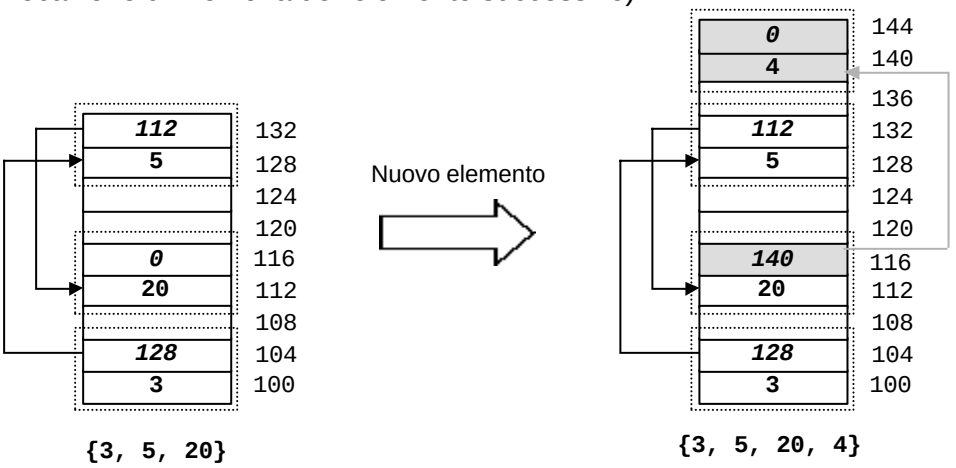
Usiamo una tecnica che non richieda che gli elementi si trovino in locazioni di memoria **contigue**



Il singolo elemento non può più essere caratterizzato solo dal dato che esso contiene ma necessita di un'informazione per "raggruppare" i vari elementi adesso sparsi in memoria (non più contigui)

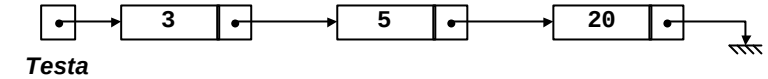
Liste semplicemente collegate

L'elemento diventa caratterizzato dall'informazione che esso contiene e da un link (collegamento) all'elemento successivo (puntatore che indica la locazione di memoria dell'elemento successivo)



Liste semplicemente collegate

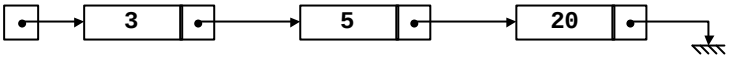
Le rappresentiamo così:

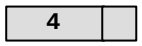


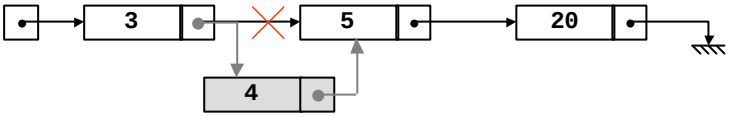
{3, 5, 20}

Testa della lista

Inserimento di un elemento:



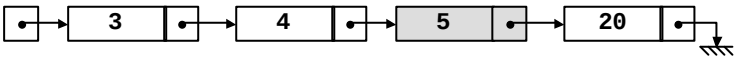




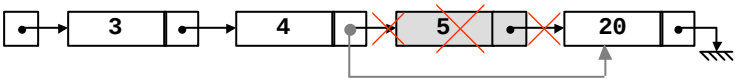
{3, 4, 5, 20}

Liste semplicemente collegate

Cancellazione di un elemento:



{3, 4, 5, 20}



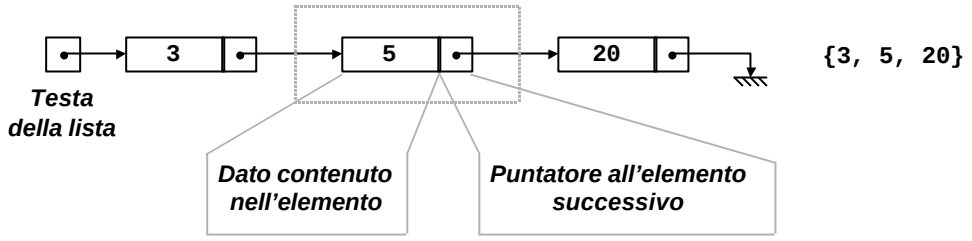
{3, 4, 20}

Accesso all'elemento **i-esimo**:

→ occorre scandire la lista a partire dalla testa

Liste semplicemente collegate: struttura

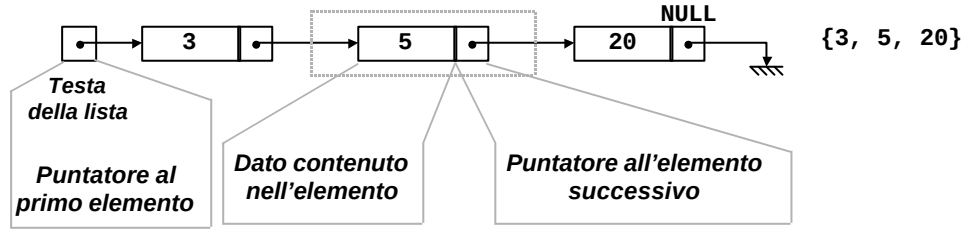
Com'è fatto il singolo elemento della lista ?




```
typedef struct {
  int   data;
  ????  * next;
} t_list_element;
```

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

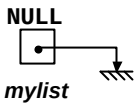

Liste semplicemente collegate: struttura



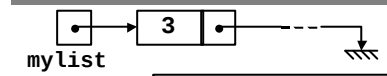

```
typedef struct t_list_element {
  int                      data;
  struct t_list_element *  next;
} t_list_element;

typedef t_list_element *   t_list;
```

Liste semplicemente collegate: implementazione

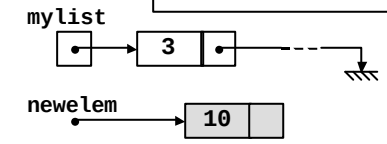
```
typedef struct t_list_element {
  int data;                          // lista di interi
  struct t_list_element * next;
} t_list_element;

typedef t_list_element * t_list;


t_list init_list (void)
{
  return NULL;
}

...

t_list mylist;
mylist = init_list ();

...
```



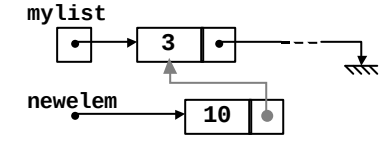
Liste semplicemente collegate: inserimento in testa



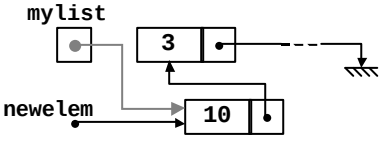

```
t_list_element * newelem;
newelem = (t_list_element *)malloc (sizeof (t_list_element));
newelem->data = data;
```



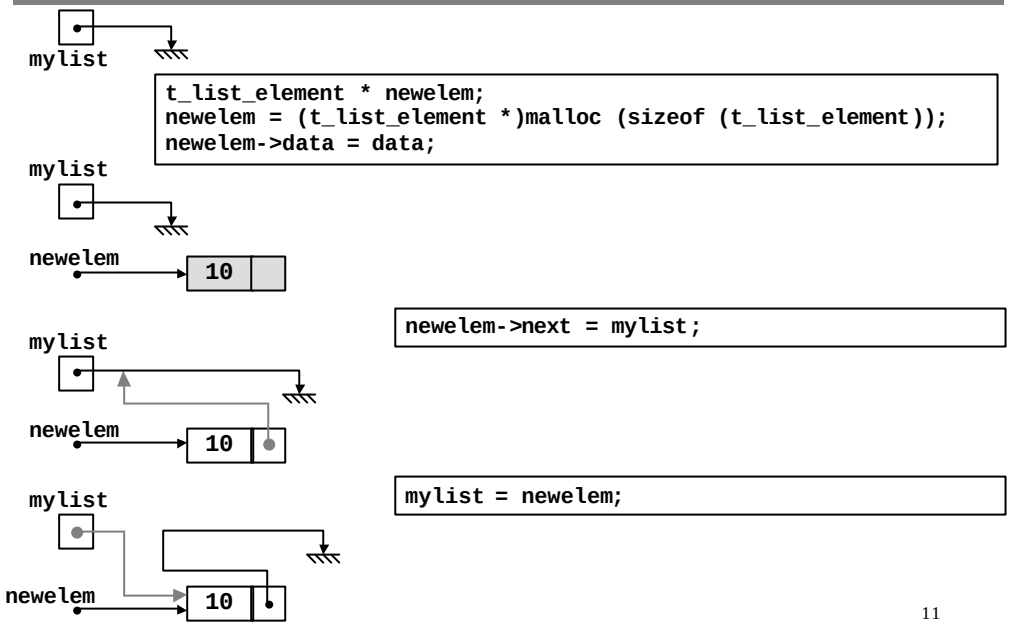

```
newelem->next = mylist;
```




```
mylist = newelem;
```



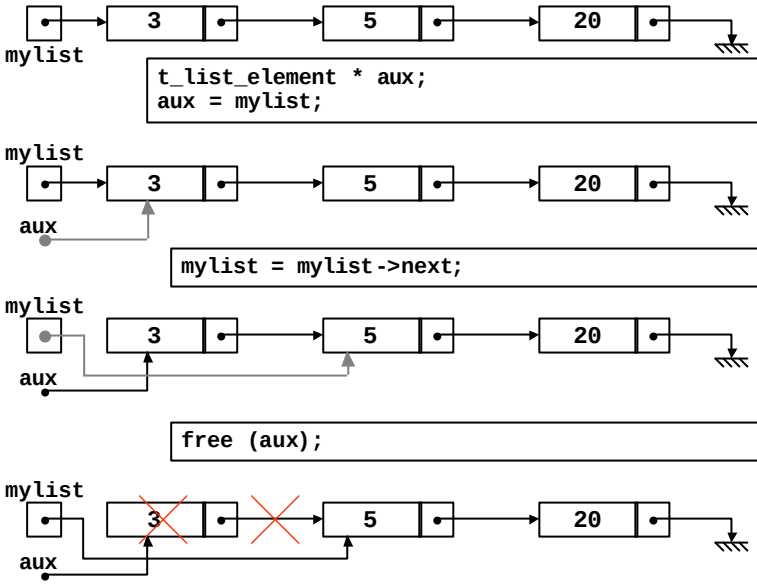
Liste semplicemente collegate: inserimento in testa (caso lista vuota)



Liste semplicemente collegate: inserimento in testa

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

typedef t_list_element * t_list;

void insert_in_list (t_list * list, int data)
{
  t_list_element * newelem;
  newelem = (t_list_element *) malloc (sizeof (t_list_element));
  if (newelem != NULL) {
    newelem->data = data;
    newelem->next = *list;
    *list = newelem;
  }
}

...
t_list mylist;
insert_in_list (&mylist, 10);
...
```

Liste semplicemente collegate: eliminazione dalla testa



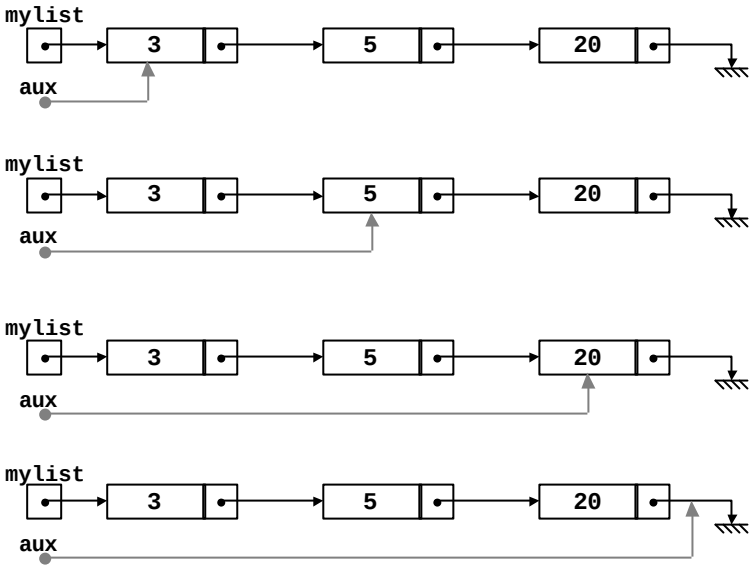
Liste semplicemente collegate: eliminazione dalla testa

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

typedef t_list_element * t_list;

int remove_from_list (t_list * list)
{
  t_list_element * aux;
  int removed_element;
  aux = *list;
  *list = (*list)->next;
  removed_element = aux->data;
  free (aux);
  return removed_element;
}
```

```
int is_list_empty (t_list list)
{
  return list == NULL;
}
```

```
t_list mylist;
if (is_list_empty (mylist) == 0)
  printf ("Elemento estratto = %d\n", remove_from_list (&mylist));
```

Liste semplicemente collegate: visita (scansione)



Liste semplicemente collegate: visita

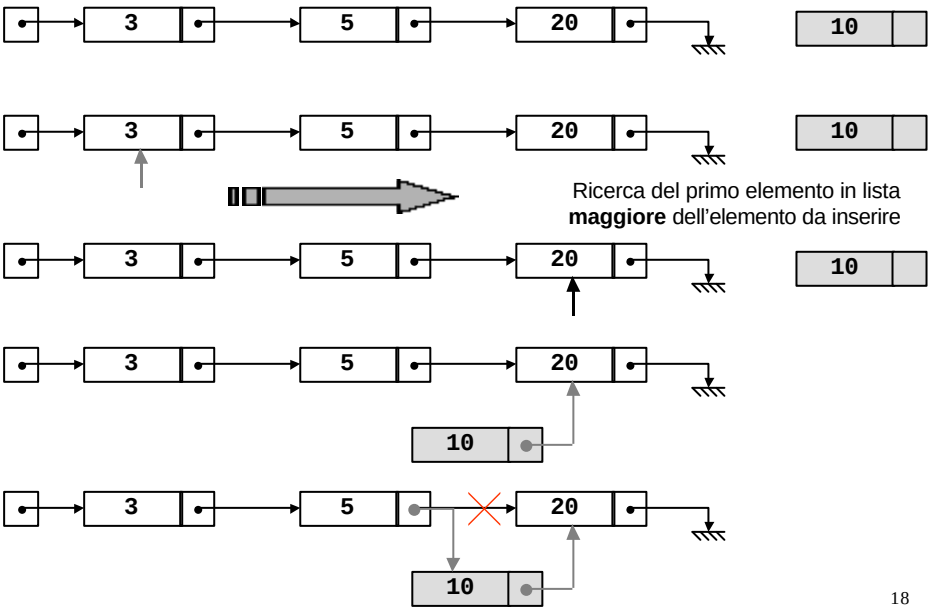
```
typedef struct t_list_element {
  int data;                          // lista di interi
  struct t_list_element * next;
} t_list_element;

typedef t_list_element * t_list;


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

...
t_list mylist;
dump_list (mylist);
...
```

Liste semplicemente collegate: lettura elemento i-esimo

```
t_list_element * get_list_element (t_list list, int elem_no)
{
  t_list_element * aux;
  int index = 0;
  aux = list;
  while (aux != NULL) {
    if (index == elem_no)
      return aux;
    aux = aux->next;
    index ++;
  }
  return NULL;
}

...
t_list mylist;
t_list_element * p;
p = get_list_element (mylist, 5);
if (p == NULL)
  printf ("L'elemento 5 non esiste\n");
else
  printf ("Il valore dell'elemento 5 e' %d\n", p->data);
...
```

17

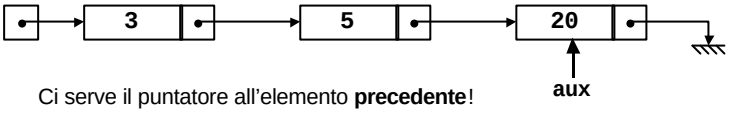
Liste semplicemente collegate **ordinate**: inserimento



18

Liste semplicemente collegate **ordinate**: inserimento

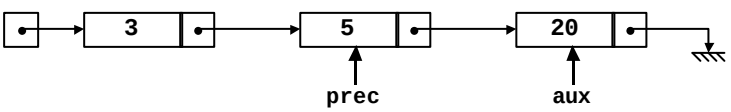
```
void insert_in_order (t_list * list, int data)
{
  t_list_element * newelem, * aux;
  newelem = (t_list_element *) malloc (sizeof (t_list_element));
  if (newelem != NULL) {
    newelem->data = data;
    aux = *list;
    while ((aux != NULL) && (aux->data < data))
      aux = aux->next;
    if (aux == NULL)
      // fine della lista: l'elemento va alla fine
    else
      // l'elemento va posto prima di aux
  }
}
```



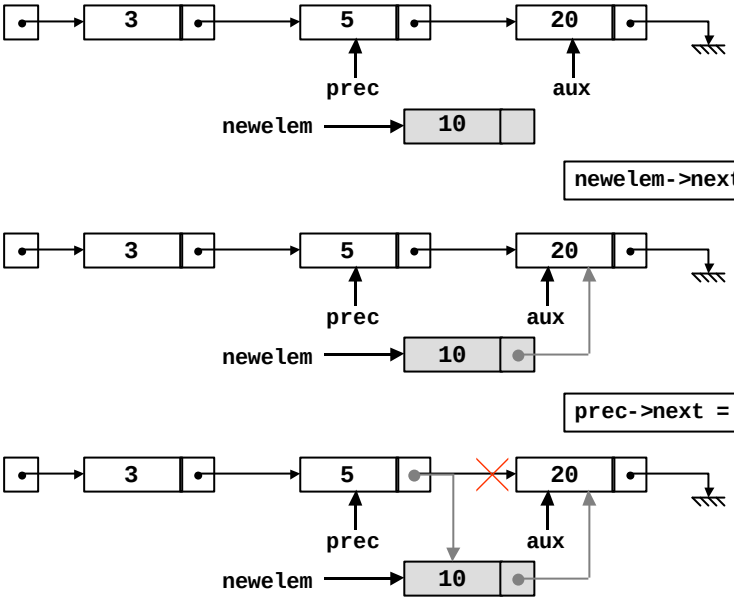
Ci serve il puntatore all'elemento **precedente**!

Liste semplicemente collegate **ordinate**: inserimento

```
void insert_in_order (t_list * list, int data)
{
  t_list_element * newelem, * aux, * prec;
  newelem = (t_list_element *) malloc (sizeof (t_list_element));
  if (newelem != NULL) {
    newelem->data = data;
    aux = *list;
    prec = NULL;
    while ((aux != NULL) && (aux->data < data)) {
      prec = aux;
      aux = aux->next;
    }
    if (aux == NULL)
      // fine della lista: l'elemento va alla fine
    else
      // l'elemento va posto prima di aux
  }
}
```



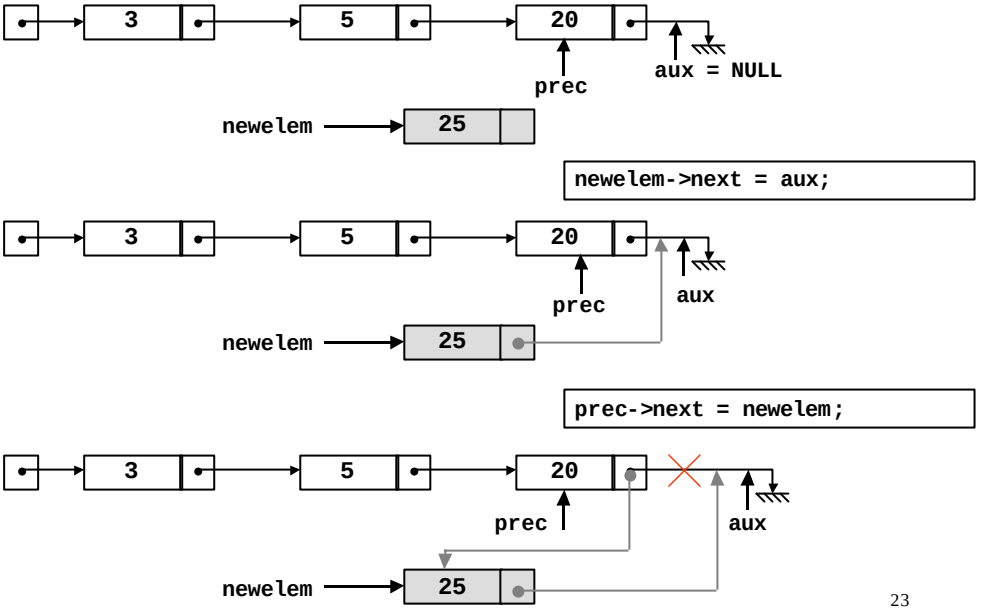
Liste semplicemente collegate **ordinate**: inserimento



Liste semplicemente collegate **ordinate**: inserimento

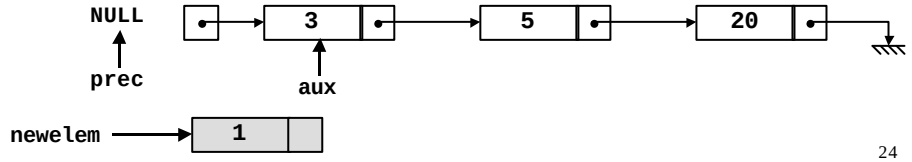
```
void insert_in_order (t_list * list, int data)
{
  t_list_element * newelem, * aux, * prec;
  newelem = (t_list_element *) malloc (sizeof (t_list_element));
  if (newelem != NULL) {
    newelem->data = data;
    aux = *list;
    prec = NULL;
    while ((aux != NULL) && (aux->data < data)) {
      prec = aux;
      aux = aux->next;
    }
    if (aux == NULL)
      // fine della lista: l'elemento va alla fine
    else {
      newelem->next = aux;
      prec->next = newelem;
    }
  }
}
```

Liste semplicemente collegate **ordinate**: inserimento

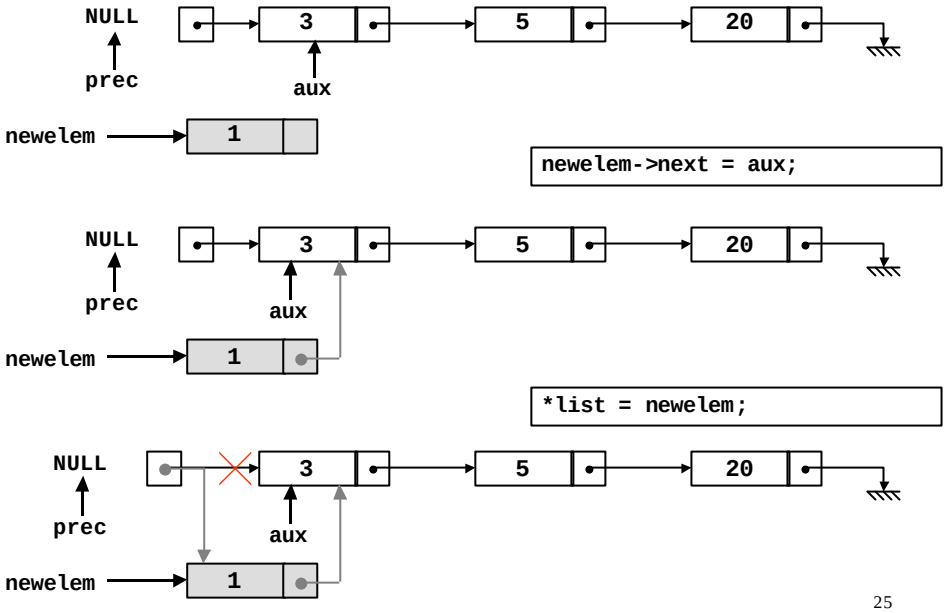


Liste semplicemente collegate **ordinate**: inserimento

```
void insert_in_order (t_list * list, int data)
{
  t_list_element * newelem, * aux, * prec;
  newelem = (t_list_element *) malloc (sizeof (t_list_element));
  if (newelem != NULL) {
    newelem->data = data;
    aux = *list;
    prec = NULL;
    while ((aux != NULL) && (aux->data < data)) {
      prec = aux;
      aux = aux->next;
    }
    newelem->next = aux;
    prec->next = newelem;
  }
}
```



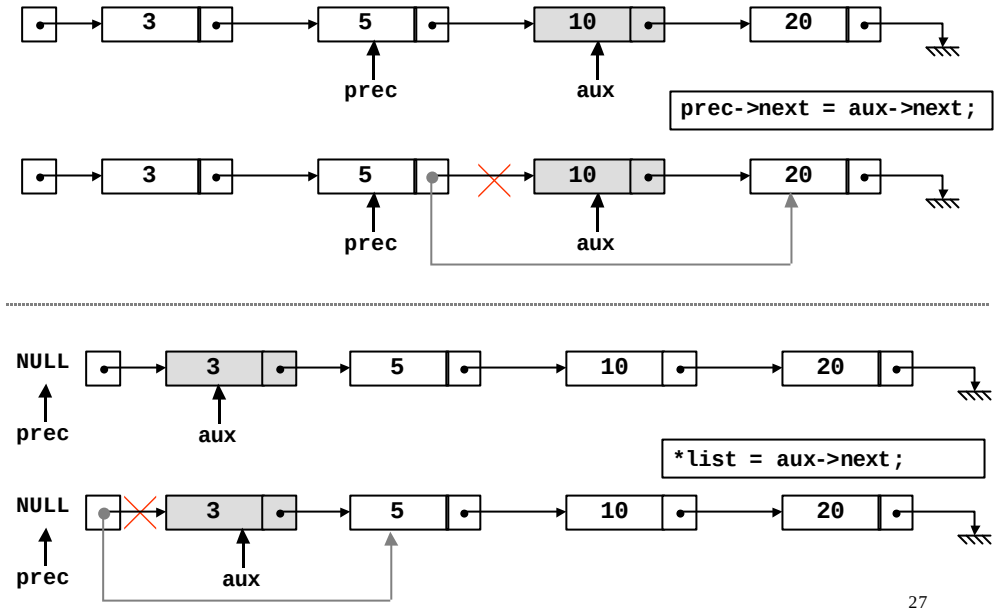
Liste semplicemente collegate **ordinate**: inserimento



Liste semplicemente collegate **ordinate**: inserimento

```
void insert_in_order (t_list * list, int data)
{
  t_list_element * newelem, * aux, * prec;
  newelem = (t_list_element *) malloc (sizeof (t_list_element));
  if (newelem != NULL) {
    newelem->data = data;
    aux = *list;
    prec = NULL;
    while ((aux != NULL) && (aux->data < data)) {
      prec = aux;
      aux = aux->next;
    }
    newelem->next = aux;
    if (prec == NULL)
      *list = newelem;
    else
      prec->next = newelem;
  }
}
```

Liste semplicemente collegate **ordinate**: cancellazione



Liste semplicemente collegate **ordinate**: cancellazione

```
void remove_element (t_list * list, int data)
{
  t_list_element * aux, * prec;
  aux = *list;
  prec = NULL;
  while ((aux != NULL) && (aux->data != data)) {
    prec = aux;
    aux = aux->next;
  }
  if (aux != NULL) {  // altrimenti l'elemento non esiste
    if (prec == NULL)
      *list = aux->next;  // l'elemento e' il primo della lista
    else
      prec->next = aux->next;
    free (aux);
  }
}
...
t_list mylist
remove_element (&mylist, 10);
...
```