51. Liste concatenate

Quando dobbiamo scandire una collezione di oggetti in modo sequenziale e non sequenziale, un modo conveniente per rappresentarli è quello di organizzare gli oggetti in un array.
Esempi:{1, 2, -3, 5, -10} è una sequenza di interi, {'a', 'd', '1', 'F'} è una sequenza di caratteri.

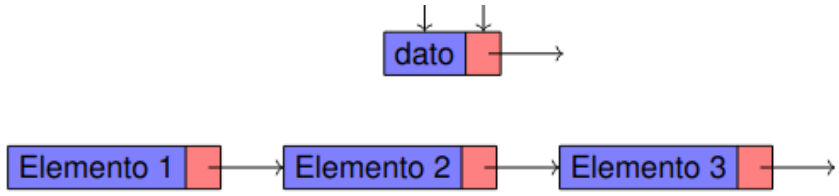
Una soluzione alternativa all'uso di array per rappresentare collezioni di oggetti quando l'accesso non sequenziale non è un requisito, consiste nell'uso delle cosiddette liste concatenate (**non** serve sapere a priori dimensione e numero dei dati da trattare).
In una lista concatenata i vari elementi che compongono la sequenza di dati sono rappresentati in zone di memoria che possono anche essere distanti fra loro (al contrario degli array, in cui gli elementi sono consecutivi).
Vantaggio: memorizzare solo informazioni che servono
Svantaggio: accesso all'elemento *i*-esimo non diretto ma sequenziale (devo attraversare tutti elementi dalla posizione 0 a quella *i*).
In una lista concatenata, ogni elemento contiene informazioni necessarie per accedere all'elemento successivo.

Una lista concatenata è un insieme di oggetti, dove ogni oggetto è inserito in un nodo contenente anche un link ad un altro nodo.

```
struct nomelista{
T valore;          // contiene i dati da memorizzare
nomelista* next;   //puntatore al nodo successivo
};
```

Esempio: lista concatenata di interi

```
struct nodo{
int dato;
nodo* next;
};
```

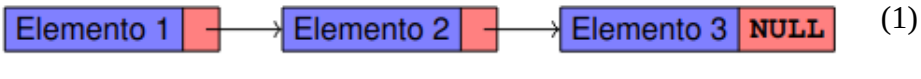


Ci possono essere anche più campi *dato*.

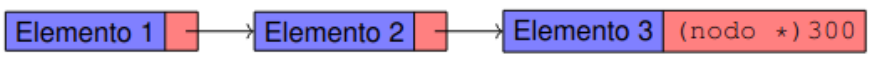
Per il nodo finale si possono adottare diverse convenzioni:

- Punta ad un link nullo che non punta a nessun nodo (*val = NULL*). (1)
- Punta ad un nodo fittizio che non contiene alcun nodo (*val = (nodo *) 300*). (2)
- Punta indietro al primo elemento della lista, creando una lista circolare. (3)

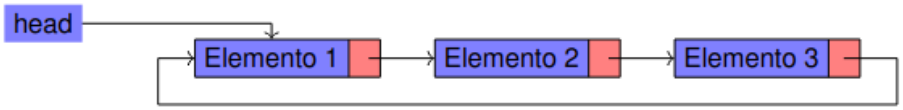
Bisogna memorizzare l'indirizzo del primo elemento di una lista concatenata in una variabile di tipo *nodo**.



(1)



(2)



(3)

51.1 Operazioni su liste concatenate

- Calcolo della lunghezza di una lista concatenata.
- Inserimento di un elemento in una lista concatenata, aumentando la lunghezza di una unità.
- Cancellazione di un elemento in una lista concatenata, diminuendo la lunghezza di una unità.
- Rovesciamento di una lista (o facendo una copia della lista originaria ribaltata oppure rovesciare la lista *in-line*: usando gli stessi nodi della lista originaria).
- Append: concatenazione di due liste.

Come scorrere una lista?

Prendo un puntatore *p* al primo elemento della nostra lista. Facendo p=*p->next* passo all'elemento successivo della lista (*p->next* è infatti il puntatore all'elemento successivo).
Se *p->next==NULL* allora sono all'ultimo elemento della mia lista.

*for(p;**p->next!=NULL**;**p=p->next**){istr;};* //un ciclo che va dal primo elemento della lista fino all'ultimo (quando usciremo *p* punta all'ultimo)

Come stampare una lista?

```
for (nodo *s =x; s != NULL; s=s->next){
cout << "valore = " << s->dato << endl;
}
```

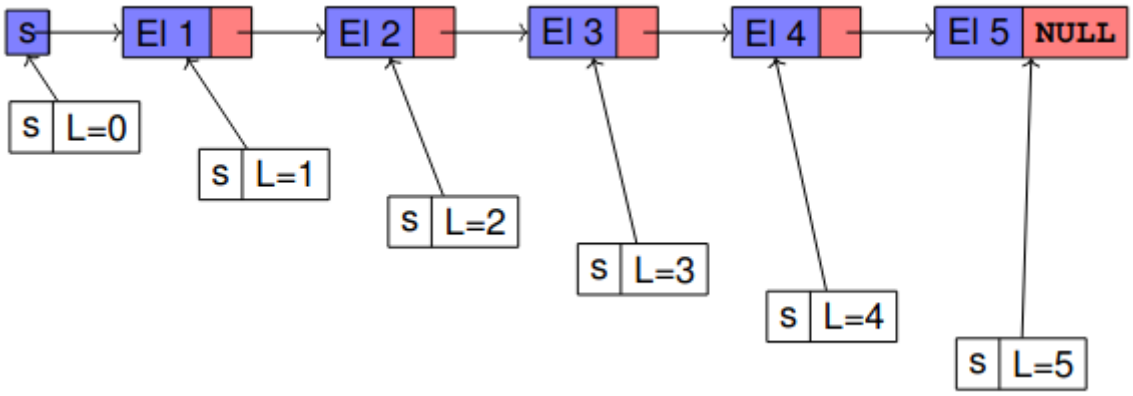
//stampa tutti i nodi (*x* è un puntatore al primo elemento)

Versione lista circolare

```
nodo* s =x;
cout << long(s) << "   "<< s->dato << " " << long(s->next) << endl;

for(s = s→next;s!=x;s=s→next)
 cout << long(s) << "   "<< s->dato << " " << long(s->next) << endl;
```

CALCOLO LUNGHEZZA DI UNA LISTA CONCATENATA

s è un puntatore al primo elemento della nostra lista.



Ilquadratino rosa invece è un puntatore all'elemento successivo della lista.
Per il calcolo della lunghezza scorro la lista confrontando il puntatore con il terminatore di lista (usando un contatore, *L*, in questo caso).

<u>Lunghezza lista</u> (ultimo nodo *NULL*)

```
int length (nodo * s) {
int l = 0;
for( ; s != NULL; s = s->next) l++;      //s punta al primo elemento della lista
                                         //entriamo quando s→next=NULL (lavoriamo anche sull'ultimo elemento)
                                         //s alla fine vale NULL
return l;
}
```

! *s* viene passato per valore, quindi quando torniamo al main punterà comunque al primo elemento della lista !

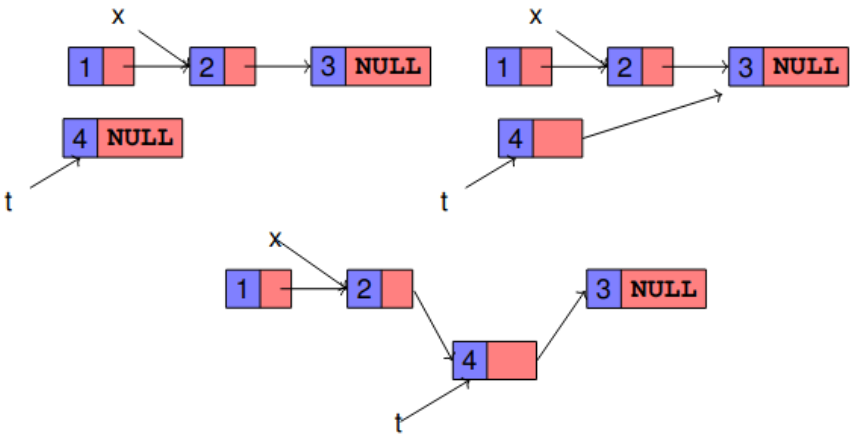
Posso fare anche così: *for(node*p=s;p!=NULL;p=p→NULL){};*

<u>Lunghezza lista circolare</u> (ultimo nodo a puntatore al primo elemento)

```
int length (nodo * s, nodo * x) {        //s e x puntano al primo elemento
int l = 0;
if (s != NULL) {                         //verifica che la lista non sia vuota
l = 1;
for( s = s->next; s != x; s = s->next) l++;   //si parte dal secondo elemento per eludere la condizione
}
return l;
}
```

<u>INSERIMENTO DI UN ELEMENTO</u>

Per inserire un nodo *t* in una lista concatenata nella posizione successiva a quella occupata da un dato nodo *x*, poniamo *t->next=x->next*, e quindi *x→next=t*. Mettiamo *t* dopo *x* e prima di *x->next* (che diventerà *t->next*, mentre *x->next* sarà proprio *t*). Se si fa il contrario perdiamo l'acceso al nodo a cui puntava *x* originariamente.
Il vantaggio rispetto agli array sta nel fatto che non dobbiamo scorrere gli altri elementi della lista.



Inserimento di un elemento dopo l'elemento *x*

```
void (node* x, node* t){
t->next=x->next;
x->next=t;
}
```

N.B.! *t* e *x* diversi da *NULL*.

RIEMPIRE UNA LISTA CONCATENATA (in ordine! *E1 E2 E3 E4*)

Primo nodo

```
nodo * x = new nodo;
cout << "Inserisci valore" << endl;
cin >> x->dato;
x->next=NULL;
```

Nodo con l'indirizzo del primo nodo (usato per inserire poi i vari elementi)

```
nodo * temp = x;
```

Inserimento degli altri elementi

```
for(int i=0;i<4;i++){
  nodo * t = new nodo;
  cout << "Inserisci valore";
  cin >> t->dato;
  t→next=NULL;
  insert(temp,t);        //non serve passarli per riferimento perché noi lavoriamo non sui puntatori ma sulla cella di memoria che puntano
  temp=t;                //serve per far si che l'elemento t del ciclo successivo venga inserito dopo l'elemento t di questo ciclo
}
```

Versione lista circolare

Analogo fino al ciclo, poi si scorre la lista fino all'ultimo elemento e lo si fa puntare al primo elemento.

```
nodo* s=x;
while(s->next!=NULL){
  s=s->next;
}
s->next=x;             //punta al primo elemento
```

Come deallocare una lista?

```
void dealloca(node * p){          //p punta al primo elemento della lista
node * l=p;

while(l!=NULL){
node * t=l;
l = l->next;
delete t;
}
}
```

Recursive version

```
void dealloc(nodo * p){

  if(p!=NULL){
    dealloc(p->next);
    delete p;
  }
}
```

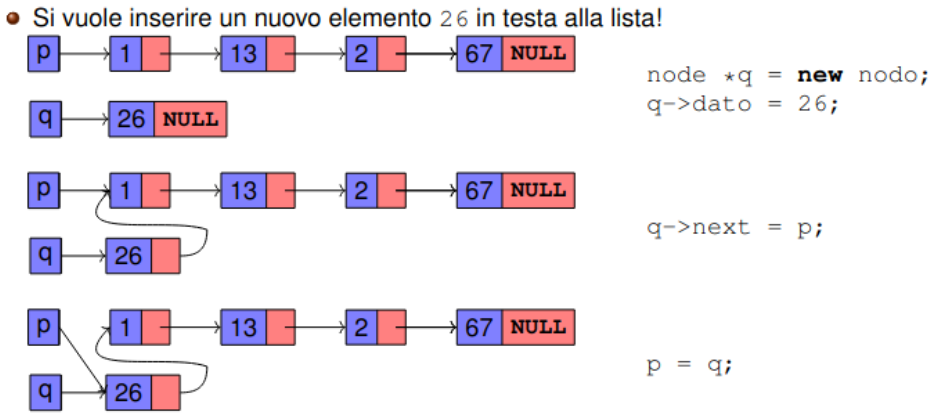
Versione lista circolare

```
void dealloc(nodo * p){        //primo elemento della lista circolare
  nodo* l = p;

  nodo* temp=l;
  l=l->next;
  delete temp;

  while(l!=p){
    temp = l;
    l=l->next;
    delete temp;
  }

}
```

Recursive version

```
void dealloc(nodo * p, nodo*x){

  if(p->next!=x){
    dealloc(p->next,x);
    delete p;
  } else {
    delete p;
  }

}
```

Inserimento di un elemento in testa



Se in *x* è memorizzato l'indirizzo di memoria del primo nodo della lista concatenata

```
node * q = new nodo;
cin >> q->dato;
q→next=x;
```

x=q; //per fare puntare *x* al primo nodo della lista che ora è *q*

2 possibili firme per una funzione che fa un inserimento in testa:

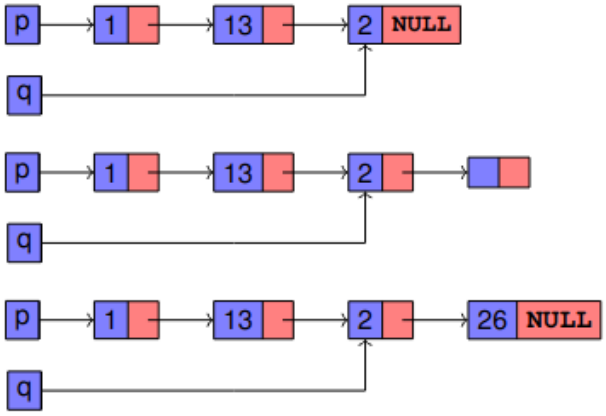
- *void insert_first(nodo * &s, int v);* : lista passata per riferimento
- *nodo * insert_first(nodo * s, int v);* : lista passata per valore ma si ritorna un puntatore

Non posso fare un passaggio per valore senza ritorno perché non modifico i puntatori e perdo l'accesso a qualche variabile. Qua modifico non modifico campi, ma puntatori.

```
void insert_first(node*&s, int v) {
node * n = new node;
n->dato = v;
n->next = s;
s = n;
}
```

```
node * insert_first(node*s, int v) {
node * n = new mode;
n->dato = v;
n->next = s;
return n;
}
```

Inserimento di un elemento in coda




```
node *q = p;
while(q->next != NULL)
  q = q->next;
```

```
q->next = new node;
```

```
q->next->dato = 26;
q->next->next = NULL;
```

Questo ragionamento è corretto solo se la lista non è vuota!

*node *q = p;*
while(q->next!=NULL)
q = q->next;

//si scorre la lista fino all'ultimo elemento poi si memorizza in *q* l'ultimo elemento

*node * r = new node;*
q->next=r;
cin >> r->dato;
r->next=NULL;

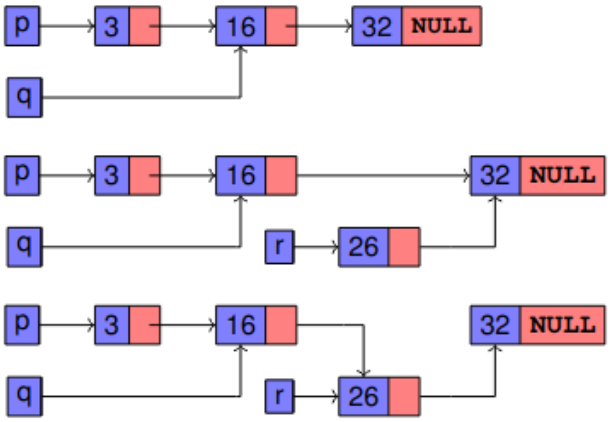
Funzione
*void insert_last(nodo * & p, int n) {* //*p* presenta il primo elemento

*nodo * r = new nodo;*
r->dato = n;
r->next = NULL;

if (p != NULL) {
*node * q = p;*
while(q->next != NULL) {
q = q->next;
}
q->next = r;
}
else {
p = r; //se la lista è vuota, *r* sarà il primo elemento
}
}

Inserimento di un elemento in una lista ordinata
Si vuole inserire un nuovo elemento in una lista ordinata mantenendo l'ordinamento.

N.B.! Vanno considerati dei casi limite




```
node *q = p;
while(q->next->dato <= 26)
  q = q->next;
```

```
node *r = new node;
r->dato = 26;
r->next = q->next;
```

```
q->next = r;
```

Parto da un puntatore al primo elemento. Scorro la lista valutando il dato nell'elemento successivo a quello in cui mi trovo. Finché è minore o uguale a ciò che voglio inserire continuo a cercare, non appena l'elemento successivo rispetto a quello in cui sto supera il valore da inserire mi fermo.
Ho il puntatore all'elemento il cui valore è minore di quello da inserire, ma l'elemento successivo è maggiore del valore.
Ora faccio l'inserimento semplice di un elemento dopo un nodo *x*.

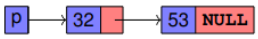
Inserimento dopo un nodo sapendo l'indirizzo di tale nodo

```
void insert(nodo* q, int n){
  node * r = new node;
  r->dato=n;
  r->next=q->next;
  q->next=r;
}
```

Non serve passare per riferimento perché non modifichiamo i puntatori.
Equivale alla funzione *insert* usata nel riempimento di una lista solo che qua si usa solo il puntatore che precederà il nodo che inseriremo.

<u>Primo caso limite</u>
Inserimento in testa, se:

- la lista è vuota (*p==NULL)*
- tutti gli elementi hanno un valore maggiore

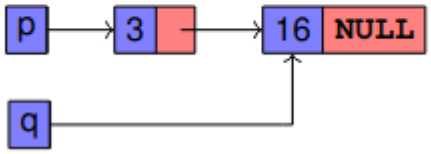



```
if ((p == NULL) || (p->dato >= 26))
  insert_first(p, 26);
```

Con questa condizione si comprendono entrambi i casi

<u>Secondo caso limite</u>
Inserimento in coda, se:

- tutti gli elementi hanno un valore minore




```
node *q = p;
while(q->next != NULL &&
      q->next->dato <= 26)
  q = q->next;
```

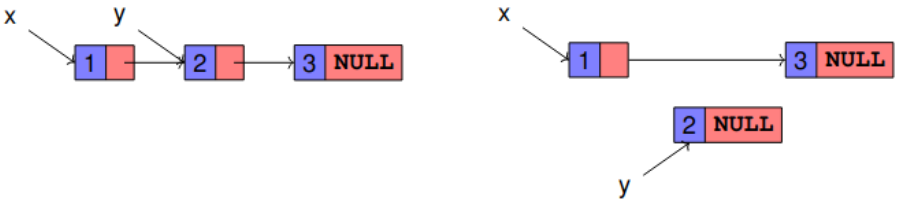
Lo stesso ciclo di prima in cui però si comprende questo caso limite. Se dobbiamo fare un inserimento in coda usciamo dal *while* con *q* che punta all'ultimo elemento.

Funzione completa (passiamo la lista per riferimento perché in *insert_first* dobbiamo lavorare direttamente sui puntatori)

```
void insert_order(nodo * &p, int inform){

if ((p==NULL) || (p->dato >= inform)) {
  insert_first(p, inform);
} else {

  nodo* q=p;
  while ((q->next != NULL) && (q->next->dato <= inform)) {
    q=q->next;
   }

nodo* r = new nodo;
r->dato = inform;
r->next = q->next;
q->next = r;
}
}
```

Rimozione di un elemento

Per rimuovere un nodo *y* generico in una lista concatenata nella posizione successiva a quella occupata da un dato nodo *x*, cambiamo *x->next* a *y->next.*



Funzione

```
node * remove_element(node *x) {
node * y = x->next;
x->next = y->next;
y->next = NULL;
return y;}          //ritorno y per deallocarlo (la deallocazione potrebbe essere anche fatta
                    immediatamente)
```

N.B.! *x*, $x \rightarrow next$ e *y* devono essere diversi da *NULL*.

Stampa con successiva deallocazione

```
while(c!=NULL) {
   temp = c;
```

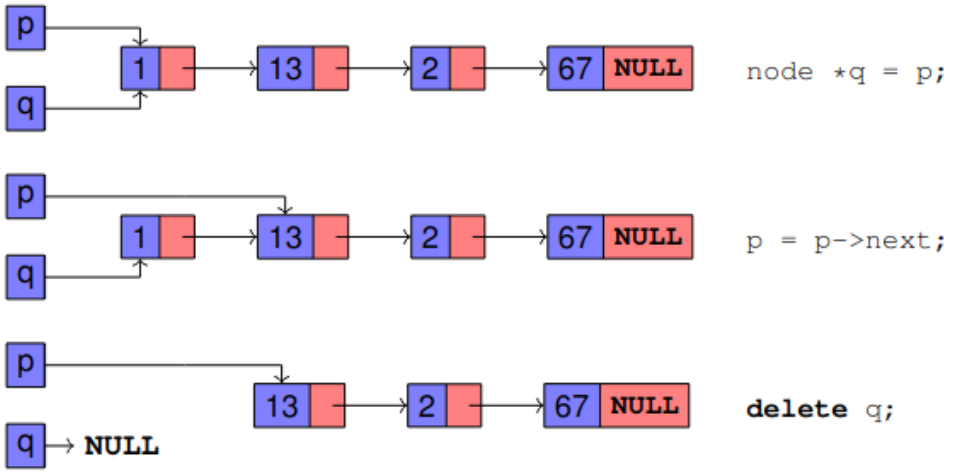
```
  cout << "valore = " << c->dato << endl;
  c=c->next;
  delete temp;
 }
```

Rimuovi, stampa e dealloca

```
for ( int i = 0; i < l; i++) {          //l lunghezza lista
nodo * t = remove_element(x);
cout << "valore = " << t->dato << endl;
delete t;
} delete x;
```

Rimozione di un elemento in testa

Si vuole rimuovere il primo elemento della lista



Creo un puntatore che punta al primo elemento. Sposto il vecchio puntatore che punta al primo facendolo puntare al secondo e dealloco la cella con il primo elemento.

2 implementazioni:

- *void remove_first(nodo * &s);*
- *nodo * remove_first(nodo * s);* deve essere invocata così ***s=remove_first(s);***

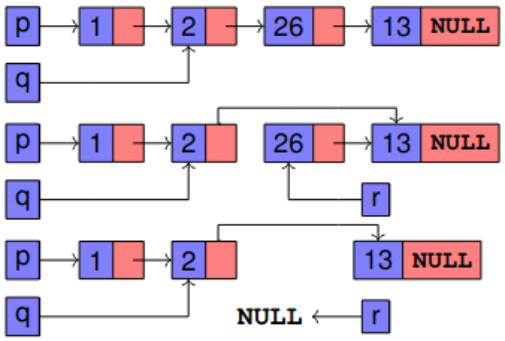
Il nodo rimosso deve essere deallocato!

```
void remove_first(nodo * & s) {          //s primo elemento
nodo * n = s;
if (s != NULL) {
s = s->next;
delete n;
}
}
```

```
nodo * remove_first(nodo * s)
{
nodo * n = s;
if (s != NULL) {
s = s->next;
delete n;
}
return s;
}
```

Rimozione di un particolare elemento

Si vuole cercare un nodo che contiene nel campo *dato* un valore ed eliminarlo!




```
nodo* q=p;


while(q->next!=NULL){
  if(q->next->dato==26){
    node *r = q->next;
    q->next = q->next->next;
    delete r;

  }
  q=q->next;
}
```

Chiaramente *q* è il nodo che precede il nodo che voglio eliminare. Prima della *delete* faccio collego elemento prima di quello eliminato all'elemento successivo.

Primo caso limite: lista vuota.

Basta mettere una guardia, che svolga il codice solo se *p!=NULL.*

```
if (p != NULL) {
        … ;
}
```

Secondo caso limite: primo nodo da levare.

Metto un'altra guardia che che verifica che il primo elemento contenga l'elemento da levaere.

All'interno di *if(p!=NULL){}* e tra l'inizializzazione di *q* e il *while* metto:

```
if (p->dato == 26) {
p = p->next; delete q;  //p deve essere propagato perché rappresenta il primo elemento
}
```

Terzo caso limite: ultimo nodo da levare.

Metto un *return;* subito dopo la *delete*.

Funzione definitiva

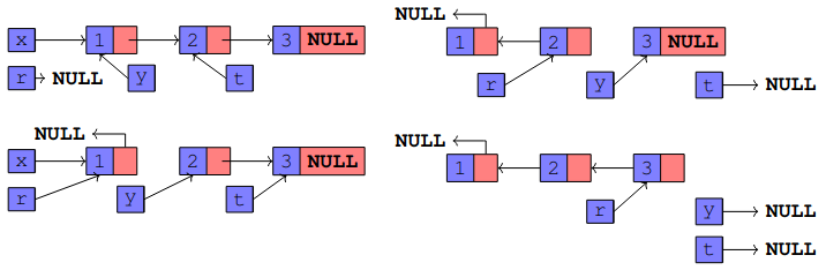
```
void search_remove(nodo* &p, int val){       //p passato per riferimento
  if (p != NULL) {
    nodo* q = p;
    if (q->dato == val) {
      p = p->next;
      delete q;
    } else {
      while(q->next != NULL) {
          if (q->next->dato == val) {
            nodo* r = q->next;
            q->next = q->next->next;
            delete r;
            return;
          }
          if (q->next != NULL) {
            q=q->next;
          }
      }
    }
  }
}
```

Rovesciamento di una lista concatenata

La funzione di rovesciamento inverte i link di una lista concatenata:

- restituisce un puntatore al nodo finale che a sua volta punta al penultimo e così via.
- Il link del primo elemento della lista è posto a NULL.
-

N.B.! Si rovescia la lista iniziale, non si crea un'altra lista.




```
node                    *
reverse(node * x) {
        node * t;
        node * y = x;
        node * r = NULL;
        while ( y != NULL ) {
                t = y->next;
                y->next = r;
                r = y;
                y = t;
        }
        return r;       //r punta al (nuovo) primo elemento della lista
}
```

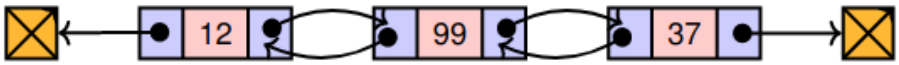
Versione che crea una copia (un'altra lista che rappresenta quella originaria rovesciata)

```
node * reverse(node* s){
node*r = NULL;

while(s!=NULL){
node*n=new node;
n→dato  = s→dato;
n→next = r;
r=n;
s=s->next;
}
return r;
}
```

52. Liste doppiamente concatenate

Sono una estensione della definizione delle liste concatenate. Differiscono per la presenza di un ulteriore puntatore al nodo che lo precede.

struct node{
int n;
node * prev;
node * next;
};



Terminatore di lista doppiamente concatenata: *NULL*.

<u>Rimozione di un nodo in una lista doppiamente concatenata</u>

Sia *q* il puntatore a un nodo.
q->next->prev=q->prev; e *q->prev=NULL;*
//faccio si che *prev* del nodo succesisivo punti al nodo precedentre
q→prev→next=q→next; e *q->next=NULL;*
//faccio si che *next* del nodo precedente punti al nodo successivo
delete q;
//dealloco il nodo che voglio eliminare

<u>Soluzione 1:</u>

```
node * remove(node * t) {
t->next->prev = t->prev;
t->prev->next = t->next;
t->next = t->prev = NULL;
return t;
}
```

<u>Soluzione 2:</u>

```
void remove(node * t) {
t->next->prev = t->prev;
t->prev->next = t->next;
delete t;
}
```

<u>Inserimento di un nodo in una lista doppiamente concatenata</u>

```
void insert_node(node * x, node * t) {      //dove x è il nodo prima di quello che inserisci
t->next = x→next;                           //mentre t è il nuovo nodo (che va dopo x)
t->next->prev = t;
t->prev = x;
x->next = t;
}
```

<u>Deallocare una lista doppiamente concatenata</u>

```
void dealloc(nodo * x){

  nodo* p = x;

  while(p!=NULL){
    nodo* temp = p;
    p=p->next;
    delete temp;
  }

}
```

Riempire una lista doppiamente concatenata

Primo nodo

```
nodo * x = new nodo;
cout << "Inserisci valore" << endl;
cin >> x->dato;
x->prev=NULL;
x->next=NULL;

nodo * temp = x;
```

Inserimento degli altri elementi

```
for(int i=0;i<4;i++){
  nodo * t = new nodo;
  cout << "Inserisci valore";
  cin >> t->dato;
  t->next=NULL;
  t->prev=temp;
  temp->next=t;
  temp=t;
}
```

Stampare una lista doppiamente concatenata

```
for(nodo * s = x;s!=NULL;s=s->next){
cout << s->dato;
}
```