Corso “Programmazione 1”
Capitolo 11: Liste Concatenate

Docente: Marco Roveri - marco.roveri@unitn.it
Esercitatori: Martina Battisti - martina.battisti-1@unitn.it
Giovanna Varni - giovanna.varni@unitn.it
Andrea E. Naimoli - andrea.naimoli@unitn.it

C.D.L.: Informatica (INF)
A.A.: 2024-2025
Luogo: DISI, Università di Trento
URL: https://shorturl.at/VRc6b

Ultimo aggiornamento: 6 novembre 2024

Terms of Use and Copyright

The material (text, figures) in these slides is authored by Marco Roveri.

Introduzione

- 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.
- 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).
- In una lista concatenata, ogni elemento contiene informazioni necessarie per accedere all'elemento successivo.

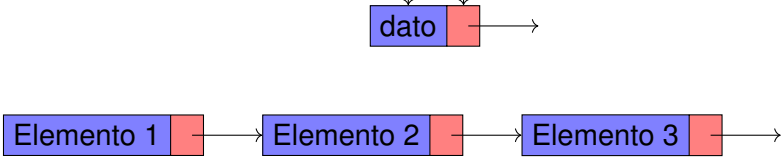
Liste concatenate

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

```
// Lista concatenata di interi
struct nodo {
  int dato;
  nodo * next;
}
```

Campo contenente informazioni del nodo

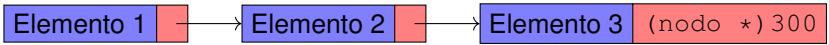
Puntatore al nodo successivo



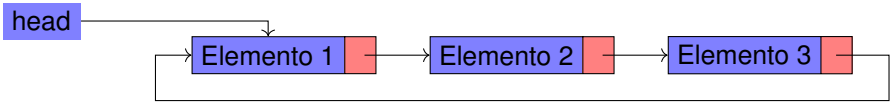
Liste concatenate (II)

- Per il nodo finale si possono adottare diverse convenzioni:
  - Punta ad un link nullo che non punta a nessun nodo (e.g. `<val> = NULL`)

  - Punta ad un nodo fittizio che non contiene alcun nodo (e.g. `<val> = (nodo *)300`)



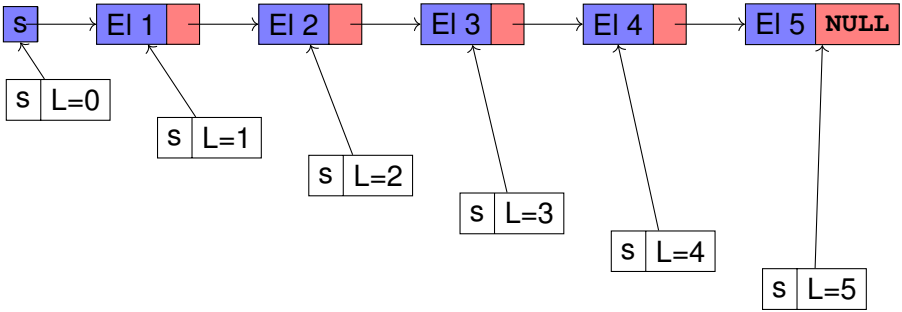
  - Punta indietro al primo elemento della lista, creando una lista circolare



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.
- Append: concatenazione di due liste.

Calcolo lunghezza di una lista



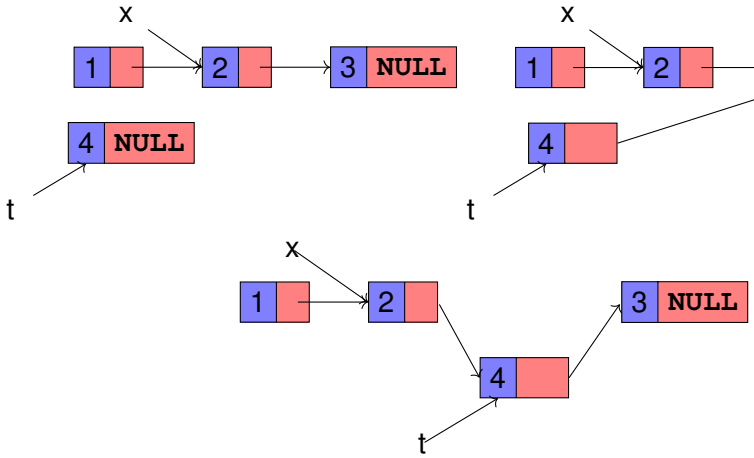
Calcolo lunghezza di una lista (II)

```
// lista con terminatore NULL
int length (nodo * s) {
  int l = 0;
  for( ; s != NULL; s = s->next) l++;
  return l;
}
```

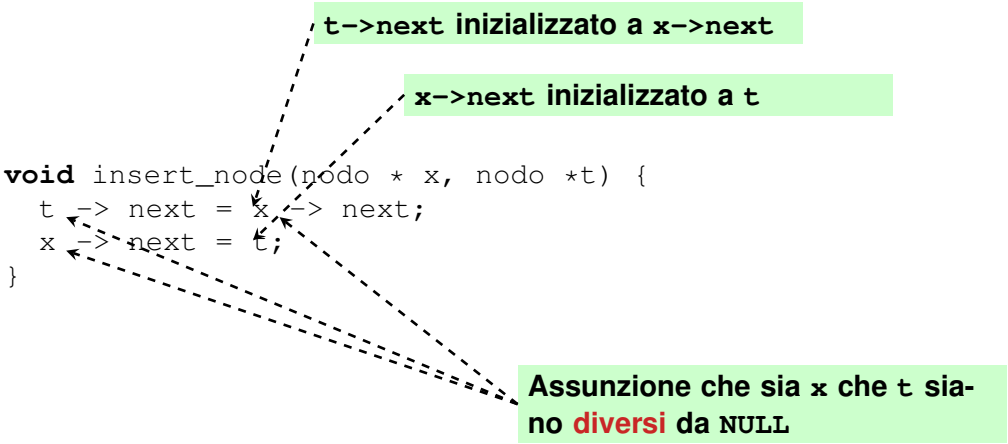
```
// lista circolare, x primo elemento
int length (nodo * s, nodo * x) {
  int l = 0;
  if (s != NULL) {
    l = 1;
    for( s = s->next; s != x; s = s->next) l++;
  }
  return l;
}
```

Inserimento di un elemento

- Per inserire un nodo `t` in una lista concatenata nella posizione successiva a quella occupata da un dato nodo `x`, poniamo `t->next` a `x->next`, e quindi `x->next` a `t`.



Inserimento di un elemento (II)



Inserimento di un elemento (II)

```
int main () {
  nodo * x = new nodo;
  cout << "Inserire numero: ";
  cin >> x->dato;
  x->next = NULL;
  for (int i = 0; i < 10; i++) {
    nodo * t = new nodo;
    cout << "Inserire un numero: ";
    cin >> t->dato;
    t->next = NULL;
    insert_node(x, t);
  }
  for (nodo *s =x; s != NULL; s=s->next)
      cout << "valore = " << s->dato << endl;
}
```

Allocazione di un `nodo` per memorizzare primo elemento

Allocazione di un nuovo `nodo` per memorizzare i-esimo elemento

Campo `next` di `t` inizializzato a `NULL`

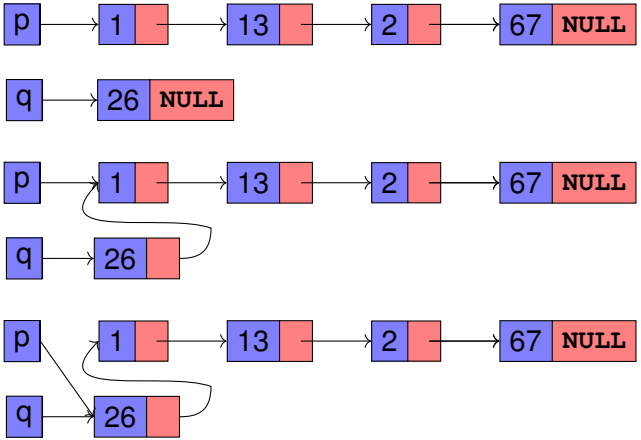
Inserzione del nuovo elemento `t` successivamente al nodo iniziale `x`

Variabile temporanea per scorrere lista

Manca deallocazione della lista!!!

Inserimento di un elemento in testa

- Si vuole inserire un nuovo elemento `26` in testa alla lista!




```
node *q = new nodo;
q->dato = 26;
```

```
q->next = p;
```

```
p = q;
```

Inserimento di un elemento in testa (II)

- Se dobbiamo inserire un elemento in testa della lista:
  - `void insert_first(nodo * s, int v);`
  - `void insert_first(nodo * &s, int v);`
  - `nodo * insert_first(nodo * s, int v);`
- La seconda e la terza sono le uniche possibili dichiarazioni corrette.
  - Nella seconda side effect sull'argomento.
  - Nella terza lista nuova ritornata dalla funzione.

Inserimento di un elemento in testa (III)

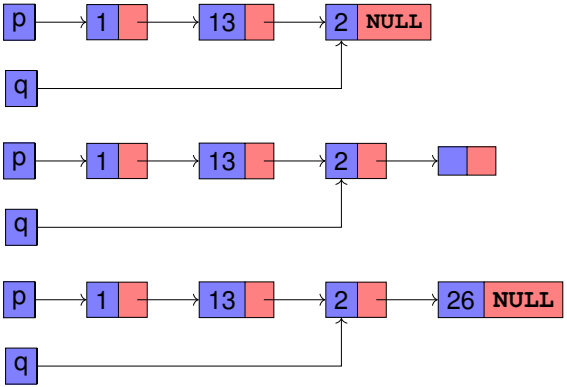
```
void insert_first(node * s, int v) {
  node * n = new node;
  n->dato = v;
  n->next = s;
  s = n;
}
```

```
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

- Si vuole inserire un nuovo elemento `26` in coda alla lista!




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

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

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

Nota:

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

Inserimento di un elemento in coda (II)

```
void insert_last(nodo * & p, int n) {
  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;
  }
}
```

Allocazione del nuovo `nodo`

Se la lista non è vuota, cerco in `q` il puntatore all'ultimo elemento

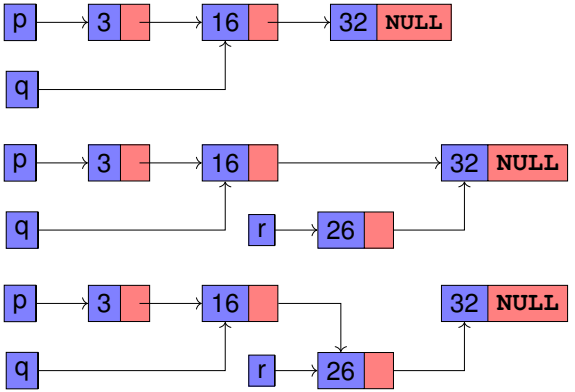
`q` è garantito essere diverso da `NULL`

Memorizzo in `q->next` il nuovo nodo `r` allocato in precedenza

Se la lista è vuota, `p` punta al nuovo nodo allocato: `p` è passato per riferimento

Inserimento di un elemento in lista ordinata

- Si vuole inserire un nuovo elemento `26` in una lista ordinata mantenendo ordinamento!




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

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

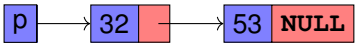
```
q->next = r;
```

Nota:

Devono essere considerati alcuni casi limite!!!

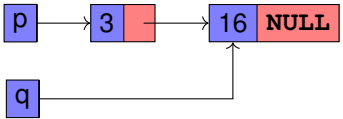
Inserimento di un elemento in lista ordinata (II)

- Primo caso limite: inserimento in testa
  - perchè la lista è vuota: `p == NULL`
  - perchè tutti gli elementi hanno un valore maggiore




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

- Secondo caso limite: inserimento in coda
  - perchè tutti gli elementi hanno un valore minore



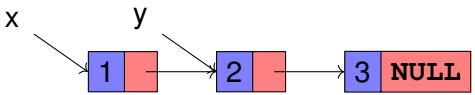

```
node *q = p;
while(q->next != NULL &&
      q->next->dato <= 26)
  q = q->next;
```

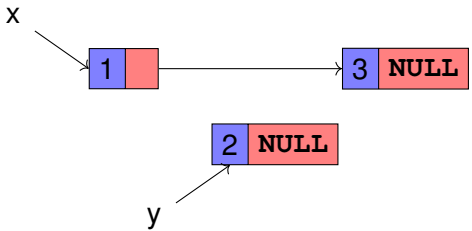
Inserimento di un elemento in lista ordinata (III)

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





Rimozione di un elemento (II)

```
node * remove_element(node **x) {
  node * y = x->next;
  x->next = y->next;
  y->next = NULL;
  return y;
}
```

y inizializzato a x->next

x->next punta a y->next

Assunzione che x, x->next (e quindi anche y) siano diversi da NULL

y ritornato per ad esempio essere deallocato!

Rimozione di un elemento (II)

```
int main () {
  nodo * x = new nodo;
  cout << "Inserire numero: ";
  cin >> x->dato
  x->next = NULL;
  for (int i = 0; i < 10; i++) {
    nodo * t = new nodo;
    cout << "Inserire un numero: ";
    cin >> t->dato
    t->next = NULL;
    insert_node(x, t);
  }
  for ( int i = 0; i < 10; i++) {
    nodo * t = remove_element(x);
    cout << "valore = " << t->dato << endl;
    delete t;
  }
  delete x;
}
```

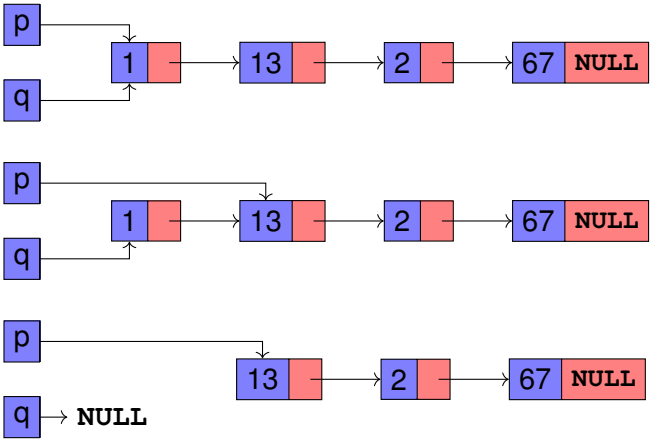
Variabile temporanea per memorizzare nodo rimosso!

Deallocazione del nodo rimosso

Deallocazione del nodo `x` iniziale

Rimozione di un elemento in testa

- Si vuole eliminare primo elemento della lista!



Rimozione di un elemento in testa (II)

- Se dobbiamo rimuovere un elemento in testa della lista:
  - `void remove_first(nodo * s);`
  - `void remove_first(nodo * &s);`
  - `nodo * remove_first(nodo * s);`
- La seconda e la terza sono le uniche possibili dichiarazioni corrette.
  - Nella seconda side effect sull'argomento.
  - Nella terza lista nuova ritornata dalla funzione.

Nota:

Il nodo rimosso deve essere deallocato!!!

Rimozione di un elemento in testa (III)

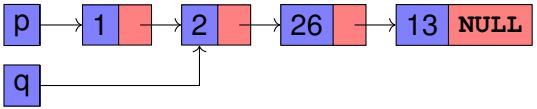
```
void remove_first(nodo * s) {
  nodo * n = s;
  if (s != NULL) {
    s = s->next;
    delete n;
  }
}
```

```
void remove_first(nodo * & s) {
  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;
} // attenzione a come invocata
```

Rimozione di un elemento particolare

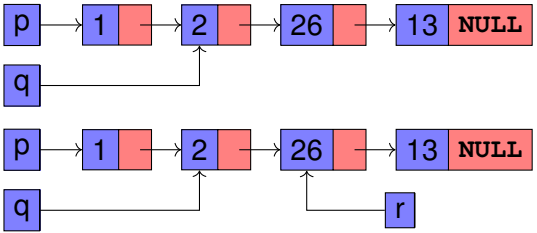
- Si vuole cercare un nodo che contiene nel campo `dato` un valore ed eliminarlo!




```
nodo* q=p;


while(q->next!=NULL){
  if(q->next->dato==26){



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

Rimozione di un elemento particolare

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

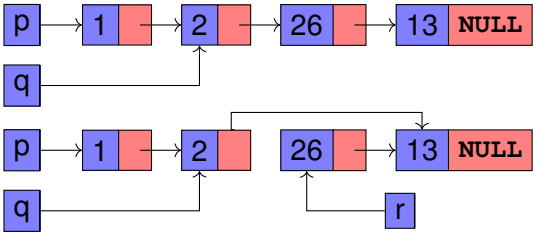



```
nodo* q=p;
```

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


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

Rimozione di un elemento particolare

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

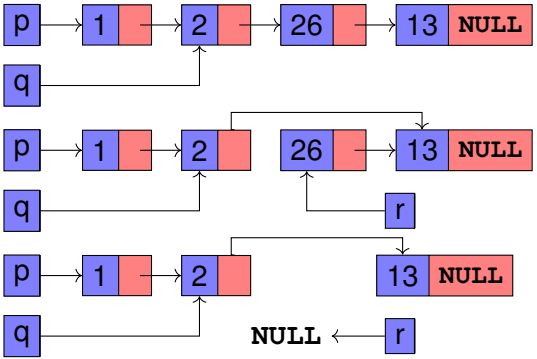



```
nodo* q=p;
```

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


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

Rimozione di un elemento particolare

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

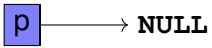



```
nodo* q=p;
```

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

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

Rimozione di un elemento particolare

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



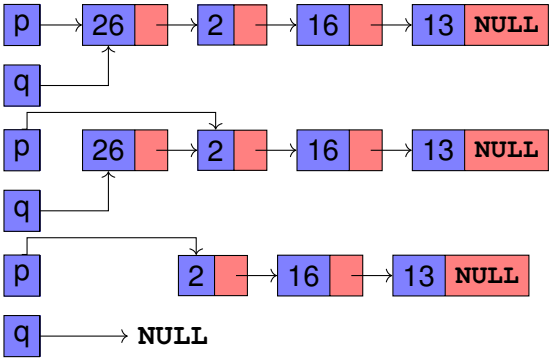

```
if (p != NULL) {
  nodo* q=p;


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

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

Nota:

Primo caso limite: Lista vuota!!!

Rimozione di un elemento particolare

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



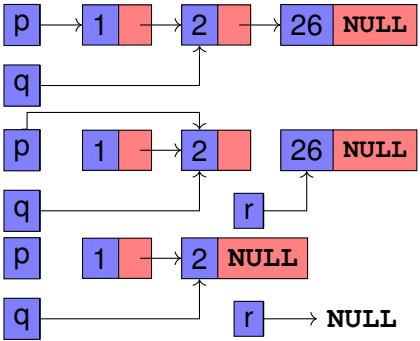

```
if (p != NULL) {
  nodo* q=p;
  if (p->dato == 26) {
    p = p->next; delete q;
  }
  while(q->next!=NULL){
    if(q->next->dato==26){
      node *r = q->next;
      q->next = q->next->next;
      delete r;

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

Nota:

Secondo caso limite: primo nodo da levare!!!

Rimozione di un elemento particolare

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



Nota:

Terzo caso limite: ultimo nodo da levare!!!

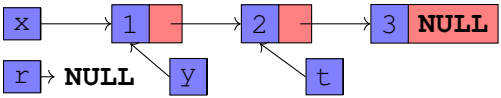
```
if (p != NULL) {
  nodo* q=p;
  if (p->dato == 26) {
    p = p->next; delete q;
  }
  while(q->next!=NULL){
    if(q->next->dato==26){
      node *r = q->next;
      q->next = q->next->next;
      delete r;
      return;
    }
    q=q->next;
  }
}
```

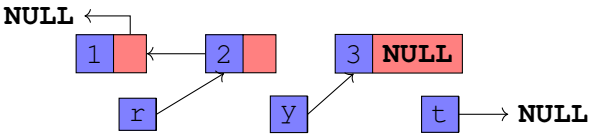
Rimozione di un elemento particolare (II)

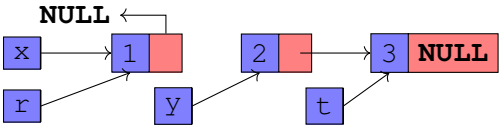
```
void search_remove(nodo* &p, int val){
  if (p != NULL) {
    nodo* q = p;
    if (q->dato == val) {
      p = p->next;
      delete q;
    }
    else {
      while(q->next != NULL) {
        if (q->next->dato == val) {
          nodo* r = q->next;
          q->next = q->next->next;
          delete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

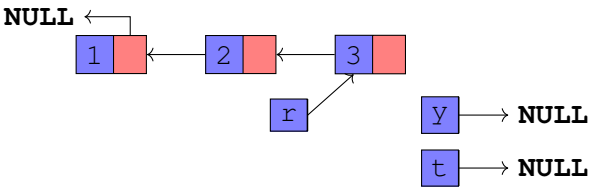
Rovesciamento di una lista concatenata

- 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**.





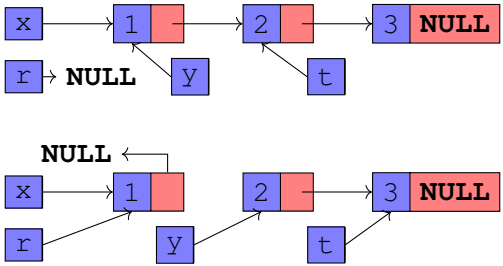




Rovesciamento di una lista concatenata (II)

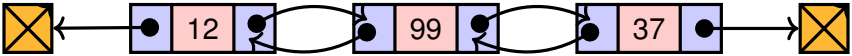
- La funzione di rovesciamento inverte i link di una lista concatenata:
  - restituisce un puntatore al nodo finale che a sua volta punta al penultimo e così via.
  - Il link del primo elemento della lista è posto a **NULL**.

```
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;
}
```



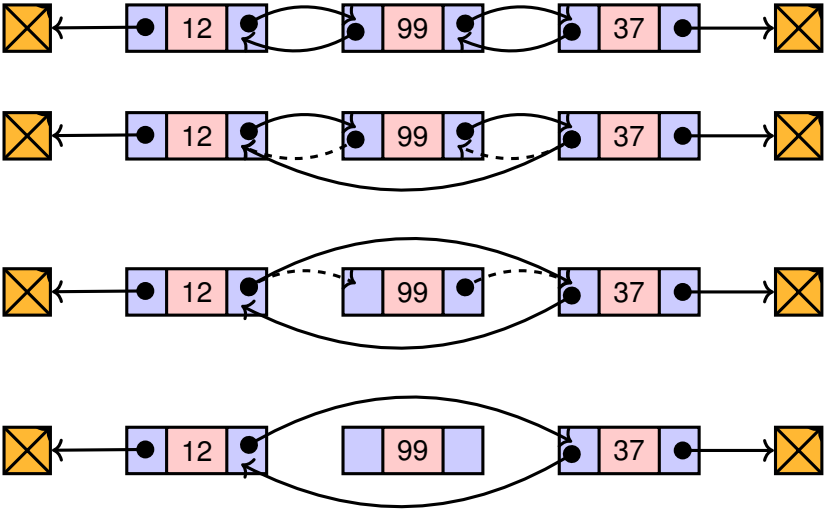
Liste doppiamente concatenate

- 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;
};
```

Rimozione di un nodo in una lista doppiamente concatenata



Inserimento di un nodo in una lista doppiamente concatenata

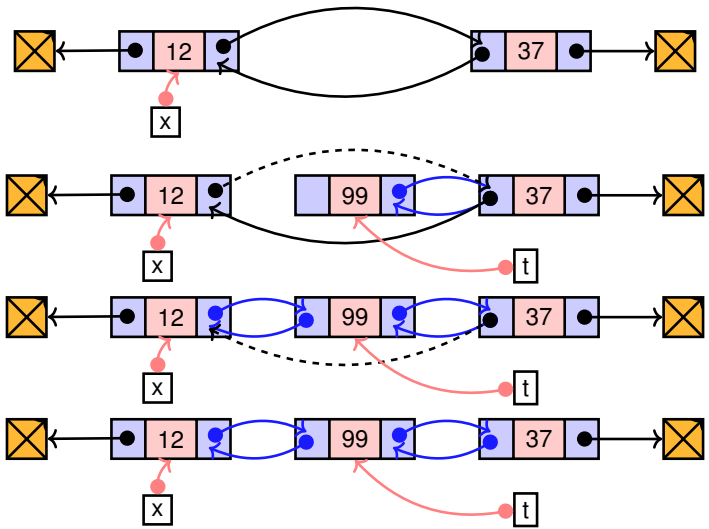
- Soluzione 1:

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

- Soluzione 2:

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

  delete t;
}
```

Inserimento di un nodo in una lista doppiamente concatenata



Inserimento di un nodo in una lista doppiamente concatenata

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

Esercizi proposti

Esercizi proposti!:
{ LISTE_CONCATENATE/ESERCIZI_PROPOSTI.txt }