Inf Gen II

INFORMATICA GENERALE II

Ingegneria delle Telecomunicazioni
Università di Trento

Marco Roveri

roveri@irst.itc.it

Liste Concatenate

AA 2005/2006

MR

Inf Gen II

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 le liste 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.

AA 2005/2006

MR

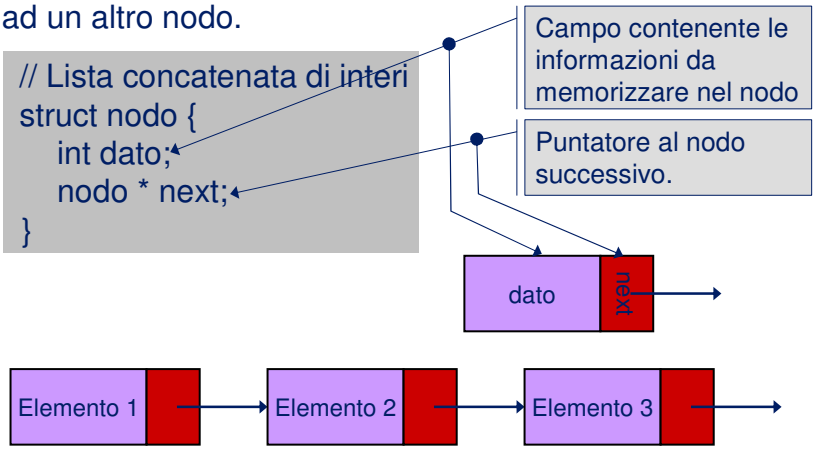
Liste concatenate

- Vantaggi rispetto all'array:
  - Flessibilità di modifica.
  - Memorizzo solo quello che mi serve. Mentre con array potrei sprecare memoria (array sovradimensionato).
- Svantaggi rispetto all'array:
  - Lo svantaggio principale consiste nell'onerosità nell'accesso ai suoi elementi.
    - Unico modo per raggiungere un dato elemento consiste nello scorrere la lista dal nodo iniziale.

Liste concatenate

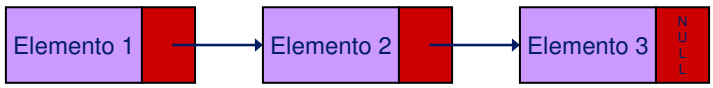
- Una *lista concatenata* è un insieme di oggetti, dove ogni oggetto è inserito in un *nodo* contenente anche un *link* ad un altro nodo.

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



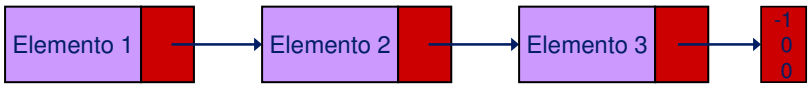
Liste concatenate

- Per il nodo finale adotteremo le seguenti convenzioni:
  - È un *link nullo* che non punta ad alcun nodo (e.g. **NULL**).



Liste concatenate

- Per il nodo finale adotteremo le seguenti convenzioni:
  - Punta ad un *nodo fittizio* che non contiene alcun oggetto (e.g. (nodo *)-100))



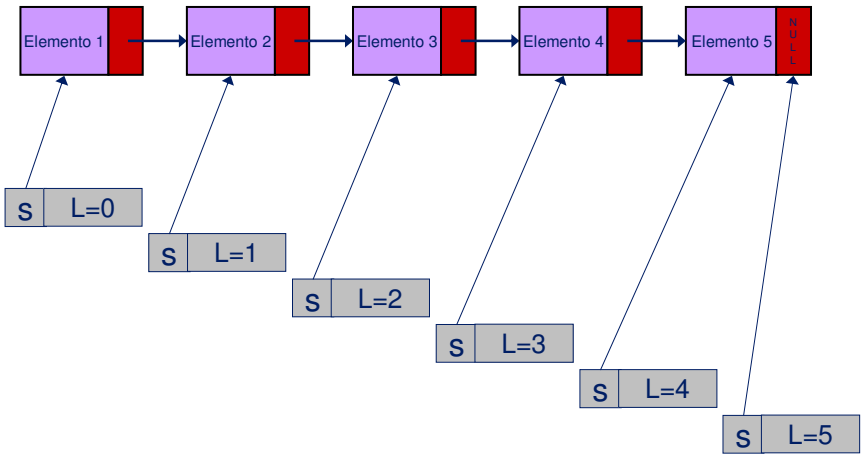
Liste concatenate

- Per il nodo finale adotteremo le seguenti convenzioni:
  - Punta indietro al primo elemento della lista, creando quindi 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



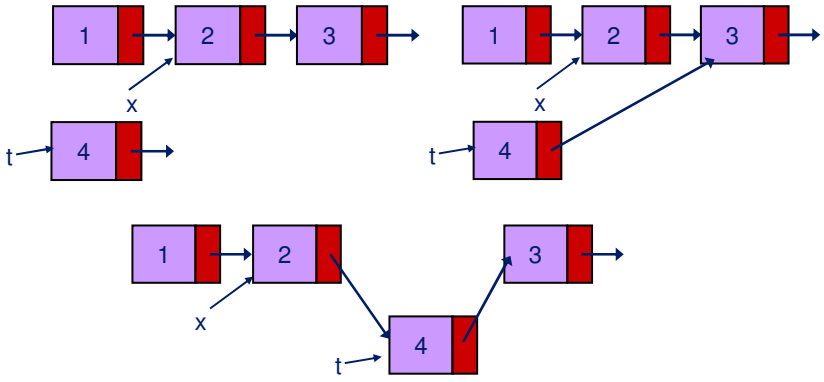
Calcolo lunghezza di una lista

```
// 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

t->next inizializzato a x->next

x->next inizializzato a t

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

Assunzione che sia x e t sono **diversi** da NULL

Inserimento di un elemento

```
int main () {
   sequenza * x = new nodo;
   cout << "Inserire numero: ";
   cin >> x->dato
   x->next = NULL;
   for (int i = 0; i < 10; i++) {
      sequenza * 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

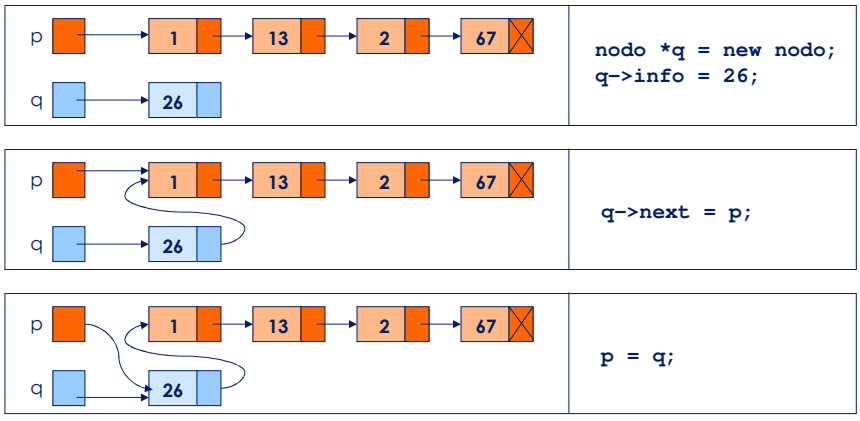
Inserzione del nuovo elemento t successivamente al nodo iniziale x.

Variabile temporanea per scorrere la lista

Manca deallocazione della lista!!!!

Inserimento di un elemento in testa

- Vogliamo inserire un nuovo elemento in testa alla lista, contenente per esempio il numero 26.



Inserimento di un elemento in testa

- Se dobbiamo inserire un elemento in testa della lista:
  - void insert_first(sequenza * s, int v);
  - void insert_first(sequenza * &s, int v);
  - sequenza * insert_first(sequenza * 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

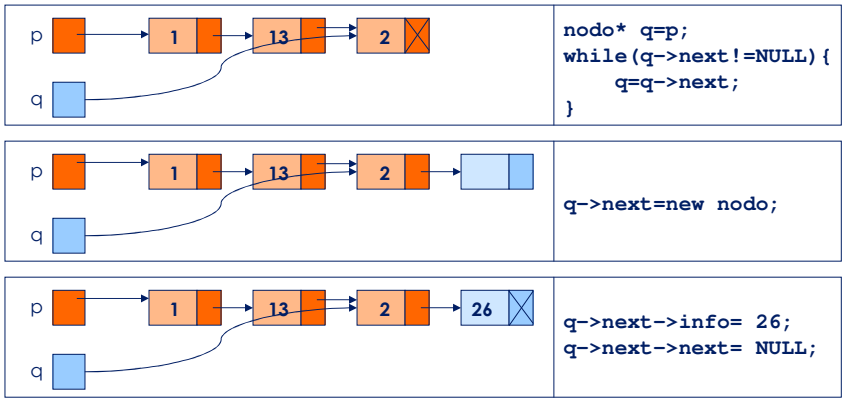
```
void insert_first(node * s, int v)  {
 node * n = new node;
 n->dato = v;
 n->next = s;
 s = n;
}
```

```
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

- Vogliamo inserire un nuovo elemento in coda alla lista, contenente per esempio il numero 26.



Attenzione: Va bene solo se la lista non è vuota!

Inserimento di un elemento in coda

```
void insert_last(nodo * & p, int n) {
   nodo * r = new nodo;
   r->info = 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

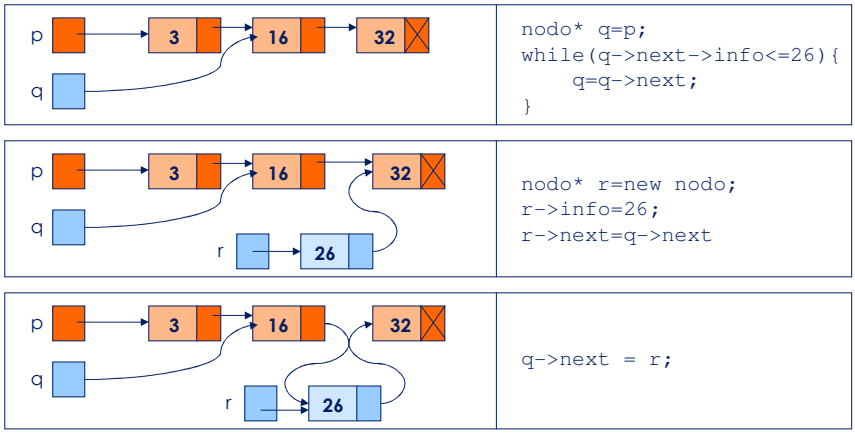
q qui è garantito essere diverso da NULL

Memorizzo in q->next il nuovo nodo r allocato precedentemente.

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

Inserimento di un elemento ordinato

- Vogliamo inserire un nuovo elemento (contenente per esempio il numero *26*) in una **lista ordinata** (ordine *crescente*).

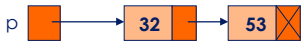


Attenzione: dobbiamo considerare i casi limite!

Inserimento di un elemento ordinato

- **Primo caso limite:** dobbiamo inserire l'elemento in testa
  - Perche' la lista e' vuota

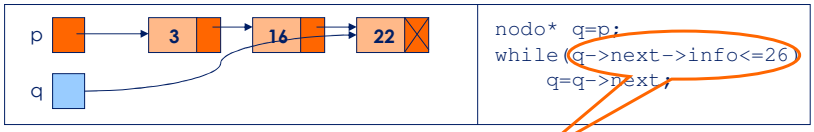
  - oppure perche' tutti gli altri elementi hanno un valore maggiore




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

Inserimento di un elemento ordinato

- **Secondo caso limite:** dobbiamo inserire l'elemento in coda
  - Perchè tutti gli altri elementi hanno un valore minore.



OCCORRE CONTROLLARE SE È L'ULTIMO ELEMENTO!

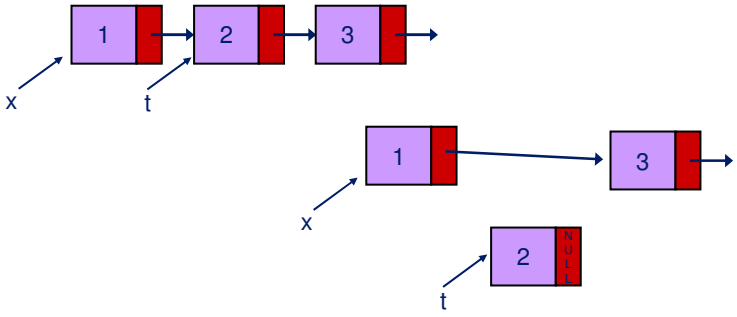
```
nodo* q=p;
while((q->next!=NULL)&&(q->next->info<=26))
   q=q->next;
```


Inserimento di un elemento ordinato

```
void insert_order(nodo * &p, int inform){
  if ((p==NULL) || (p->info >= inform))
      insert_first(p, inform);
  else {
    nodo* q=p;
    while ((q->next != NULL) &&
              (q->next->info <= inform)) {
       q=q->next;
    }
    nodo* r=new nodo;
    r->info=inform;
    r->next=q->next;
    q->next = r;
  }
}
```

Rimozione di un elemento

- Per cancellare un nodo *t* che segue un nodo dato *x* da una lista concatenata, cambiamo x->next in modo che punti a t->next.
  - Il nodo *t* può essere usato per riferirsi al nodo rimosso (e.g. per **deallocarlo**).



Rimozione di un elemento

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

t inizializzato a x->next

x-next punta a t->next

t ritornato per poter ad esempio essere deallocato

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

Rimozione di un elemento

```
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;
}
```

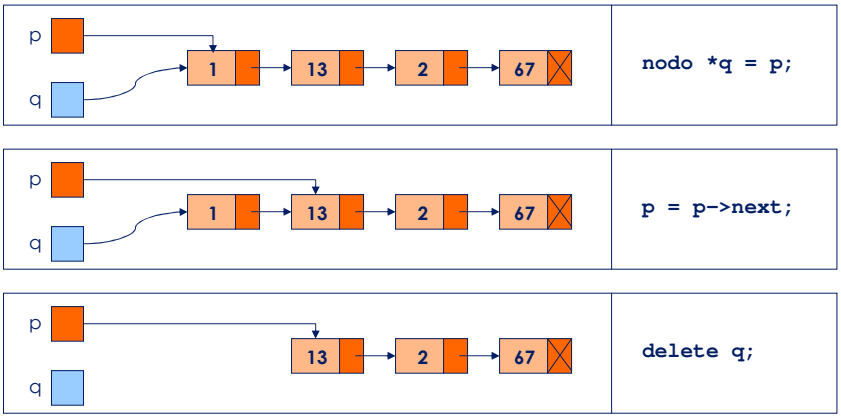
Variabile temporanea per memorizzare nodo rimosso

Deallocazione del nodo rimosso.

Deallocazione del nodo x iniziale.

Rimozione di un elemento in testa

- Vogliamo eliminare il primo elemento della lista.



Rimozione di un elemento in testa

- Se dobbiamo rimuovere il primo elemento 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, la lista nuova è ritornata dalla funzione.
- Nota: il nodo rimosso deve essere deallocato.

Rimozione di un elemento in testa

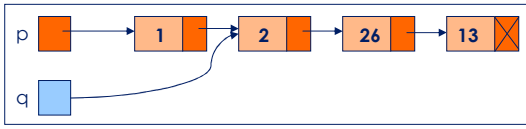
```
void remove_first(nodo * s)  {
 nodo * n = s;
 if (s != NULL) {
   s = s->next;
   delete n;
 }
}
```

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

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

Rimozione di un elemento particolare

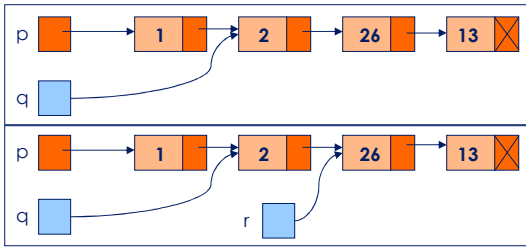
- Vogliamo cercare un elemento che contiene nel campo info un determinato valore (es. 26) ed eliminarlo.




```
nodo* q=p;

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

            ...

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


Rimozione di un elemento particolare

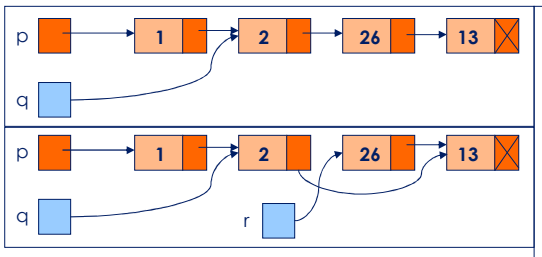
- Vogliamo cercare un elemento che contiene nel campo info un determinato valore (es. 26) ed eliminarlo.




```
nodo* q=p;

while(q->next!=NULL){
 if(q->next->info==26){
   nodo * r = q->next;
            ...

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

Rimozione di un elemento particolare

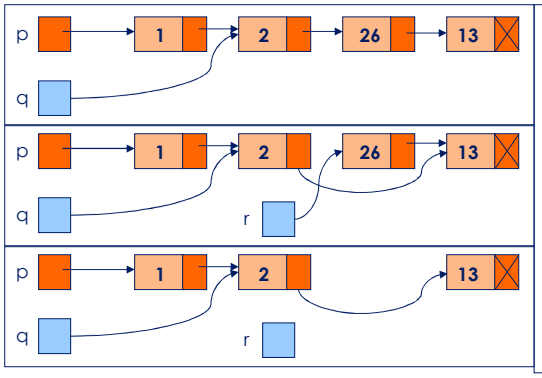
- Vogliamo cercare un elemento che contiene nel campo info un determinato valore (es. 26) ed eliminarlo.




```
nodo* q=p;

while(q->next!=NULL){
 if(q->next->info==26){
  nodo * r = q->next;
  q->next=q->next->next;

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


Rimozione di un elemento particolare

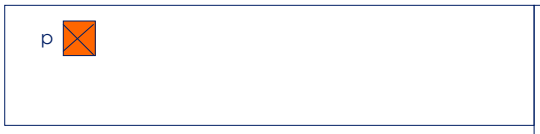
- Vogliamo cercare un elemento che contiene nel campo info un determinato valore (es. 26) ed eliminarlo.




```
nodo* q=p;

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

Rimozione di un elemento particolare

- Vogliamo cercare un elemento che contiene nel campo info un determinato valore (es. 26) ed eliminarlo.

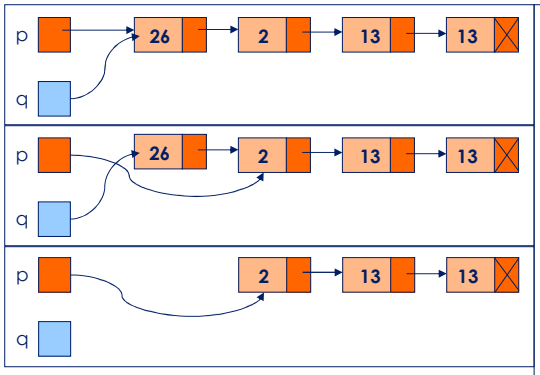



```
if (p != NULL) {
 nodo* q=p;

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

I caso limite: lista vuota!

Rimozione di un elemento particolare

- Vogliamo cercare un elemento che contiene nel campo info un determinato valore (es. 26) ed eliminarlo.

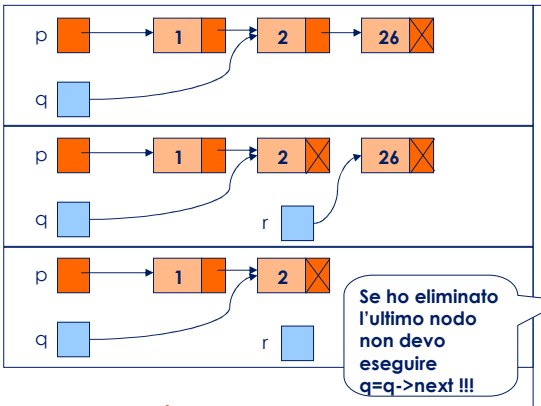



```
if (p != NULL) {
 nodo* q=p;
 if (q->info == 26) {
   p = p->next;
   delete q;
 }
 else {
  while(q->next!=NULL){
   if(q->next->info==26){
    nodo * r = q->next;
    q->next=q->next->next;
    delete r;
   }
   q=q->next;
  }
 }
}
```

II caso limite: è il primo nodo quello da eliminare!

Rimozione di un elemento particolare

- Vogliamo cercare un elemento che contiene nel campo info un determinato valore (es. 26) ed eliminarlo.



III caso limite: è l'ultimo nodo quello da eliminare!

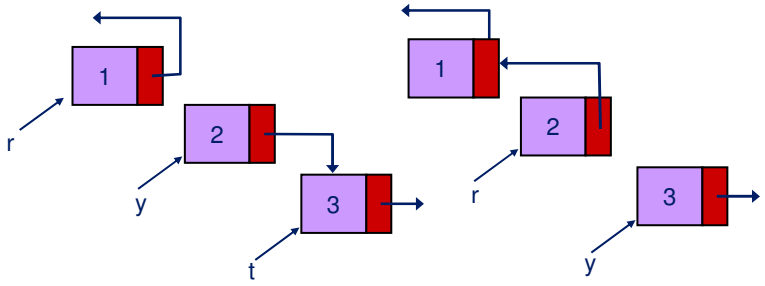
```
if (p != NULL) {
 nodo* q=p;
 if (q->info == 26) {
   p = p->next;
   delete q;
 }
 else {
  while(q->next!=NULL){
   if(q->next->info==26){
    nodo * r = q->next;
    q->next=q->next->next;
    delete r;
    return;
   }
   q=q->next;
  }
 }
}
```


Rimozione di un elemento particolare

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

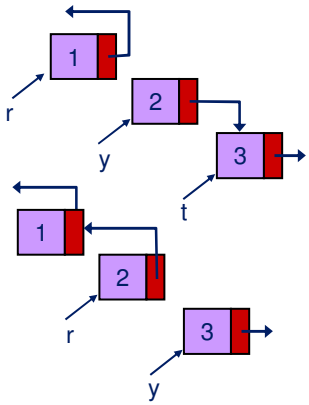
Rovesciamento di una lista

- 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

```
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;
}
```

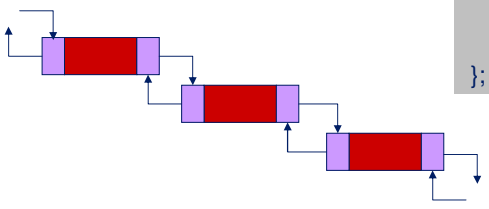


Esercizi

- Scrivere una funzione che determini l'oggetto *t* che precede un dato oggetto *x*. Se *x* primo elemento ritornare *x* stesso.
- Scrivere una funzione che ritorni true se un elemento *x* occorre nella lista, false se l'elemento non occorre.
- Scrivere una funzione che costruisca una copia di una data lista (cioè, una nuova lista che contiene le stesse informazioni nello stesso ordine).
- Scrivere una funzione che sposti l'elemento più grande di una lista concatenata nell'ultimo nodo della lista.
- Scrivere una funzione che sposti l'elemento più piccolo di una lista concatenata nel primo nodo della lista.
- Scrivere una funzione che costruisca la lista risultante dalla concatenazione di due liste *x* e *y*.
  - Diverse soluzioni sono possibili:
    - Side effects sulla lista destinazione.
    - Nuova lista.

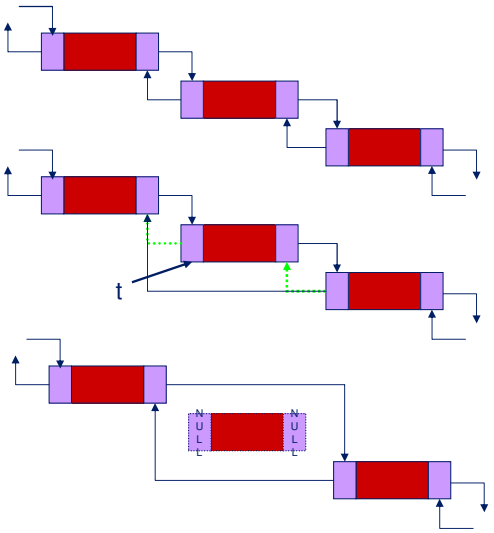
Liste doppiamente concatenate

- Derivano dalle liste definite in precedenza.
  - differiscono per la presenza di un ulteriore puntatore al nodo che lo precede.

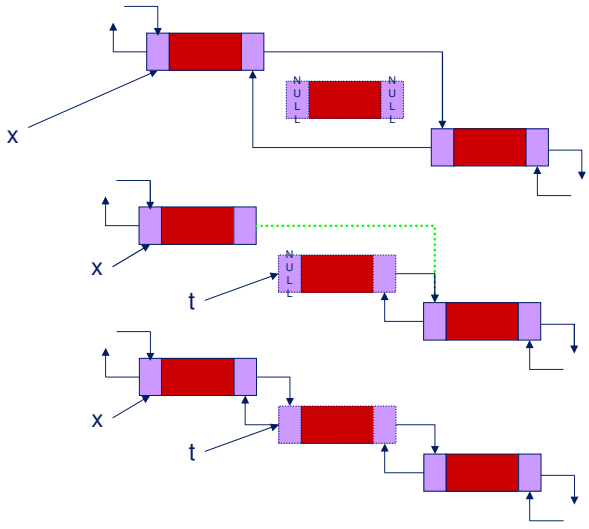


```
struct node {
   int n;
   node * prev;
   node * next;
   node(int x, node * p, node * n) {
      n = x; prev = p; next = n; }
};
```

Cancellazione in una lista doppiamente collegata



Inserimento in una lista doppiamente collegata



Esercizi

- Scrivere una funzione che cancella un elemento *x* da una lista doppiamente concatenata *s.*
- Scrivere una funzione che inserisce un elemento *x* in una lista doppiamente concatenata *s.*

Esercizio: problema di Giuseppe Flavio

- Immaginiamo che N persone debbano eleggere un *leader*.
- L'elezione avviene nel seguente modo:
  - le persone si dispongono in cerchio, eliminando una persona ogni M, seguendo l'ordine del cerchio, e richiudendo il cerchio ad ogni eliminazione.
- Il problema è quello di scoprire quale persona rimarrà per ultima.
- O più in generale è quello di determinare l'ordine in cui le persone verranno eliminate.

Esercizio: problema di Giuseppe Flavio