



UNIVERSITY OF TRENTO - Italy

---

# Laboratorio 17

DISI – aa 2021/22

**Pierluigi Roberti**  
**Carmelo Ferrante**

Liste Semplicemente Concatenate  
Generali e Ordinate

Università degli Studi di Trento - Dipartimento DISI

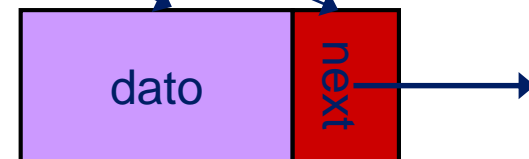
# Liste semplicemente concatenate

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 struct  
typedef struct Tnodo {  
    Tdato dato;  
    Tnodo* next;  
} Tnodo;
```

Campo contenente le informazioni da memorizzare nel nodo

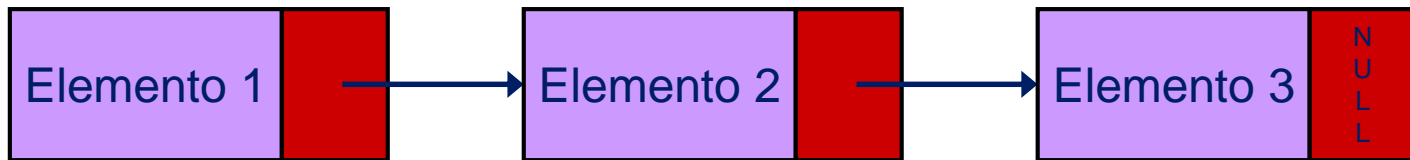
Puntatore al nodo successivo.



# Liste semplicemente concatenate

Per il nodo finale adotteremo le seguenti convenzioni:

- È un *link nullo* che non punta ad alcun nodo (e.g. **NULL**).



Inserimento dati in lista:

- In testa
- In coda
- In posizione ordinata
- In posizione generica

Cancellazione dati da lista:

- In testa
- In coda
- In posizione data

# Realizzazioni LIFO – FIFO

---

## Stack – Pila – LIFO: Last In – First Out

- Opzione 1:
  - Inserimento: insert\_first → push
  - Rimozione: remove\_first → pop
- Opzione 2
  - Inserimento: insert\_last → push
  - Rimozione: remove\_last → pop

## Coda – FIFO: First In – First Out

- Opzione 1:
  - Inserimento: insert\_first → put
  - Rimozione: remove\_last → get
- Opzione 2
  - Inserimento: insert\_last → put
  - Rimozione: remove\_first → get

# Esercizio 1


## struttura file .h e .cpp per liste

```
typedef struct Tdato{
    // Add dati
    // add costruttori, distruttore, stampa
} Tdato;

typedef struct Tnodo{
    Tdato dato;
    Tnodo* next;

    Tnodo(){ dato = Tdato(); next = NULL; }
    Tnodo(Tdato d){ dato = d; next = NULL; }
    Tnodo(Tdato d, Tnodo* n){ dato = d; next = n; }
    ~Tnodo(){}
    void stampa()const{ dato.Stampa(); }
} Tnodo;

typedef Tdato Dato;
typedef Tnodo Nodo;
typedef Tnodo* Nodoptr;
```



ALIAS

# Esercizio 1 parte 1

## creazione lista di strutture

---

- Modificare la struttura denominata **Tdato**:
  - Aggiungere i campi:
    - `int index;`
    - `float value;`
  - Costruttore di default (attributi con valore pari a 0)
  - Costruttore specifico
  - Distruttore
  - Aggiungere i seguenti metodi:
    - `stampa`  
con formato stampa: `[index-value]`
    - `lt (less than)` – parametro `Tdato`  
confronto per campo valore (attenzione che è di tipo `float`)
    - `gt (greater than)` – parametro `Tdato`  
confronto per campo valore (attenzione che è di tipo `float`)
- Nel main verificare che il codice compili ed esegua in modo corretto

# Esercizio 1 parte 1

## inserimento in lista di strutture

- Implementare le seguenti funzioni:

//inserimento in testa alla lista

Nodoptr insert\_first (Nodoptr s, Tdato d);

//stampa della lista

void stampa (Tnodo\* s);

//conteggio numero elementi della lista

int lung (Nodoptr s);

//inserimento in coda alla lista

Nodoptr insert\_last (Nodoptr s, Tdato d);

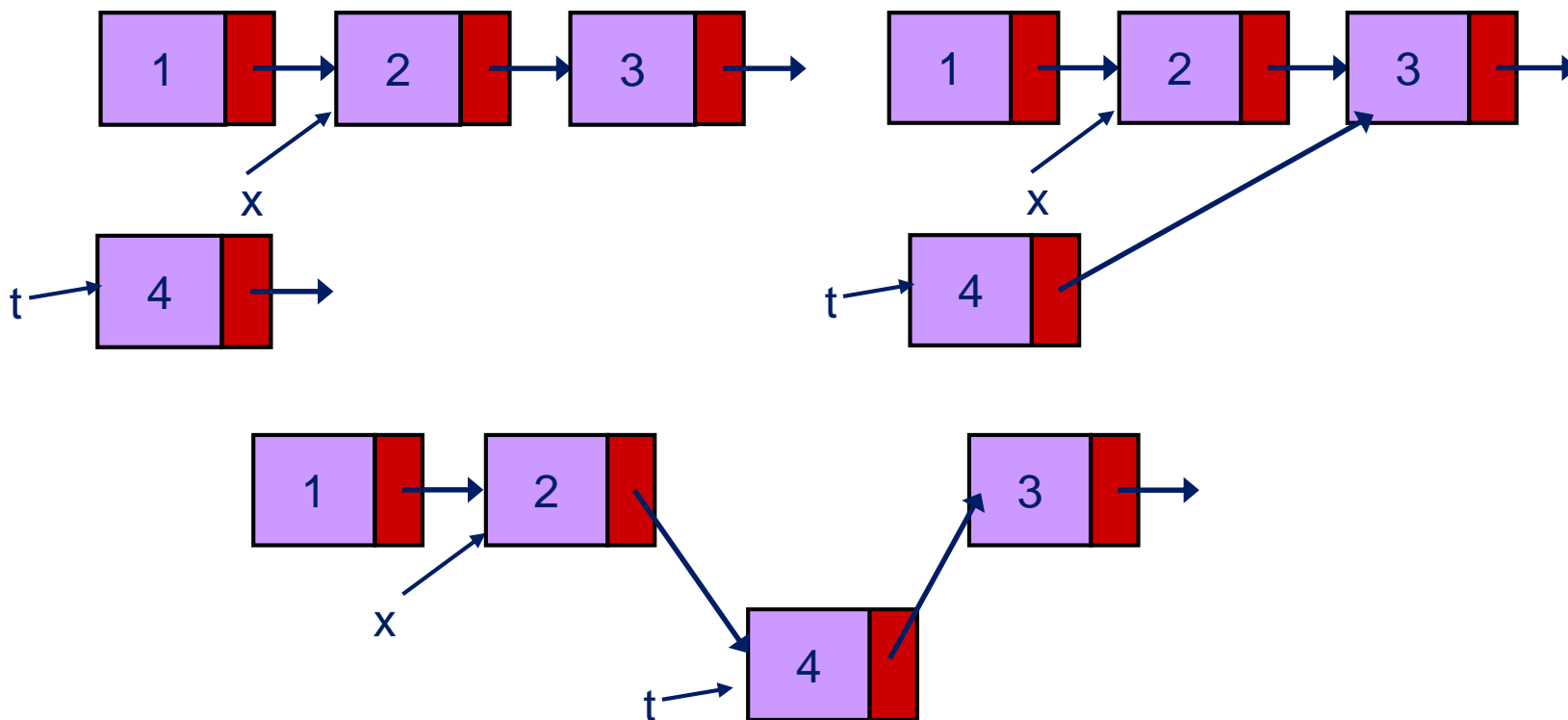
Nodoptr == Tnodo\*

```
Tdato d;  
Nodoptr x = NULL;    //Tnodo* x = NULL;  
d.index = 2; d.value = 10.4;  
x = insert_first (x,d);  
stampa(x);  
d.index = 3; d.value = -1.5;  
x = insert_last (x,d);  
stampa(x);  
d.index = 4; d.value = 0.0;  
x = insert_first (x,d);  
stampa(x);  
cout << "numero elementi in lista:" << x.lung() << endl;
```

MAIN DI PROVA

# 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  $t \rightarrow \text{next}$  a  $x \rightarrow \text{next}$ , e quindi  $x \rightarrow \text{next}$  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 () {  
    Tnodo* x = new Tnodo();  
    cout << "Inserire numero: ";  
    cin >> x->dato;  
    x->next = NULL;  
    for (int i = 0; i < 10; i++) {  
        Tnodo* t = new Tnodo();  
        cout << "Inserire un numero: ";  
        cin >> t->dato;  
        t->next = NULL;  
        insert_node(x, t);  
    }  
  
    for (Tnodo* 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

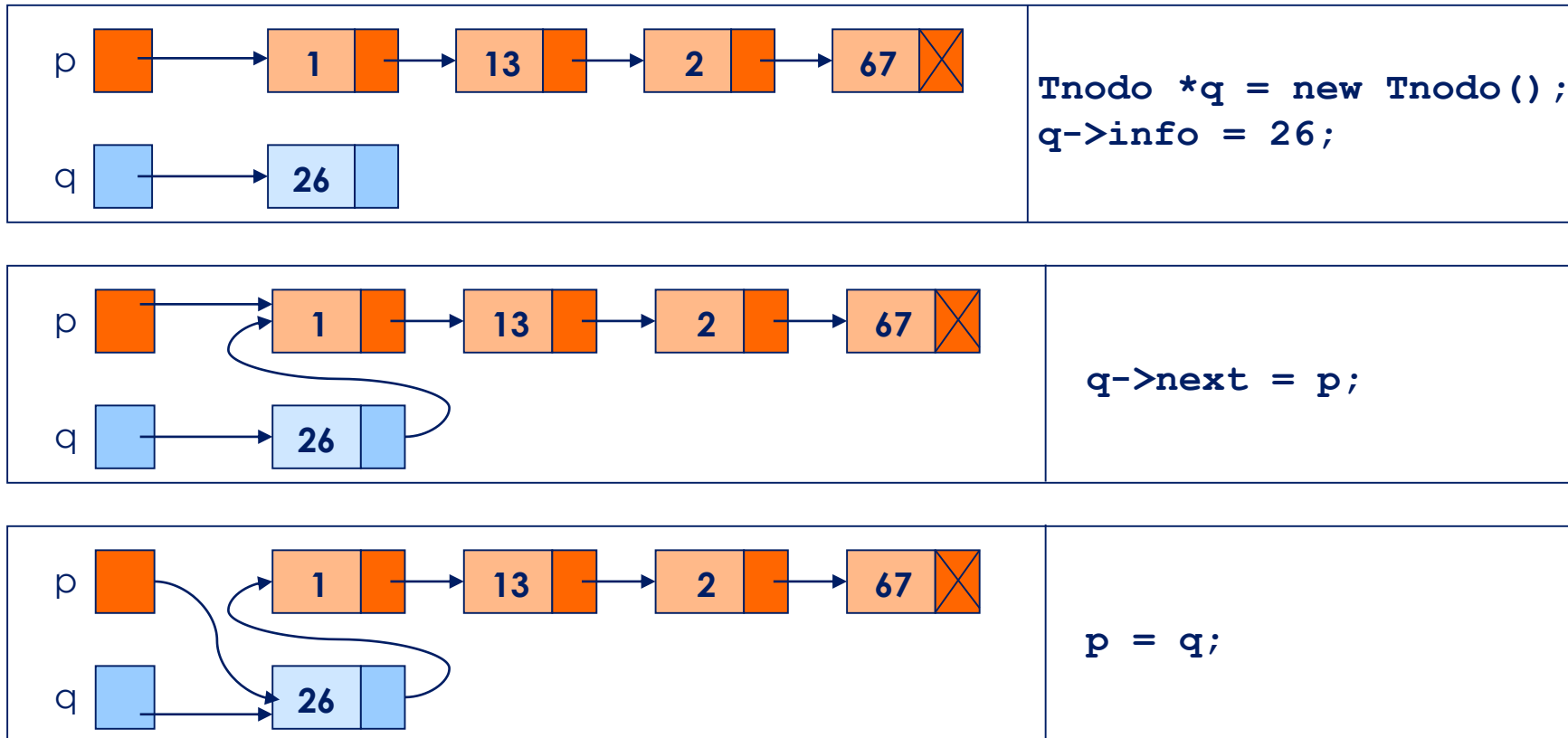
Inserzione del nuovo  
elemento t  
successivamente al  
Nodo iniziale x.

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

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

Nodo con valore intero

```
Tnodo* insert_first (Nodo* s, Tdato d) {  
    Tnodo* n = new Tnodo();  
    n->dato = d;  
    n->next = s;  
    return n;  
}
```

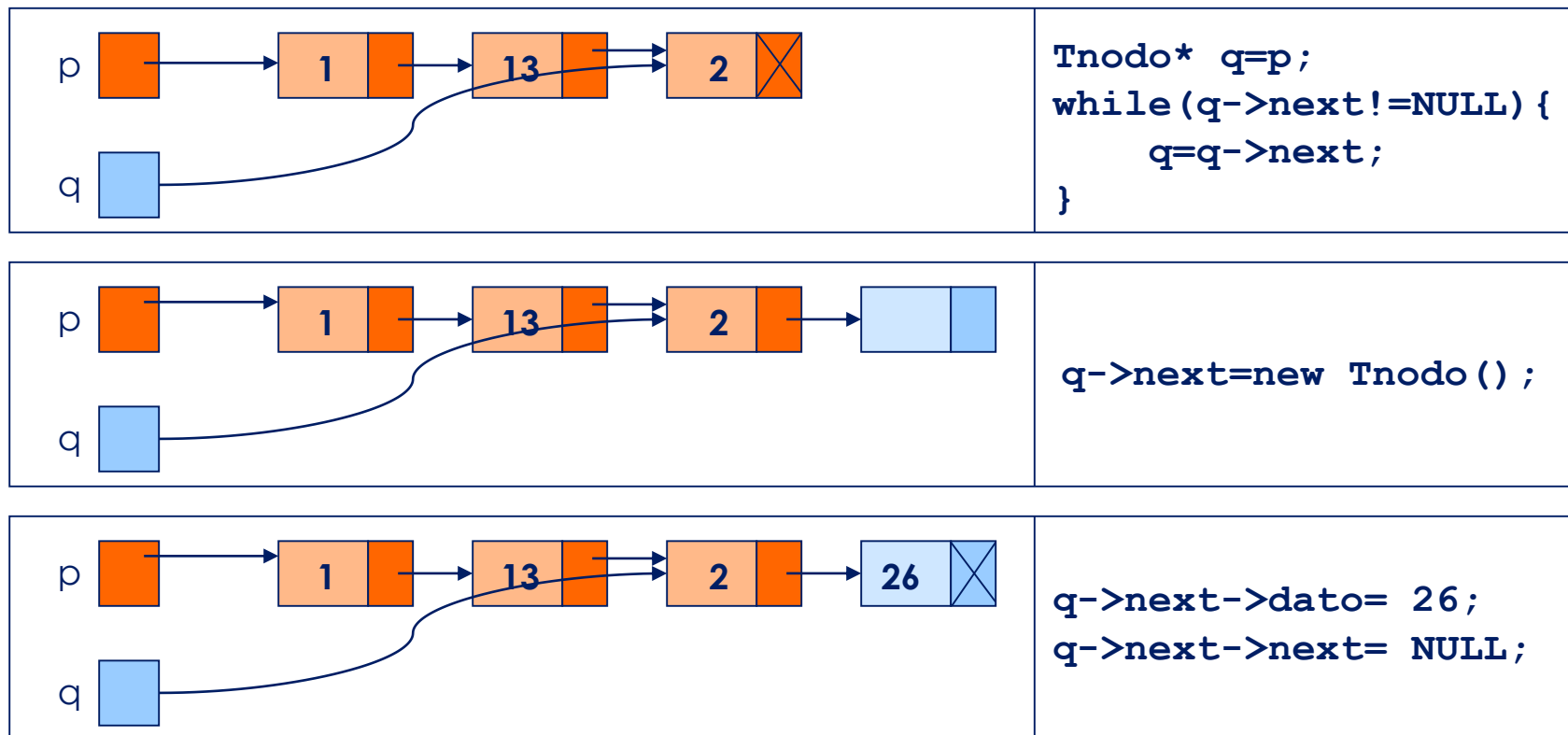
Nodo con valore  
strutturato

```
Tnodo* insert_first (Tnodo * s, Tdato d) {  
    Tnodo* n = new Tnodo(d);  
    n->next = s;  
    return n;  
}
```

```
Tnodo* insert_first (Tnodo * s, Tdato d) {  
    return new Tnodo(d, s);  
}
```

# 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

```
Tnodo* insert_last(Tnodo * p, int d) {  
    Tnodo* r = new Tnodo();  
    r->dato = d;  
    r->next = NULL;  
    if (p != NULL) {  
        Tnodo * q = p;  
        while(q->next != NULL) {  
            q = q->next;  
        }  
        q->next = r;  
    }  
    else {  
        p = r;  
    }  
    return p;  
}
```

## Nodo con valore intero

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 in coda

```
Tnodo* insert_last(Tnodo * p, Tdato d) {  
    Tnodo* r = new Tnodo();  
    r->dato = d;  
    r->next = NULL;  
    if (p != NULL) {  
        Tnodo * q = p;  
        while(q->next != NULL) {  
            q = q->next;  
        }  
        q->next = r;  
    }  
    else {  
        p = r;  
    }  
    return p;  
}
```

Nodo con valore  
strutturato

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 in coda

```
Tnodo* insert_last(Nodo * p, Tdato d) {  
    Tnodo* r = new Tnodo(d);
```

Nodo con valore  
strutturato

Allocazione del nuovo  
Nodo.

```
    if (p != NULL) {  
        Tnodo * q = p;  
        while(q->next != NULL) {  
            q = q->next;  
        }
```

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

q qui è garantito essere  
diverso da NULL

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

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

```
    }  
    else {  
        p = r;
```

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

```
    }  
    return p;  
}
```



# Inserimento di un elemento in coda

```
Tnodo* insert_last(Tnodo * p, Tdato d) {  
    if (p == NULL) {  
        p = new Tnodo(d);  
        return p;  
    }  
    Tnodo* q = p;  
    while(q->next != NULL) {  
        q = q->next;  
    }  
    q->next = new Tnodo(d);  
    return p;  
}
```

Nodo con valore  
strutturato

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 in coda

```
Tnodo* insert_last(Tnodo * p, Tdato d) {  
    if (p == NULL) {  
        return new Tnodo(d);  
    }  
    Tnodo * q = p;  
    while(q->next != NULL) {  
        q = q->next;  
    }  
    q->next = new Tnodo(d);  
    return p;  
}
```

Nodo con valore  
strutturato

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

# Stampa Lista

```
void stampa (Nodoptr s) {  
    Nodoptr p = s;  
    while (p!=NULL){  
        p->stampa();  
        p = p->next;  
    }  
    cout << endl;  
}
```

Copia del puntatore.  
Per «sicurezza» non si  
modifica il puntatore  
originale, potrebbe  
servire per altre  
operazioni nella  
funzione.  
Nel metodo stampa la  
modifica è  
assolutamente **ERRATA**.

# Esercizio 1 parte 2

## rimozione elementi da lista di strutture

- Implementare le seguenti funzioni:

// rimozione elemento in testa alla lista

Nodoptr remove\_first (Nodoptr s);

// rimozione elemento in coda alla lista

Nodoptr remove\_last (Nodoptr s);

// rimozione elemento particolare nella lista (index == val)

Nodoptr search\_remove (Nodoptr p, int val);

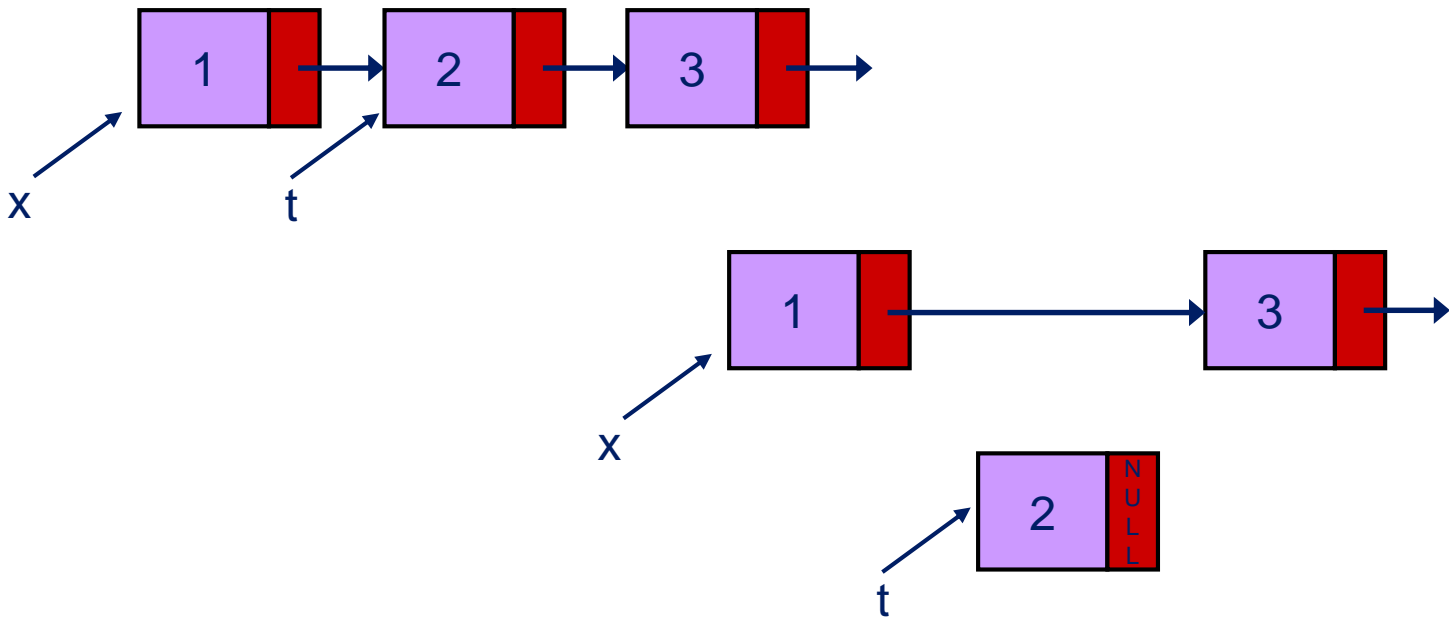
```
//...  
x = remove_first(x);  
stampa(x);  
x = remove_last(x);  
stampa(x);  
x = search_remove(x,4);  
stampa(x);
```

MAIN DI PROVA

# Rimozione di un elemento

Per cancellare un Nodo  $t$  che segue un Nodo dato  $x$  da una lista concatenata, cambiamo  $x \rightarrow \text{next}$  in modo che punti a  $t \rightarrow \text{next}$ .

- Il Nodo  $t$  può essere usato per riferirsi al Nodo rimosso (e.g. per **deallocarlo**).



# Rimozione di un elemento

t inizializzato a x->next

x->next punta a t->next

t ritornato per poter ad esempio essere deallocato

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

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

# Rimozione di un elemento

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

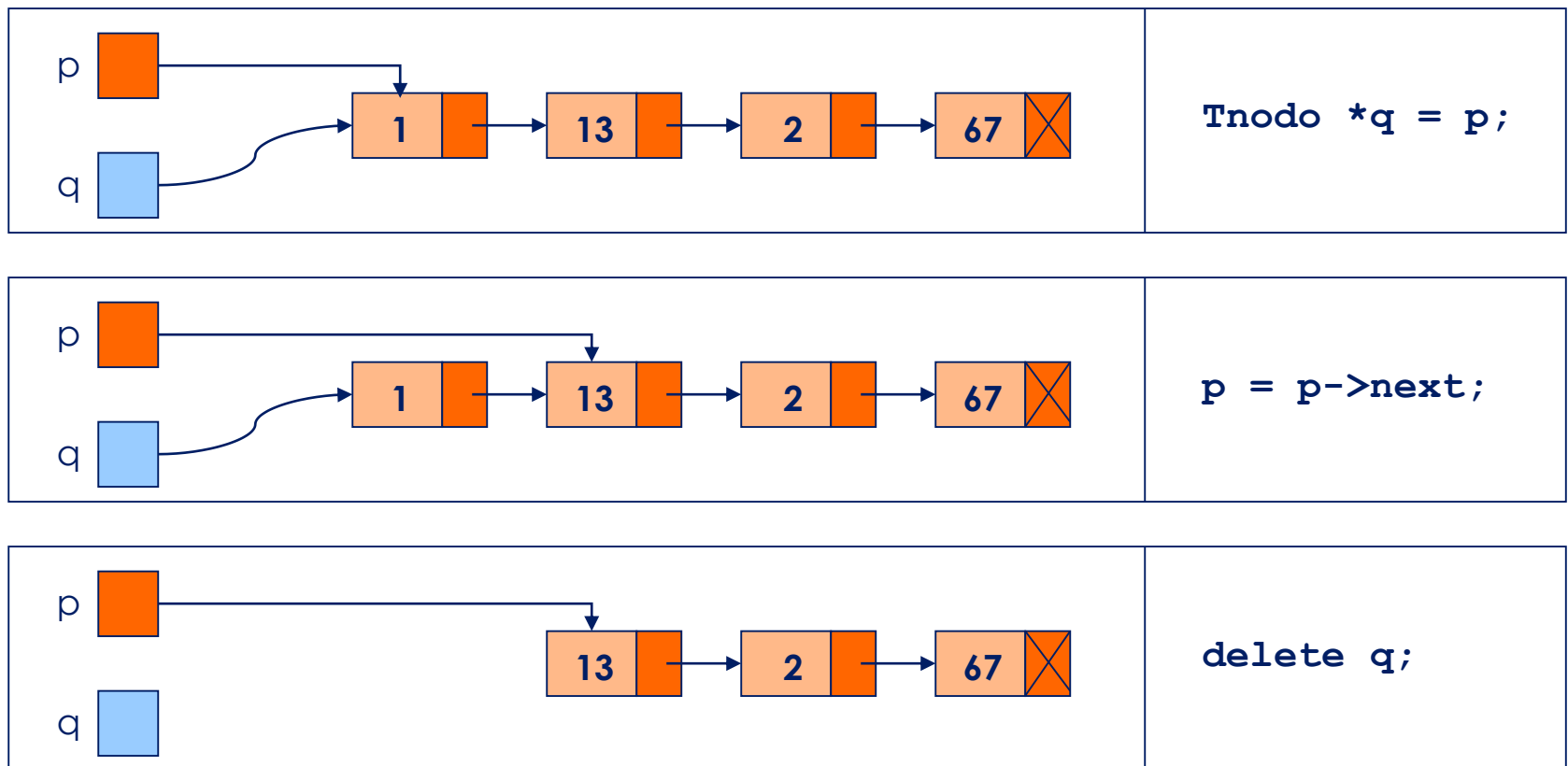
Variabile temporanea  
per memorizzare Nodo  
rimosso

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

```
Tnodo* remove_first (Tnodo* s) {  
    Tnodo* n = s;  
    if (s != NULL) {  
        s = s->next;  
        delete n;  
    }  
    return s;  
} // attenzione all'invocazione
```

```
Nodoptr remove_first (Nodoptr s) {  
    Nodoptr n = s;  
    if (s != NULL) {  
        s = s->next;  
        delete n;  
    }  
    return s;  
} // attenzione all'invocazione
```

```
// invocazione  
my_list = remove_first (my_list);
```

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

Codice  
equivalente

# Rimozione di un elemento in coda

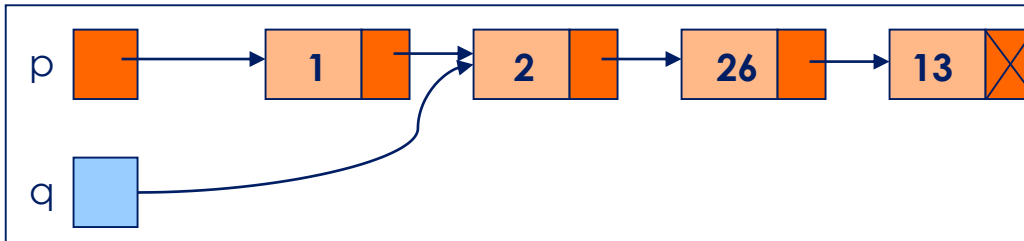
```
Nodoptr remove_last (Nodoptr s) {  
    if (s == NULL) { //caso di lista vuota  
        return s;  
    }  
    Nodoptr q = s;  
    if (q->next == NULL) { //caso di lista con 1 solo elemento  
        delete q;  
        s=NULL;  
    } else {  
        //caso di lista con più elementi  
        while (q->next->next != NULL) {  
            q = q->next;  
        }  
        delete q->next;  
        q->next = NULL; //q->next=q->next->next;  
    }  
    return s;  
}
```

# Rimozione di un elemento in coda

```
Nodoptr remove_last (Nodoptr s) {  
    if (s == NULL) {  
        //caso di lista vuota  
        return s;  
    }  
    Nodoptr q = s;  
    if (q->next == NULL){  
        //caso di lista con 1 solo elemento  
        delete s;  
        return NULL;  
    }  
    //caso di lista con più elementi  
    while (q->next->next != NULL){  
        q = q->next;  
    }  
    delete q->next;  
    q->next = NULL; //q->next=q->next->next;  
    return s;  
}
```

# Rimozione di un elemento particolare

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



```
Tnodo* q=p;
```

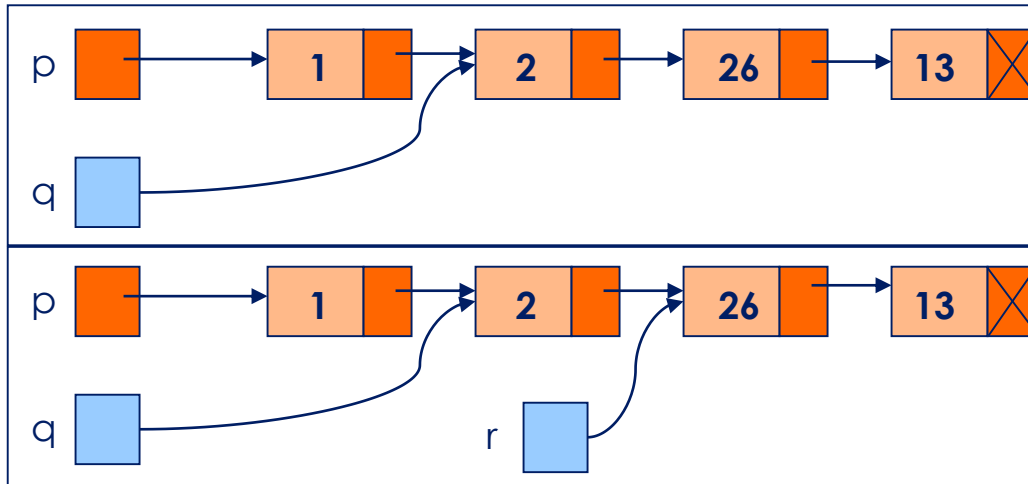
```
while(q->next!=NULL){  
    if(q->next->dato==26){
```

```
        ...
```

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

# Rimozione di un elemento particolare

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



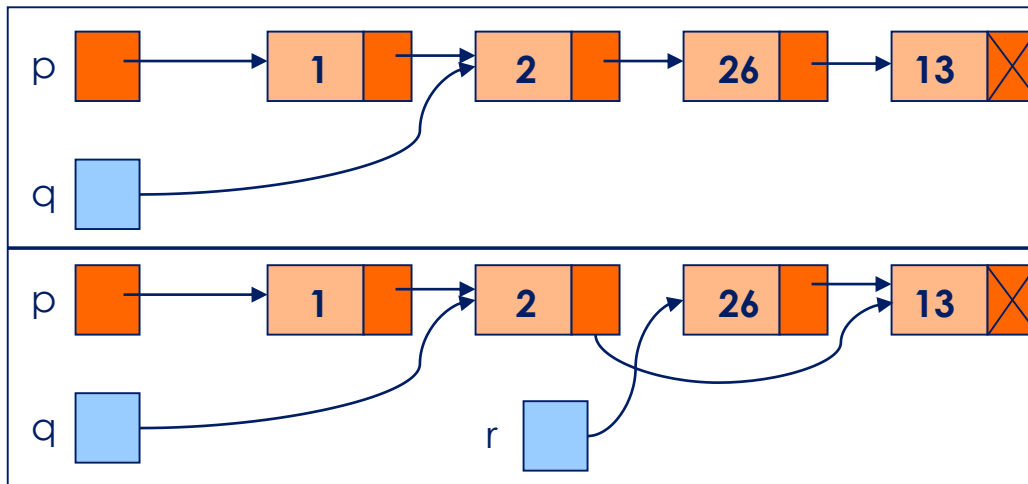
```
Tnodo* q=p;
```

```
while(q->next!=NULL){  
    if(q->next->dato==26){  
        Tnodo * 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.

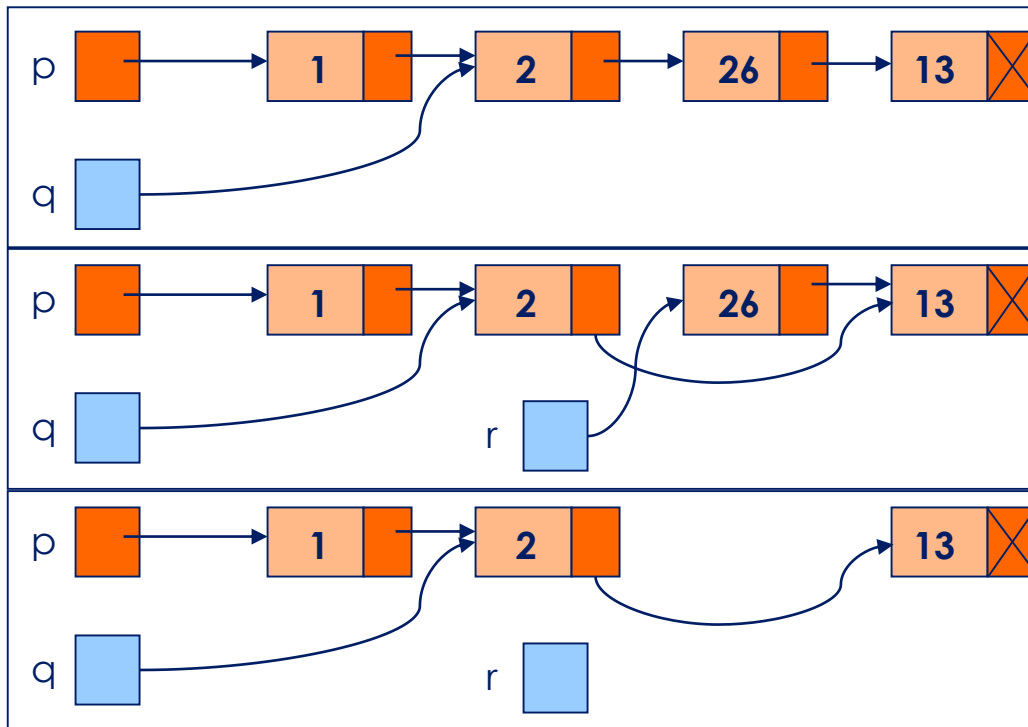


```
Tnodo* q=p;
```

```
while(q->next!=NULL){  
    if(q->next->dato==26){  
        Tnodo * 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.



```
Tnodo* q=p;
```

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

# Rimozione di un elemento particolare

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



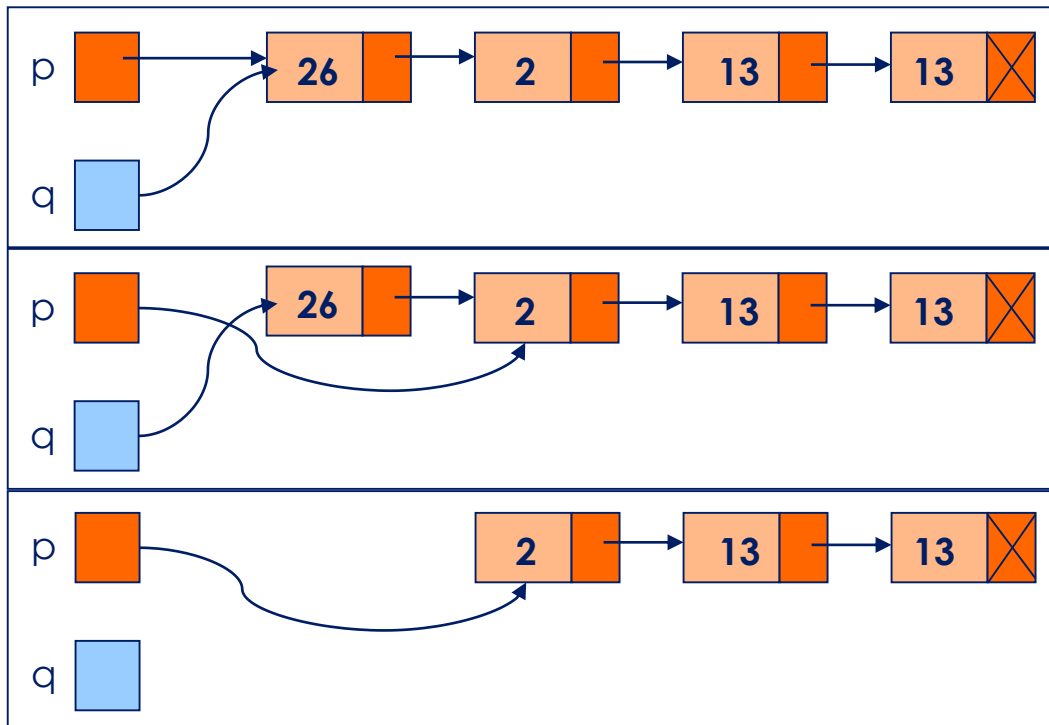
```
if (p != NULL) {  
    Tnodo* q=p;  
  
    while(q->next!=NULL){  
        if(q->next->dato==26){  
            Tnodo * r = q->next;  
            q->next=q->next->next;  
            delete r; return p;  
        }  
        q=q->next;  
    }  
}
```

**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) {  
    Tnodo* q=p;  
    if (q->dato== 26) {  
        p = p->next;  
        delete q; return p;  
    }  
    else {  
        while(q->next!=NULL){  
            if(q->next->dato==26){  
                Tnodo * r = q->next;  
                q->next=q->next->next;  
                delete r; return p;  
            }  
            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.

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

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

# Rimozione di un elemento particolare

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

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

# Rimozione di un elemento particolare

```
Nodoptr search_remove (Nodoptr p, Tdato d){
    if (p != NULL) {
        Nodoptr q = p;
        if ( q->dato.eq (d) ) {
            p = p->next;
            delete q; return p;
        }
        else {
            while(q->next != NULL) {
                if ( q->next->dato.eq(d) ) {
                    Nodoptr r = q->next;
                    q->next = q->next->next;
                    delete r;
                    return p;
                }
                // if (q->next != NULL)
                q=q->next;
            }
        }
    }
    return NULL;
}
```

## bool eq (Tdato d)

Metodo eq -equal- di Tdato restituisce true se il dato è uguale al dato d passato come parametro.

Il concetto di «equal» dipende dal tipo di informazioni in Tdato.

# Rimozione di un elemento particolare

---

```
Nodoptr search_remove(Nodoptr p, Tdato d){
    if (p == NULL)
        return p;
    Nodoptr q = p;
    if ( q->dato.eq(d) ) {
        // p = p->next;
        // delete q; return p;
        return remove_first(p);
    }
    while(q->next != NULL) {
        if ( q->next->dato.eq(d) ) {
            Nodoptr r = q->next;
            q->next = q->next->next;
            delete r;
            return p;
        }
        q=q->next;
    }
    return p;
}
```

# Rimozione di un elemento particolare

---

```
Nodoptr search_remove(Nodoptr p, Tdato d){
    if (p == NULL)
        return p;
    Nodoptr q = p;
    if ( q->dato.eq(d) ) {
        return remove_first(p);
    }
    while(q->next != NULL) {
        if ( q->next->dato.eq(d) ) {
            // Nodoptr r = q->next;
            // q->next = q->next->next;
            // delete r;
            q->next = remove_first(q->next);
            return p;
        }
        q=q->next;
    }
    return p;
}
```

# Esercizio 1 parte 3

## creazione lista ordinata di strutture

---

Aggiungere all'esercizio precedente le seguenti funzioni:

(1) `insert_order`: inserisce in ordine crescente per il campo `value` (usare metodo `gt` per il confronto)

`Nodoptr insert_order (Nodoptr s, Tdato d);`

(2) `search_remove`: rimuove un (o tutti) elemento che corrisponde al dato passato `d`, attenzione che il campo `value` è di tipo `float`!!!!

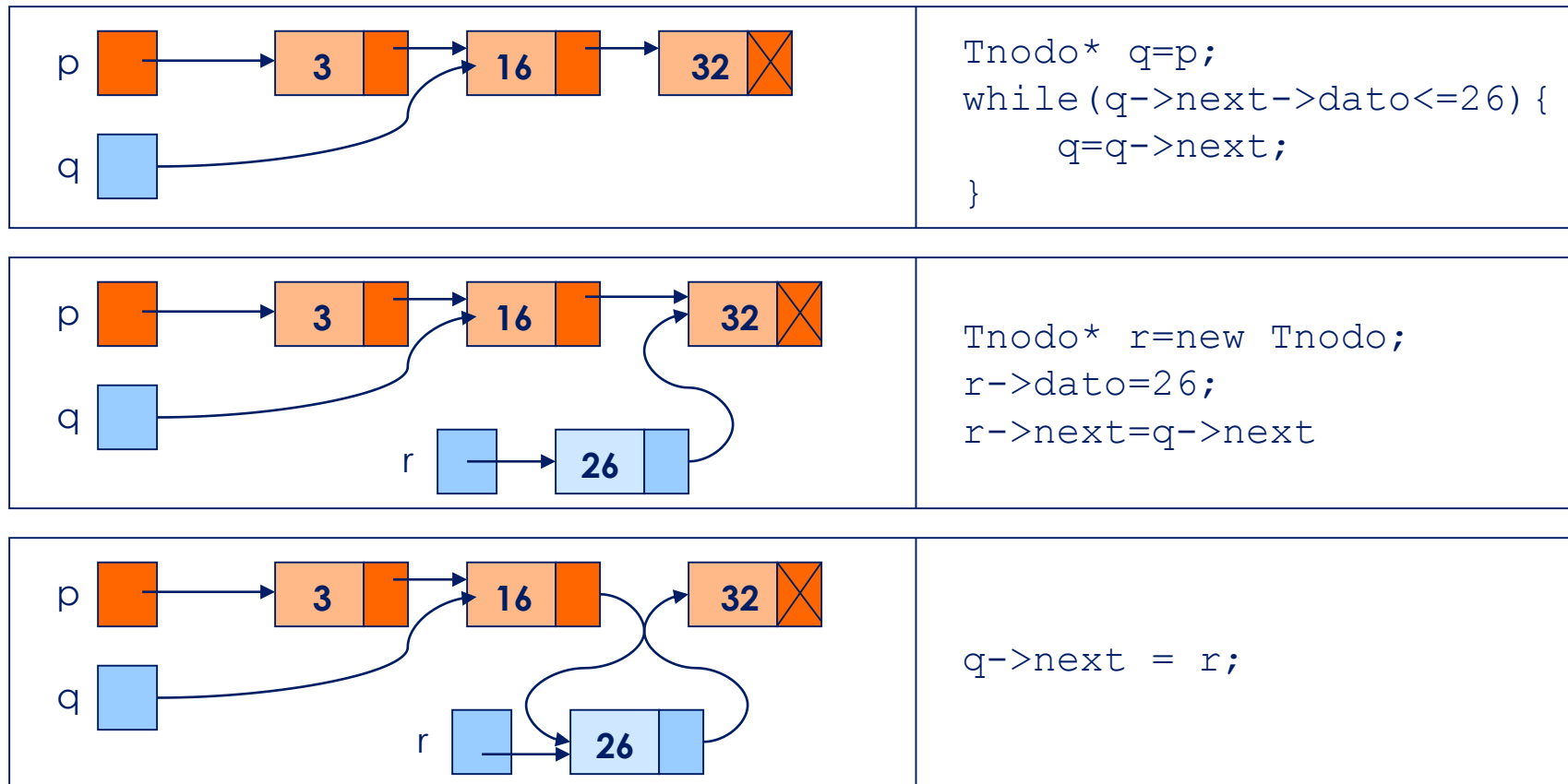
`Nodoptr search_remove (Nodoptr s, Tdato d);`

(3) `remove_cond`: rimuove tutti gli elementi in cui il campo `index` contiene un valore pari

`Nodoptr remove_cond (Nodoptr s);`

# 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!**



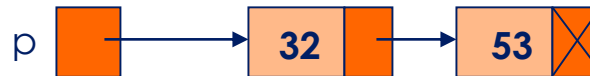
# Inserimento di un elemento ordinato

## ■ Primo caso limite: dobbiamo inserire l'elemento in testa

- Perchè la lista è vuota



- oppure perchè tutti gli altri elementi hanno un valore maggiore

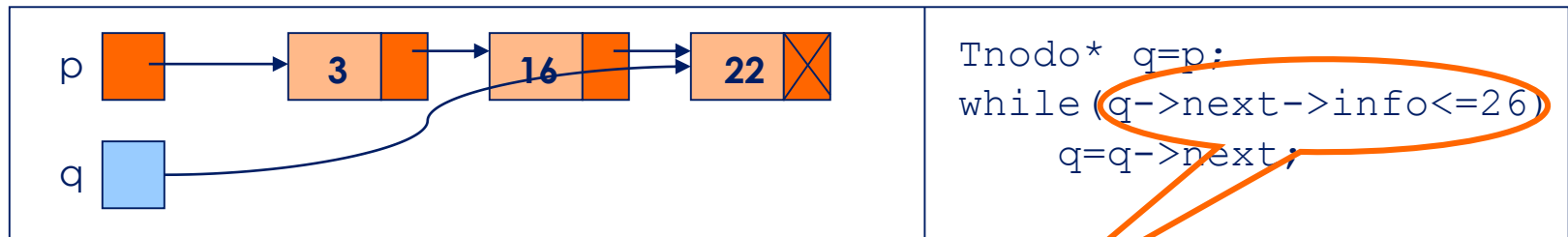


```
if((p==NULL) || (p->dato>= 26))  
    insert_first(p, 26);
```

# 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!**

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

# Inserimento di un elemento ordinato

VER 1

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

# Inserimento di un elemento ordinato

VER 2

```
Nodoptr insert_order(Nodoptr p, Tdato d){
    if ((p==NULL) || p->dato.gt(d) ) {
        p = insert_first(p, d);
    } else
    {
        Nodo* q=p;
        while ((q->next != NULL) && q->next->dato.lt(d))
        { q=q->next; }
        // Nodo* r=new Nodo(d);
        // r->dato = d;
        Tnodo* r=new Tnodo(d);
        r->next=q->next;
        q->next = r;
    }
    return p;
}
```

bool gt(Tdato d)

Metodo gt -greater than- di Tdato restituisce true se il dato e' maggiore del dato d passato come parametro.

Il concetto di «greater than» dipende dal tipo di informazioni in Tdato.

bool lt(Tdato d)

Metodo lt -less than- di Tdato restituisce true se il dato e' minore del dato d passato come parametro.

Il concetto di «less than» dipende dal tipo di informazioni in Tdato.

# Inserimento di un elemento ordinato

VER 3

```
Nodoptr insert_order(Nodoptr p, Tdato d){  
    if ((p==NULL) || p->dato.gt(d) ) {  
        p = insert_first(p, d);  
    } else {  
        Nodoptr q=p;  
        while ((q->next != NULL) && q->next->dato.lt(d)) {  
            q=q->next;  
        }  
        q->next = new Tnodo(d, q->next);  
    }  
    return p;  
}
```

# Inserimento di un elemento ordinato

VER 4

```
Nodoptr insert_order(Nodoptr p, Tdato d){
    if ((p==NULL) || p->dato.gt(d) ) {
        return insert_first(p, d);
    }
    Tnodo* q=p;
    while ((q->next != NULL) && q->next->dato.lt(d)) {
        q=q->next;
    }
    q->next = new Tnodo(d, q->next);
    return p;
}
```

Ordine delle condizioni  
è RILEVANTE!!!

# Inserimento di un elemento ordinato

---

Nota: i seguenti cicli hanno condizioni SIMILI. Differiscono per i casi limite con valore *dato* uguale al valore *d*

```
while ((q->next != NULL) && q->next->dato.lt(d)){ /*...*/ }
```

```
while ((q->next != NULL) && ! q->next->dato.gt(d)){ /*...*/ }
```

```
while ((q->next != NULL) && d.gt(q->next->dato)){ /*...*/ }
```

```
while ((q->next != NULL) && ! d.lt(q->next->dato)){ /*...*/ }
```

# Esercizio 1 parte 4

## lettura elementi da lista di strutture

---

- Implementare le seguenti funzioni:

// controllo se la lista è vuota

bool is\_empty (Nodoptr s);

// controllo se lista ha elementi, fatta in funzione chiamante

// restituisce copia del primo elemento

Tdato get\_first (Nodoptr s);

// controllo se lista ha elementi, fatta in funzione chiamante

// restituisce copia dell'ultimo elemento

Tdato get\_last (Nodoptr s);



# Esercizio 2 – estensione esercizio 1

## Array di liste

Simulazione  
esame

1. Nel main dichiarare un array di 3 liste (variabile denominata **arrayliste**) semplicemente concatenate contenenti dati di tipo Tdato e inizializzare opportunamente gli elementi:  

```
Nodoptr arrayliste[3];  
arrayliste[0] = NULL;  
arrayliste[1] = NULL;  
arrayliste[2] = NULL;
```
2. inserisci (**insert\_order**) 20 elementi casuali (per index e value) scegliendo ogni volta casualmente la lista
3. stampare le 3 liste
4. identificare la coda “più piena”
5. svuotare la coda “più piena” => **remove\_last**
6. inserire man mano i valori cancellati in un file denominato “**lista.txt**” nella forma **[index,value]**

Necessario salvare il dato prima di cancellarlo!!

➔ Funzione “get\_last”

# Esercizio 3

Simulazione  
esame

## utilizzo di struttura lista aggiuntiva

- Possibile utilizzare una struttura Lista che contiene puntatore a lista di elementi di tipo Tnodo (Nodo)
- Esempio: Lista caratterizzata da nome e identificativo;

```
typedef struct Tlista{  
    Tnodo* head;  
    int id;  
    char nome[MAX_NOME];  
  
    Tlista() { head = NULL; }  
    Tlista(Nodoptr h) { head = h; }  
    Tlista(Tdato d) { head = new Tnodo(d); }  
    ~Tlista() {  
        while(head)  
            head = remove_first(head);  
    }  
    //Add ulteriori metodi come stampa. Prima si erano usate  
    //funzioni che possono essere «convertite» in metodi!  
} Tlista;
```