12.4 Dichiarazioni typedef (I)

Parola chiave *typedef*:

- **definisce degli identificatori (detti *nomi typedef*) che vengono usati per riferirsi a tipi nelle dichiarazioni.**

Le dichiarazioni typedef non creano nuovi tipi

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    int i = 1;
    typedef int* intP;
    intP p = &i;
    cout << *p << endl;                          // 1

    typedef int vett[5];                 // vettore di 5 interi
    vett v = {1, 10, 100, 10, 1};
    cout << "v = [" << v[0];
    for (int j = 1; j < 5; j++)
        cout << ' ' << v[j];
    cout << ']' << endl;

    typedef int intero;
    int a = 4;
    intero b = a; // OK, typedef non introduce un nuovo tipo

    cout << a << '\t' << b << endl;              // 4 4

    system("PAUSE");
    return 0;
}
```

```
1
v = [1 10 100 10 1]
4       4
Premere un tasto per continuare . . .
```

13.1 Memoria dinamica (I)

- **Programmi precedenti:**
  - **il programmatore specifica, utilizzando definizioni, numero e tipo delle variabili utilizzate.**
- **Situazioni comuni:**
  - **il programmatore non è in grado di stabilire a priori il numero di variabili di un certo tipo che serviranno durante l'esecuzione del programma.**
  - **Per variabili di tipo array, per esempio, dover specificare le dimensioni (costanti) è limitativo.**
  - **Vorremmo poter dimensionare un array dopo aver scoperto durante l'esecuzione del programma, quanto deve essere grande.**
  - **Per esempio, somma di N numeri inseriti da tastiera, con N letto da tastiera.**
- **Meccanismo della memoria libera (o memoria dinamica):**
  - **risulta possibile allocare delle aree di memoria durante l'esecuzione del programma, ed accedere a tali aree mediante puntatori;**
  - **gli oggetti così ottenuti sono detti *dinamic*i, ed *allocati nella memoria libera*.**

13.1 Memoria dinamica (II)

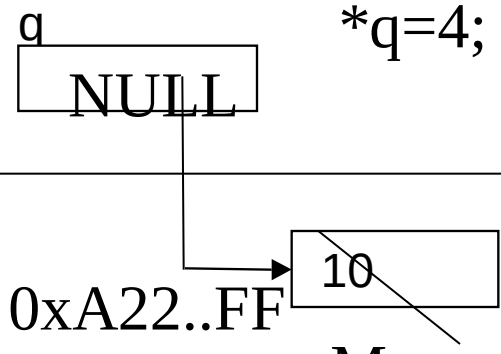
- **Allocazione di oggetti dinamici:**
  - **operatore prefisso *new*:**
    - **ha come argomento il tipo dell'oggetto da allocare;**
    - **restituisce l'indirizzo della memoria ottenuta, che può essere assegnato a un puntatore;**
    - **se non è possibile ottenere la memoria richiesta, restituisce l'indirizzo 0.**

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main ()
{
   int* q;
   q = new int;
   *q = 10;
   cout << *q << endl;          // 10

   int * p;
   int n;
   cin >>n;
   p = new int [n];             // n > 0
   for (int i = 0; i < n; i++)
         p[i] = i;              // anche *(p+i) = i;

    system("PAUSE");
    return 0;
}
```

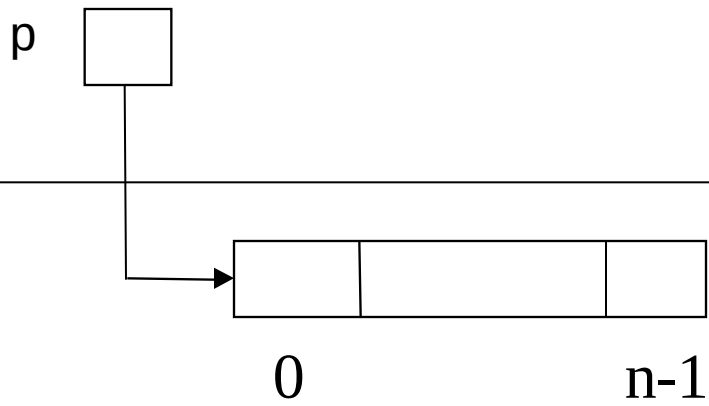
```
q = new int;
delete q;
*q =4;  // SBAGLIATA
```

Memoria stack

```
q=NULL;
*q=4; errore segnalato
        a  tempo di esecuz
```



Memoria dinamica



Memoria dinamica

13.1 Memoria dinamica (III)

- **Buon esito dell'operatore *new*:**
  - **può essere controllato usando la funzione di libreria *set_new_handler()*, dichiarata nel file <new>:**
    - » **questa funzione ha come argomento una funzione *void* senza argomenti, che viene eseguita se l'operatore *new* fallisce (se l'allocazione non è possibile).**

```
#include <cstdlib>
#include <iostream>
#include <new>
using namespace std;
void myhandler()
{
   cerr << "Memoria libera non disponibile" << endl;
   exit(1);
}
int main()
{
   int n;
   set_new_handler(myhandler);
   cout << "Inserisci la dimensione " << endl;
   cin >> n;
   int** m = new int* [n];
   for (int i = 0; i<n; i++)
      m[i] = new int[n];
   system("PAUSE");
   return 0;
}
```

Memoria dinamica (IV)

- **Oggetti allocati nella memoria libera:**

– **esistono finché non vengono distrutti dall'operatore prefisso *delete*:**

  - » **esso ha come argomento un puntatore all'oggetto da distruggere;**
  - » **può essere applicato solo ad un puntatore che indirizza un oggetto allocato mediante l'operatore *new* (in caso contrario si commette un errore).**

- **Se l'operatore *delete* non viene utilizzato:**
  - **– gli oggetti allocati vengono distrutti al termine del programma.**

```
int main()
{  int n = 12;
    int* p = new int(10);
    cout << *p << endl;
    delete p;
    // cout << *p << endl;
    // SBAGLIATO, NON SEGNALA ERRORE
    p = 0;
 // cout << *p << endl;
 // SEGNALA ERRORE A TEMPO DI ESECUZIONE
    int* m = new int [n];
    delete[] m;
//  delete n;
//  ERRORE - oggetto non allocato dinamicamente
   system("PAUSE");
    return 0;
}
```

Eserizio 1

Leggere 100 numeri interi da tastiera e salvarli in memoria.

Soluzione
Dimensione del vettore nota a tempo di compilazione

```
int main() {
 …
 int v[100];
 for (int i=0; i< 100; i++)
    cin >> v[i];
…..
}
```

Esercizio 2

Leggere interi da tastiera e salvarli in memoria.
Numero di interi da leggere inserito
da tastiera.

Soluzione
Dimensione del vettore nota a tempo
di esecuzione.

```
int main() {
 …
int n;
cin >> n;
 int *  v = new int [n];
 for (int i=0; i< n; i++)
    cin >> v[i];
…..
}
```

Esercizio 3

Leggere interi da tastiera e salvarli in memoria.
Inserire il cararattere '.' per terminare.

Soluzione
Il numero di interi che posso inserire non è noto.
Dipende da quando inserisco il carattere '.'

```
int main() {
 …
```

LISTA di elementi, ogni elemento punta all'elemento successivo inserito.

```
Leggo da tastiera
   se ho letto un intero
       creo un nuovo elemento
            e lo aggiungo ai precedenti
…..
}
```

Struttura che punta ad un elemento
dello stesso tipo

```
struct elem {
        int info;
        elem* p;
};
```

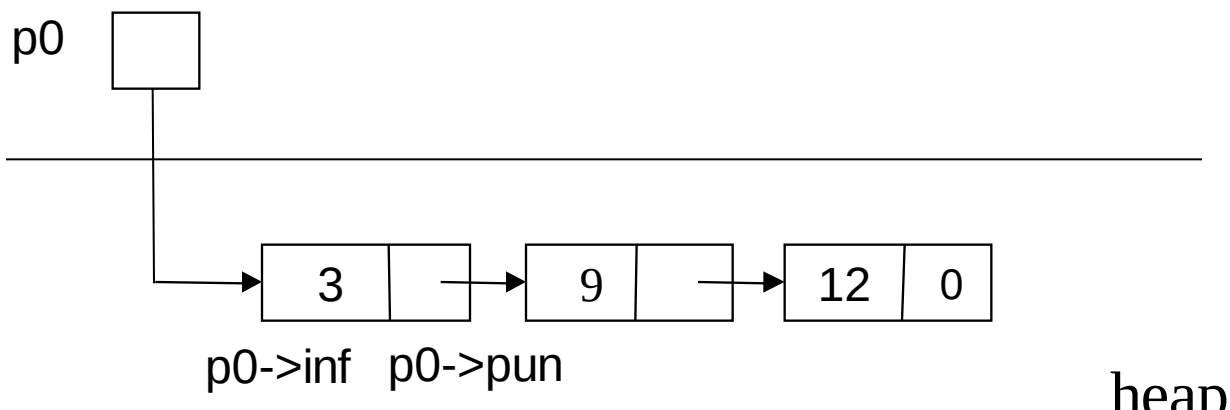
13.2 Liste (I)

Problema: memorizzare numeri inseriti da tastiera finchè non viene inserito il carattere '.'.

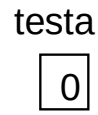
Struttura dati, formata da elementi dello stesso tipo collegati in catena, la cui lunghezza varia dinamicamente.

- **Lista:**
  - **ogni elemento è una struttura, costituita da uno o più campi contenenti informazioni, e da un campo puntatore contenente l'indirizzo dell'elemento successivo;**
  - **il primo elemento è indirizzato da un puntatore (puntatore della lista);**
  - **il campo puntatore dell'ultimo elemento contiene il puntatore nullo.**

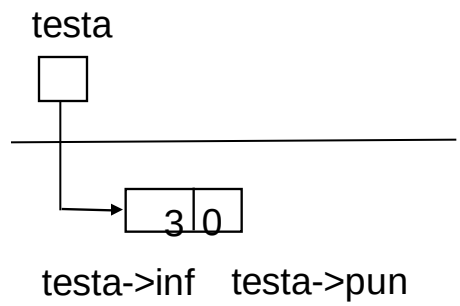
```
typedef int T;
struct elem
{
        T inf;
        elem* pun;
};
```



```
struct elem {
  int inf;
  elem* pun;
};

int main() {

  elem*  testa;
  testa = NULL;  // lista vuota
```

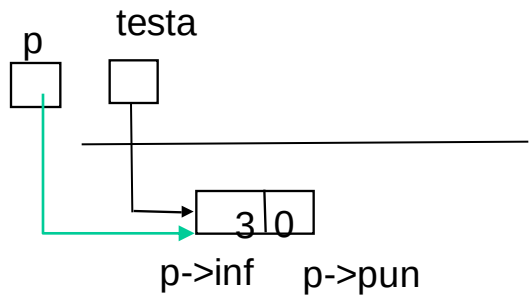



```
testa = new elem;

  cin >> testa->info;
  testa-> pun =NULL;
```

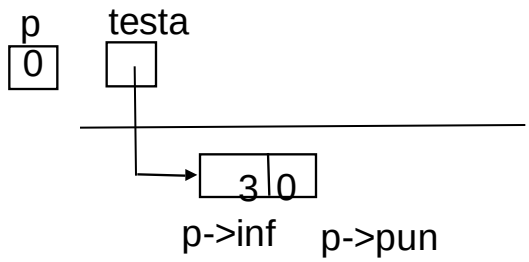



```
    if(testa==NULL)
        cout  <<   "Lista vuota… " << endl;
   else
     cout <<  "Lista non vuota … " << endl;
```

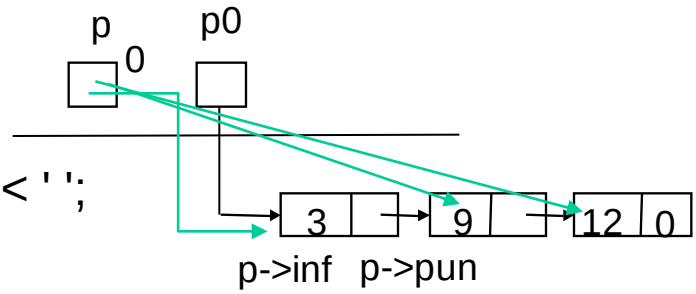
```
elem* p = testa;
    while (p != 0)
    {
        cout << p->inf << ' ';
        p = p->pun;
    }
```



p = p->pun;




```
p= p0;

while (p != 0)
    {
        cout << p->inf << ' ';
        p = p->pun;
    }
```



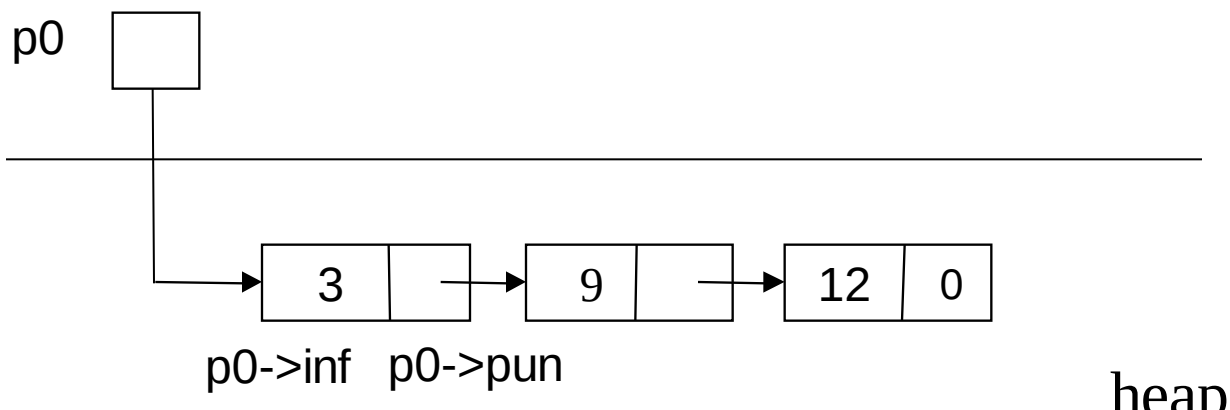
13.2 Liste (I)

Problema: memorizzare numeri inseriti da tastiera finchè non viene inserito il carattere '.'.

Struttura dati, formata da elementi dello stesso tipo collegati in catena, la cui lunghezza varia dinamicamente.

- **Lista:**
  - **ogni elemento è una struttura, costituita da uno o più campi contenenti informazioni, e da un campo puntatore contenente l'indirizzo dell'elemento successivo;**
  - **il primo elemento è indirizzato da un puntatore (puntatore della lista);**
  - **il campo puntatore dell'ultimo elemento contiene il puntatore nullo.**

```
typedef int T;
struct elem
{
        T inf;
        elem* pun;
};
```

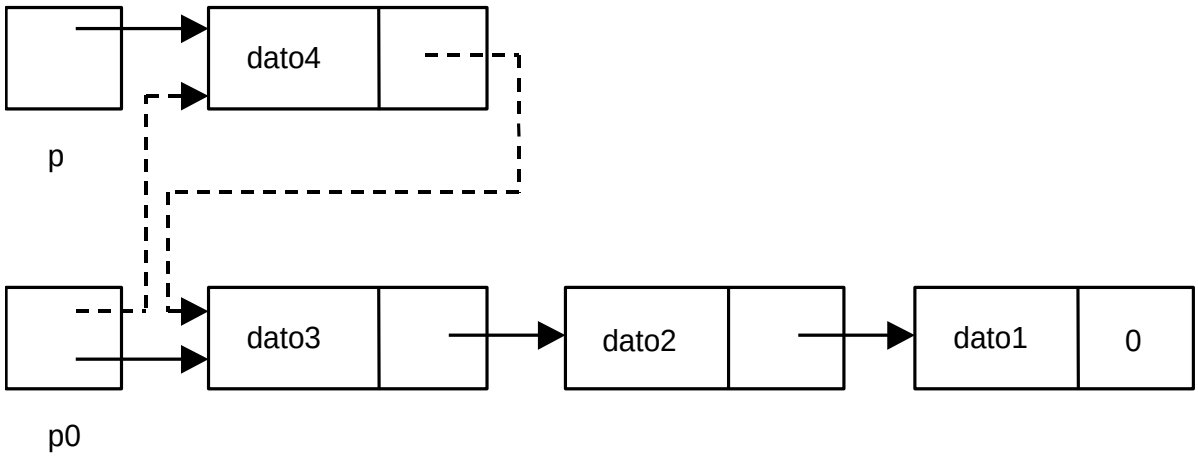


13.2 Liste (II)

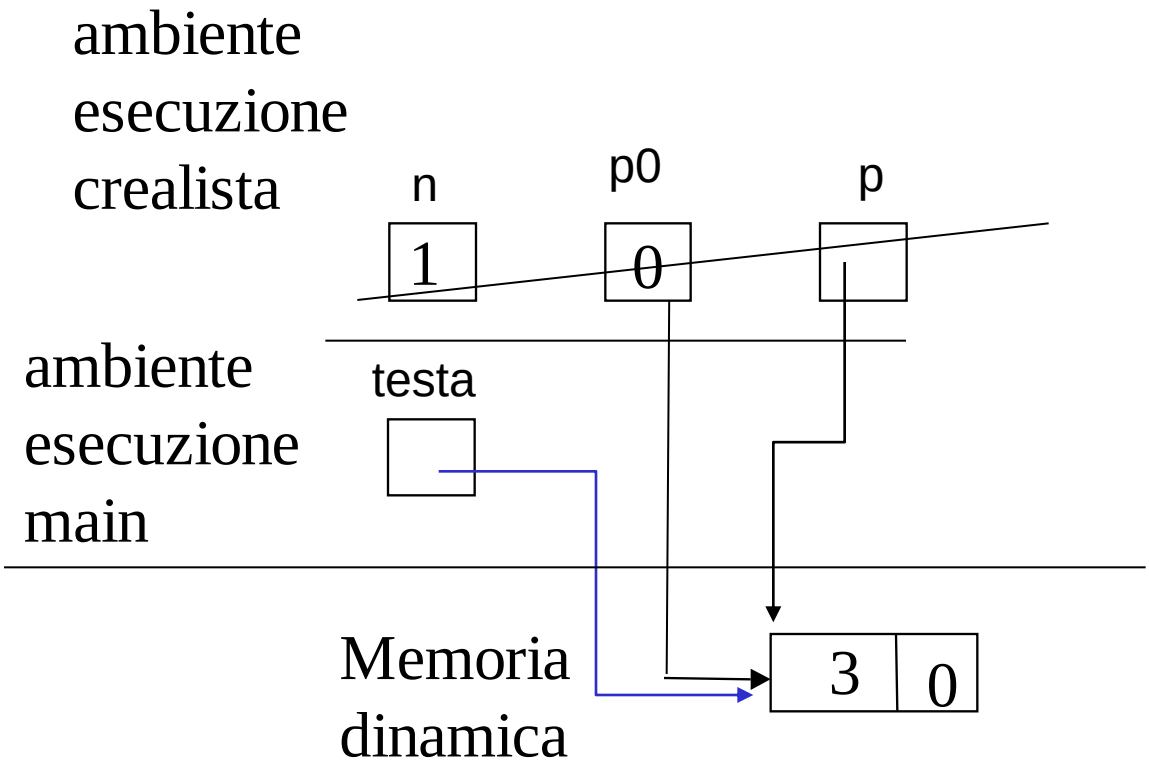
Creazione di una lista

1. Leggere l'informazione
2. Allocare un nuovo elemento con l'informazione da inserire
3. Collegare il nuovo elemento al primo elemento della lista
4. Aggiornare il puntatore di testa della lista a puntare al nuovo elemento

```
typedef elem* lista;          // tipo lista
lista crealista(int n)
{
    lista p0 = 0; elem* p;
    for (int i = 0; i < n; i++)
    {
        p = new elem;
        cin >> p->inf;
        p->pun = p0; p0 = p;
    }
    return p0;
}
```



```
struct elem {
  int info;
  elem* pun;
};

int main() {

  elem*  testa;
  testa = crealista(1);
```




```
}
```

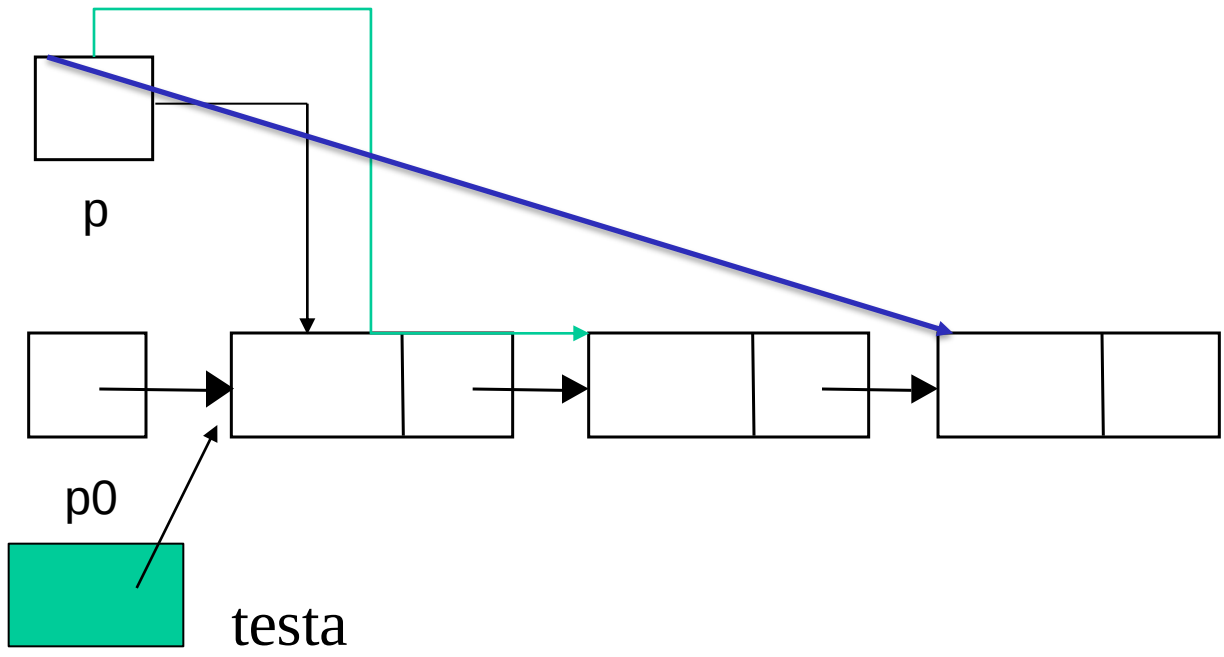
13.2 Liste (III)

Stampa lista

1. Scandire la lista dall'inizio alla fine e per ogni elemento stampare su video il campo informazione

```
void stampalista(elem* p0)
{
    elem* p = p0;
    while (p != 0)
    {
        cout << p->inf << ' ';
        p = p->pun;
    }
}
```

```
int main() {
elem * testa;
….. inserito
      elementi


stampalista(testa);

}
```



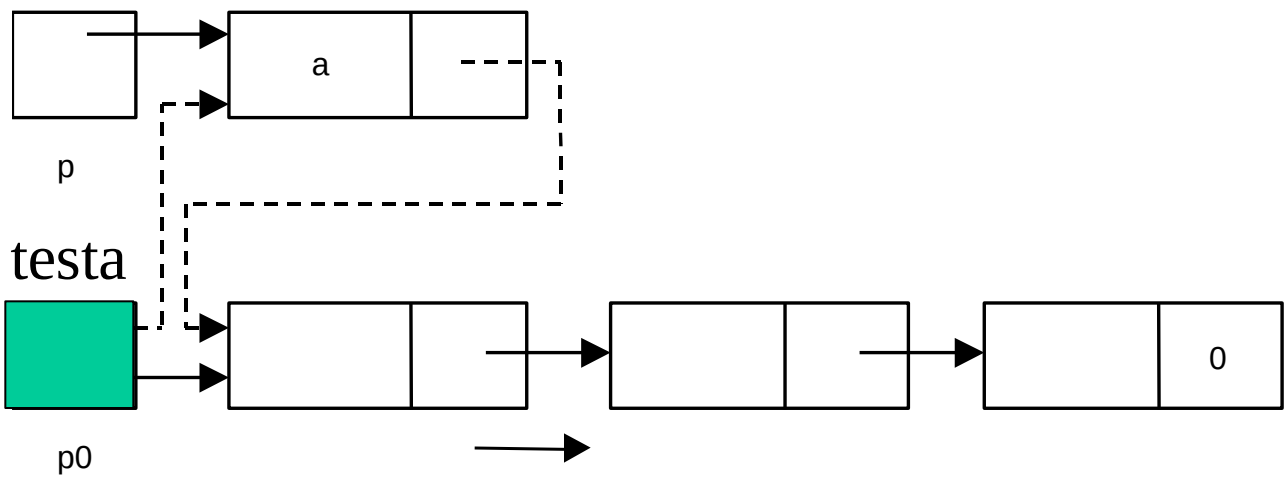
13.2 Liste(IV)

Inserimento in testa

1. Allocare un nuovo elemento con l'informazione da inserire
2. **Collegare il nuovo elemento al primo elemento della lista**
3. **Aggiornare il puntatore di testa della lista**

```
void instesta(elem*& p0, T a)
{
    elem* p = new elem;
    p->inf = a;
    p->pun = p0;
    p0 = p;
}
```

Passaggio argomento per RIFERIMENTO



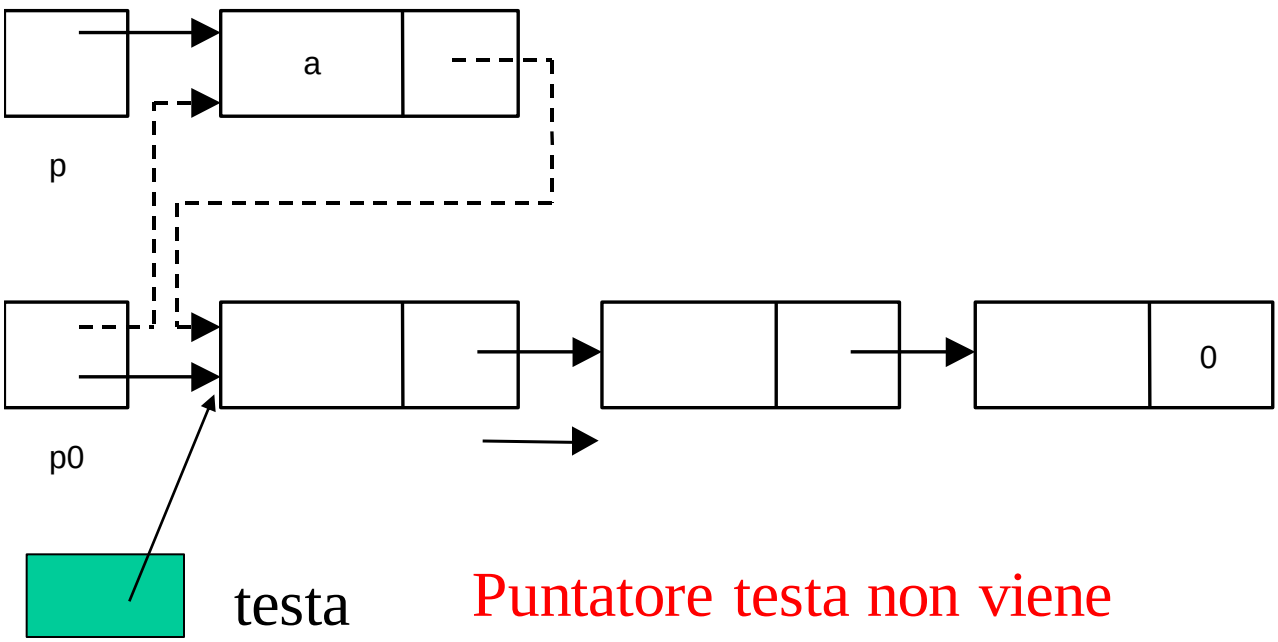
p0 altro nome per testa

Inserimento in testa

1. Allocare un nuovo elemento con l'informazione da inserire
2. **Collegare il nuovo elemento al primo elemento della lista**
3. **Aggiornare il puntatore di testa della lista**

```
void instesta(lista p0, T a)
{
    elem* p = new elem;
    p->inf = a;
    p->pun = p0;
    p0 = p;
}
```

Passaggio argomento per VALORE



Puntatore testa non viene aggiornato

Estrazione dalla testa

Se la lista non è vuota

1. Aggiornare il puntatore di testa della lista
2. Deallocare l'elemento

```
bool esttesta(elem*& p0, int& a)
{     elem* p = p0;
      if (p0 == 0)
            return false;
      a = p0->inf;
      p0 = p0->pun;
      delete p;
      return true;
}
```

Main

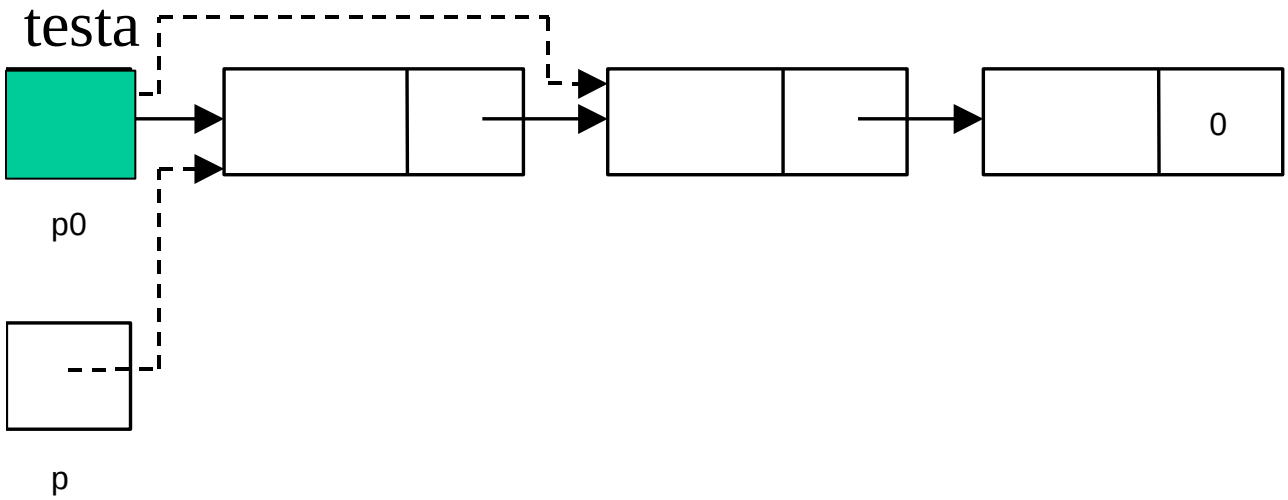
……

elem* testa =NULL;

// aggiunti elementi etc

int val;

esttesta(testa, val);



PROCEDIMENTO GENERALE

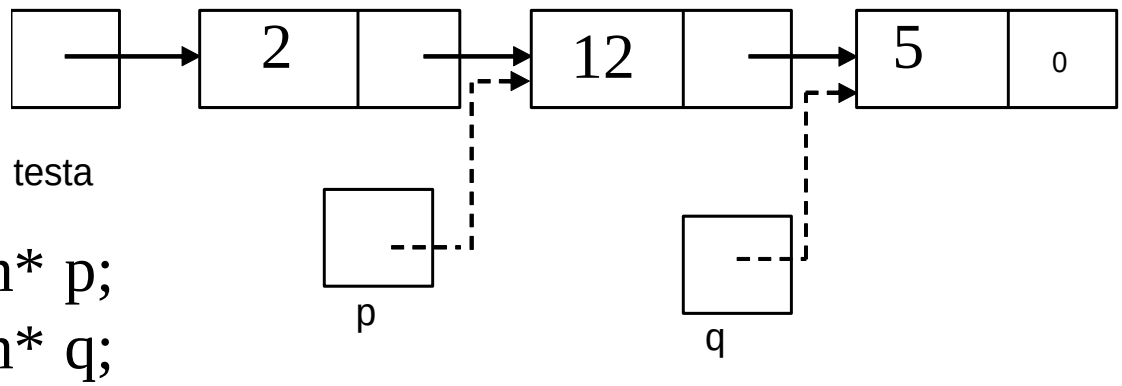
SCORRERE la lista con due puntatori p, q

p punta all'elemento precedente a quello puntato da q

Cercare un elemento nella lista: esempio ultimo elemento oppure elemento con valore 5 oppure .....

con q mi fermo sull'elemento cercato
con p mi fermo sull'elemento precedente

Esempio: elemento con valore 5
casi particolari: lista vuota; 5 non è nella lista



elem* p;
elem* q;

```
for (q=testa; q!= 0 && q->info !=5; q = q->pun)
  p = q;
```

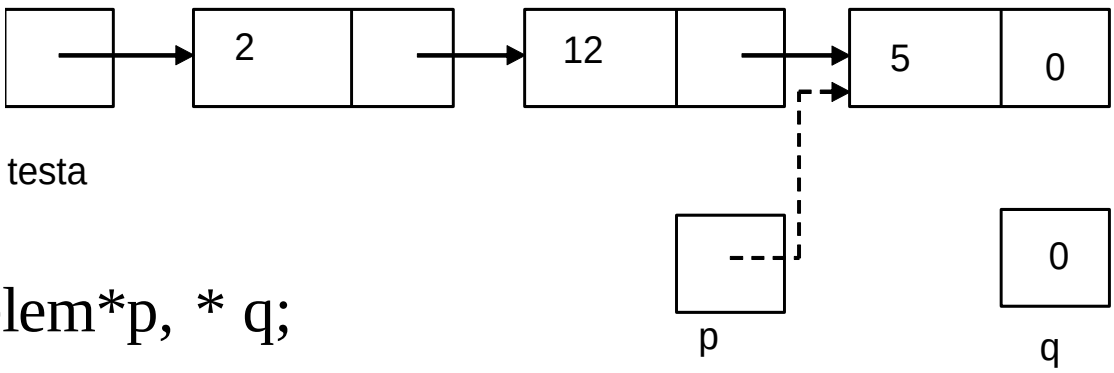
Pensare sempre ai casi particolari:
Lista vuota
se p==NULL fare *p e' un errore a run time

Elemento non trovato
q == NULL alla fine del ciclo

Se inserimento/eliminazione in testa, dobbiamo modificare il puntatore di testa della lista
q == testa alla fine del ciclo

Dobbiamo sempre aggiornare puntatori in modo da avere la lista con / senza il nuovo elemento

Esempio: elemento con valore 5



elem*p, * q;

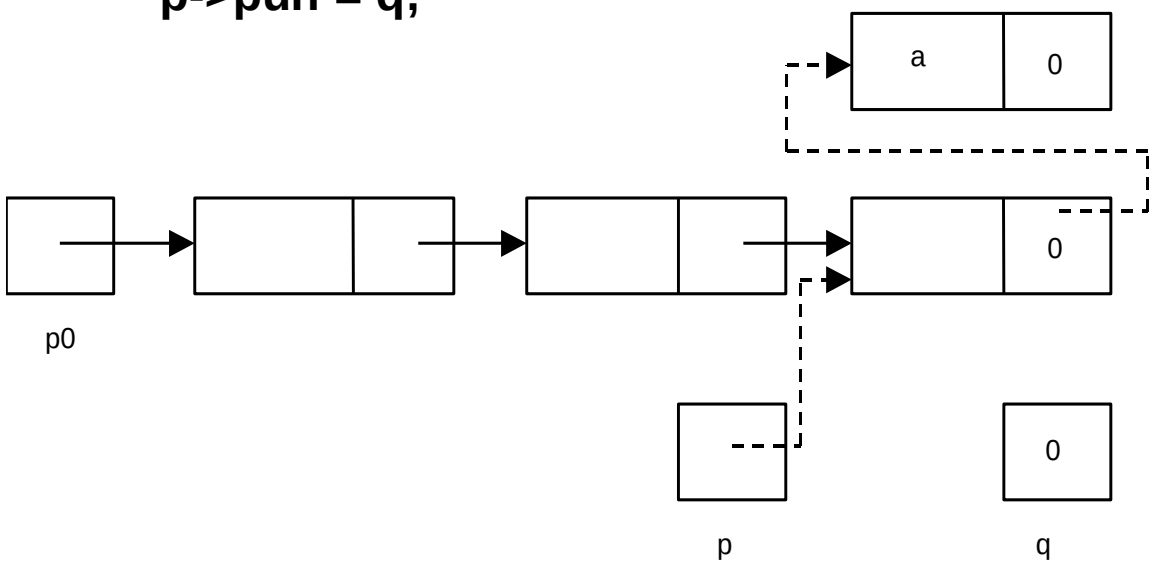
```
for (q=testa; q!= 0 && q->info !=5; q = q->pun)
  p = q;
```

13.2 Liste(VI)

Inserimento in fondo

1. Scandire la lista fino all'ultimo elemento (membro pun = 0)
2. Allocare un nuovo elemento con l'informazione da inserire
3. Collegare l'ultimo elemento al nuovo elemento

```
void insfondo(lista& p0, T a)
{     elem* p;
      elem* q;
      for (q = p0; q != 0; q = q->pun)
            p = q;
      q = new elem;
      q->inf = a;
      q->pun = 0;
      if (p0 == 0)
            p0 = q;
      else
            p->pun = q;
}
```

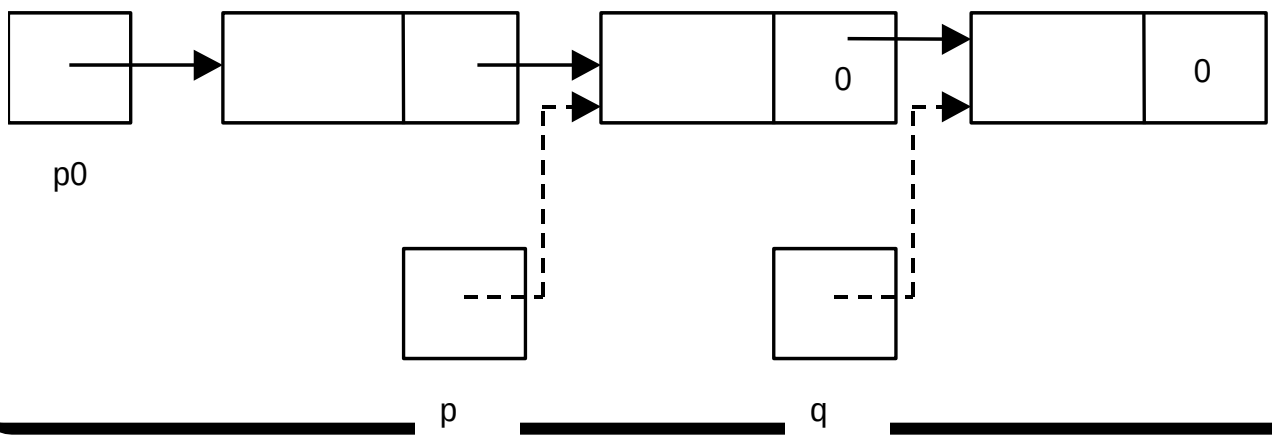


13.2 Liste(VII)

Estrazione dal fondo

ATTENZIONE: necessita di due puntatori per scandire la lista

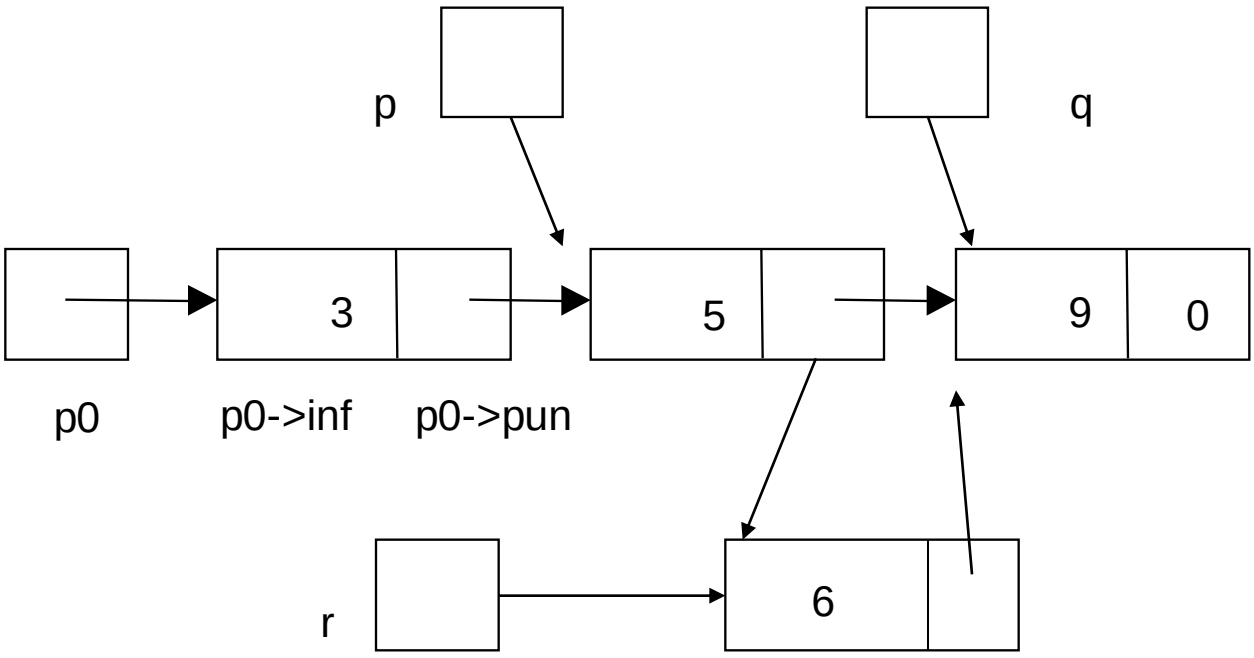
```
bool estfondo(lista& p0, T& a)
{
        elem* p = 0;
        elem* q;
        if (p0 == 0)
           return false;
        for (q = p0; q->pun != 0; q = q->pun)
           p = q;
        a = q->inf;
        // controlla se si estrae il primo elemento
        if (q == p0)
           p0 = 0;
        else
           p->pun = 0;
        delete q;
        return true;
}
```



13.2 Liste(VIII)

Inserimento in una lista ordinata

1. Scandire la lista finchè si incontra un elemento contenente nel campo inf un valore maggiore di quello da inserire oppure fine lista
2. Allocare un nuovo elemento con l'informazione da inserire
3. Inserire il nuovo elemento ….

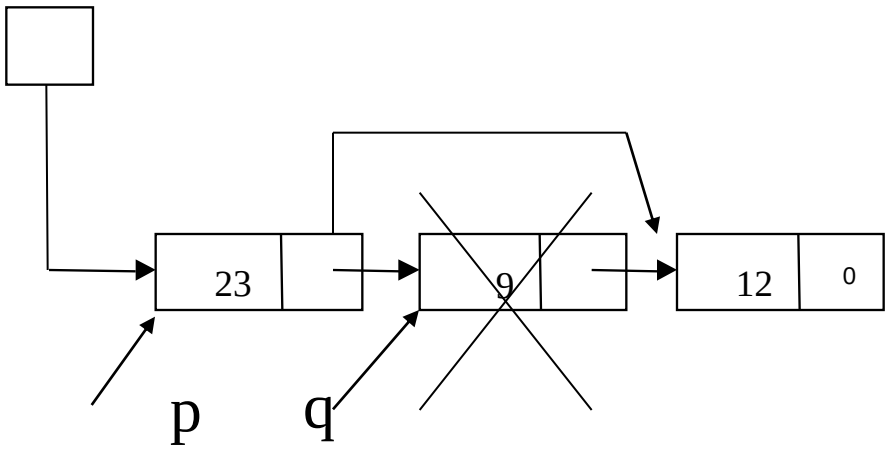



```
void inserimento(lista& p0, T a)
{       elem* p = 0; elem* q; elem* r;
        for (q = p0; q != 0 && q->inf < a; q = q->pun)
                p = q;
        r = new elem;
        r->inf = a; r->pun = q;
        // controlla se si deve inserire in testa
        if (q == p0) p0 = r; else p->pun = r;
}
```

13.2 Liste(IX)

Estrazione di un elemento da una lista

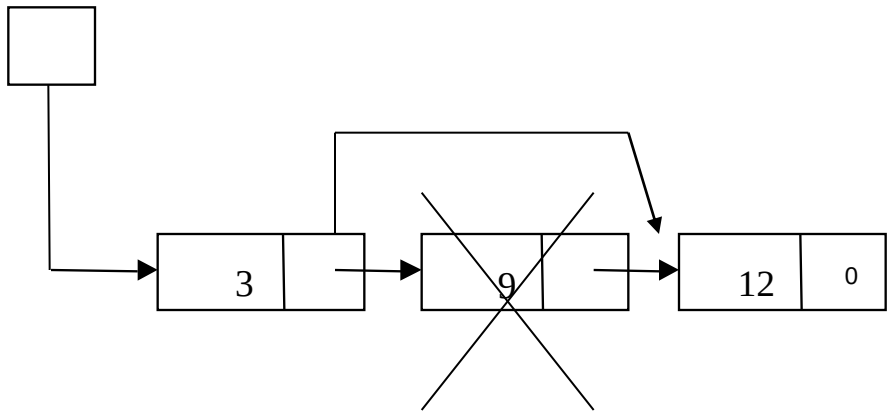
1. Scandire la lista finchè si incontra un elemento contenente l'informazione cercata
2. Se trovato, collegare i due nodi adiacenti
3. Deallocare l'elemento




```
bool estrazione(lista& p0, T a)
{    elem* p = 0; elem* q;
     for (q = p0; q != 0 && q->inf != a; q = q->pun)
          p = q;
     if (q == 0) return false;
     if (q == p0)
          p0 = q->pun;
     else
          p->pun = q->pun;
     delete q;
     return true;
}
```

Estrazione di un elemento da una lista ordinata

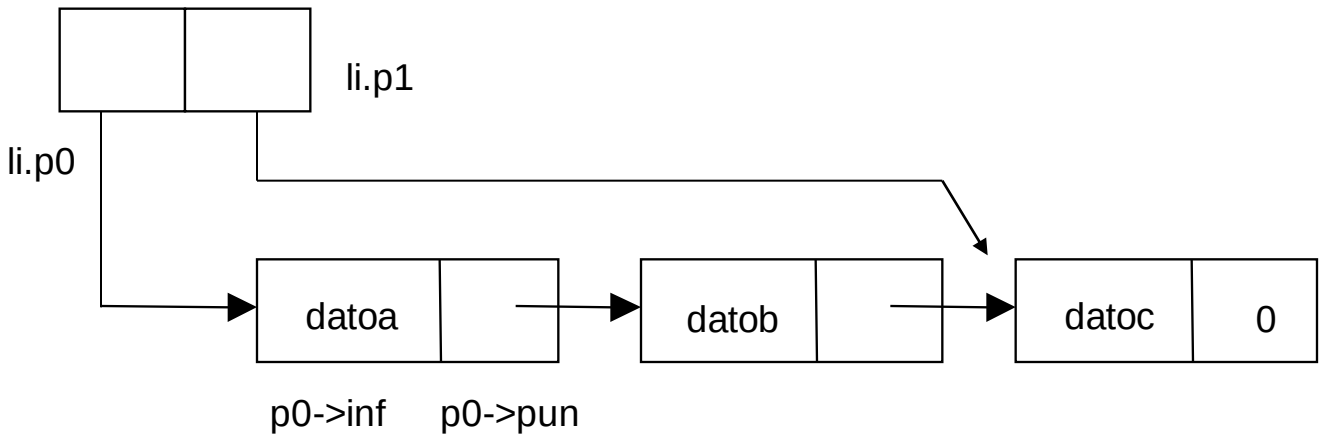
1. Scandire la lista finchè si incontra un elemento contenente l'informazione cercata o maggiore
2. Se trovato, collegare i due nodi adiacenti
3. Deallocare l'elemento




```
bool estrazione_ordinata(lista& p0, T a)
{    elem* p = 0; elem* q;
     for (q = p0; q != 0 && q->inf < a; q = q->pun)
          p = q;
     if ((q == 0)||(q->info>a)) return false;
     if (q == p0)
          p0 = q->pun;
     else
          p->pun = q->pun;
     delete q;
     return true;
}
```

13.2 Liste con puntatore ausiliario (I)

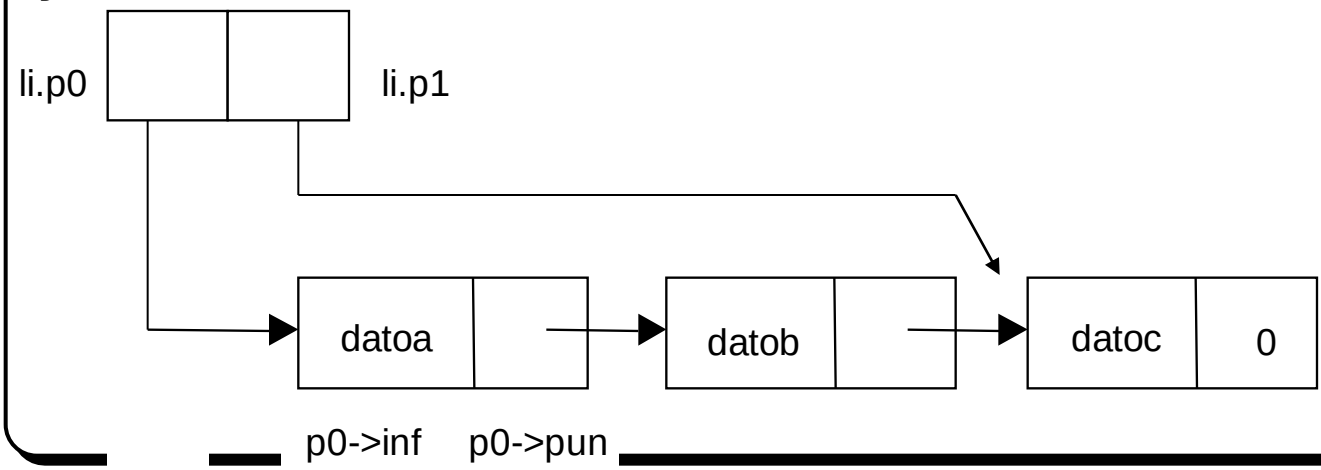
```
struct lista_n
{
    elem* p0;
    elem* p1;
};
```




```
lista_n crealista1(int n)
{   elem* p; lista_n li = {0, 0};
    if (n >= 1)
    {
        p = new elem;
        cin >> p->inf; p->pun = 0;
        li.p0 = p; li.p1 = p;
        for (int i = 2; i <= n; i++)
        {
            p = new elem;
            cin >> p->inf;
            p->pun = li.p0; li.p0 = p;
        }
    }
    return li;
}
```

13.2 Liste con puntatore ausiliario (II)

```
bool esttesta1(lista_n& li, T& a)
{    elem* p = li.p0;
     if (li.p0 == 0)
          return false;
     a = li.p0->inf; li.p0 = li.p0->pun;
     delete p;
     if (li.p0 == 0)
          li.p1 = 0;
     return true;
}

void insfondo1(lista_n& li, T a)
{    elem* p = new elem;
     p->inf = a; p->pun = 0;
     if (li.p0 == 0)
     {
          li.p0 = p; li.p1 = p;
      }
     else
     {
          li.p1->pun = p;
          li.p1 = p;
     }
}
```



13.3 Liste complesse

```
struct nu_elem
{
    T inf;
    elem* prec;
    elem* succ;
};

struct lista_c
{
    nu_elem* p0;
    nu_elem* p1;
};
```