

Liste collegate

STRUTTURE
PUNTATORI
ALLOCAZIONE DINAMICA

Marco Alberti



Dipartimento
di Matematica
e Informatica



Università
degli Studi
di Ferrara

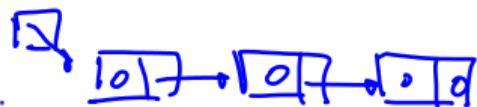
Programmazione e Laboratorio, A.A. 2020-2021

Ultima modifica: 22 dicembre 2020

Attenzione! Questo materiale didattico è per uso personale dello studente ed è coperto da copyright.
Ne sono vietati la riproduzione e il riutilizzo anche parziale, ai sensi e per gli effetti della legge sul diritto d'autore.

Sommario

- 1 Rappresentazione collegata dell'ADT Lista



- 2 Operazioni non strutturali

- 3 Operazioni strutturali fondamentali

INSERIMENTO IN TESTA
ELIMINAZIONE IN TESTA

- 4 Operazioni strutturali derivate

Sommario

1 Rappresentazione collegata dell'ADT Lista

2 Operazioni non strutturali

3 Operazioni strutturali fondamentali

4 Operazioni strutturali derivate

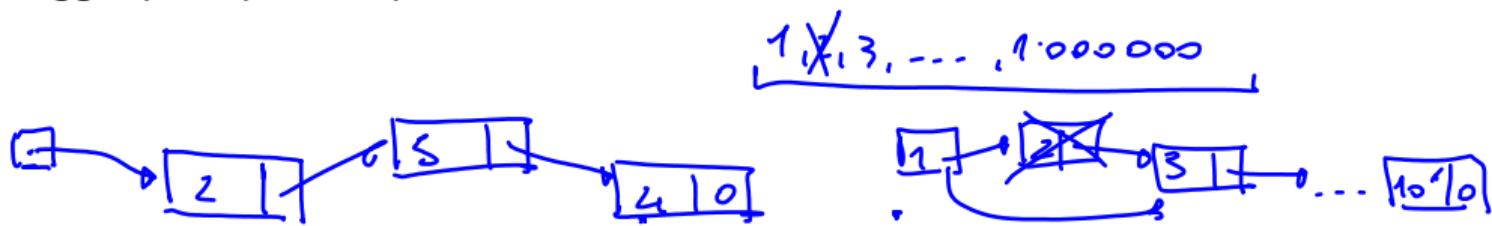
Implementazione collegata

Utilizza l'allocazione dinamica della memoria, allocando (`malloc`) quando si aggiunge un elemento alla lista e liberando (`free`) quando si elimina.

Vantaggi:

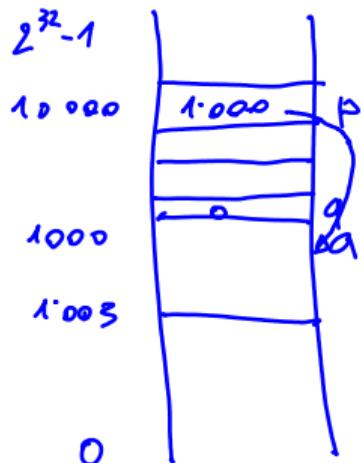
- l'occupazione di memoria di una lista collegata è in ogni momento proporzionale alla sua dimensione logica (a differenza delle liste sequenziali).
- nelle liste sequenziali, l'inserimento e l'eliminazione di un elemento richiedono di spostare a destra o a sinistra, rispettivamente, tutti quelli alla destra.

Svantaggio principale: impossibilità di accesso diretto a un elemento.



Riepilogo notazione grafica

- Un'area di memoria è rappresentata da un rettangolo.
- Una freccia dal centro di un'area di memoria **p** (normalmente un puntatore) al confine di un'area di memoria **a** significa che il valore di **p** è l'indirizzo di **a**.
- Il valore **NULL** è rappresentato dallo **0** all'interno dell'area.



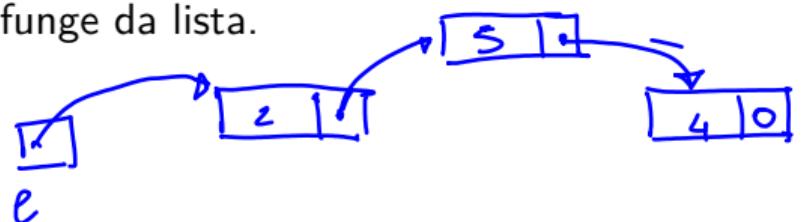
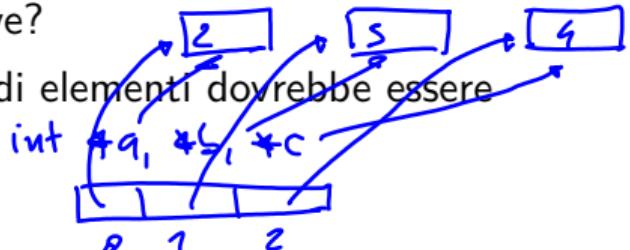
`int a;`
`int *p, int *q = NULL;`

`p = &a;`

.

Rappresentazione collegata delle liste

- La memoria necessaria per ogni elemento si ottiene dinamicamente, con `malloc`.
- Per poter successivamente accedere all'elemento creato, è necessario memorizzare il suo indirizzo (valore di ritorno di `malloc`). Dove?
 - Una variabile per ogni elemento? Il numero di elementi dovrebbe essere fissato a priori; non iterabile.
 - Un array di puntatori? Numero fissato.
 - Soluzione: nell'elemento che lo precede.
 - In questo modo possiamo accedere a tutti gli elementi, partendo dal primo e seguendo i puntatori.
- L'intera lista si può ricostruire dal puntatore al primo elemento, che pertanto funge da lista.



Struttura Nodo

Nella rappresentazione collegata, ogni elemento di tipo **T** della lista è rappresentato tramite un **nodo**, ossia una struttura con due campi:

- **dato** di tipo **T**, che contiene l'elemento
- **next** di tipo puntatore a nodo, che contiene il collegamento all'elemento successivo della lista (**NULL** se l'elemento è l'ultimo)

Esempio

La rappresentazione collegata della lista [2,0,4,3,2] è la seguente:



Definizione del tipo Nodo

Il tipo che rappresenta il nodo è definito *ricorsivamente*.

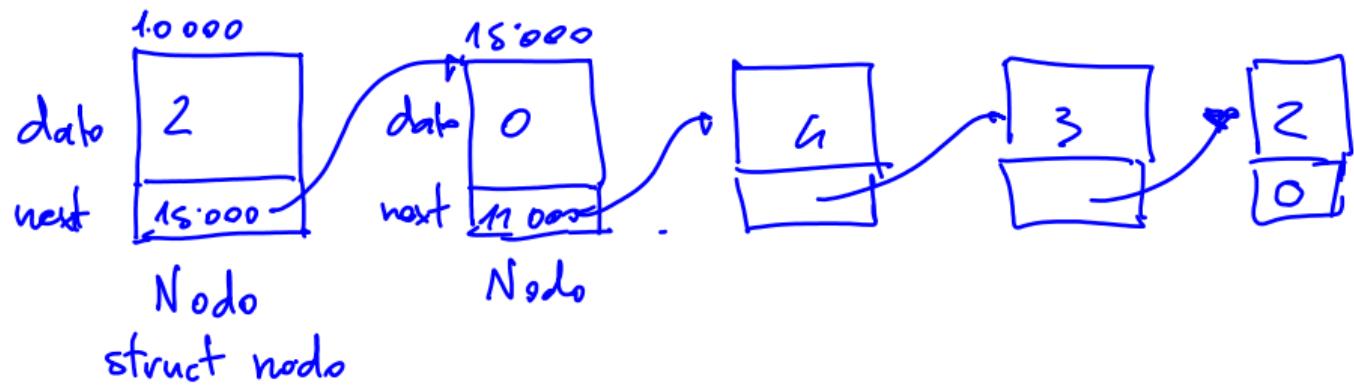
Ad esempio, se il tipo degli elementi della lista è **int**:

```
TIPO ELEMENTO typedef struct nodo {
    int dato;
    struct nodo* next;
} Nodo;
```



Nodo *next; ERRORE
DI SINTASSI
{ **Nodo**,

Il tag **nodo** è necessario per permettere la compilazione (altrimenti il compilatore non saprebbe cos'è il campo **next**).



Definizione del tipo Lista

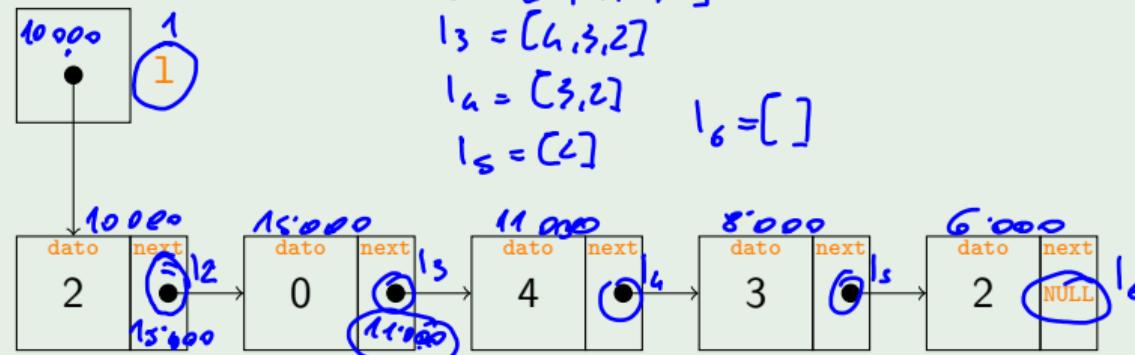
Poiché l'unico "accesso" di una lista è il suo primo elemento, una lista è definita come puntatore al primo nodo:

```
typedef Nodo* Lista;
```

LISTA
10.000 l Lista l

Esempio

Data la definizione **Lista l;**, se **l** rappresenta la lista [2,0,4,3,2], la rappresentazione in memoria è la seguente:



Quante liste collegate sono presenti in questa figura? **6**

Definizioni tipi

230_liste_collegate/tipi.h

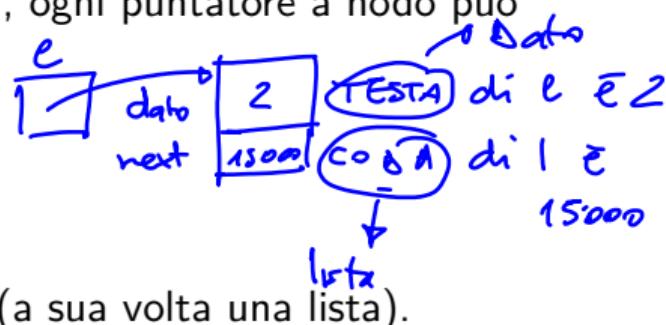
```
1 typedef int Dato; // tipo elemento
2
3 typedef struct nodo {
4     Dato dato;
5     struct nodo* next;
6 } Nodo;
7
8 typedef Nodo* Lista;
```

```
typedef struct {char nome[50],
                int eta;} Dato
typedef struct nodo{
    struct {c
            . . . {dato;
            struct nodo* next;
        } Nodo,
    .
}
```

Terminologia [2,5,3]



- Gli elementi della lista sono rappresentati da nodi.
- Nel contesto dell'implementazione collegata, la lista non è la sequenza dei suoi elementi, ma solo un puntatore a nodo. Viceversa, ogni puntatore a nodo può essere visto come lista.
- Una lista di valore **NULL** si dice **vuota**.
- Se la lista non è **NULL**, cioè punta a un nodo,
 - la **testa** della lista è il campo **dato** del nodo;
 - la **coda** della lista è il campo **next** del nodo (a sua volta una lista).



Interfaccia ADT lista collegata

230_liste_collegate/listaCollegata.h :

```

1  typedef int Dato;
2  typedef struct nodo {
3      Dato dato;
4      struct nodo* next;
5  } Nodo;
6  typedef Nodo* Lista;
7
8  void nuovaLista(Lista* pl);
9  int vuota(Lista l);
10 int piena(Lista l);
11 void insTesta(Lista* pl, int numero);
12 void insOrd(Lista* pl, int numero);
13 int ricerca(Lista l, int numero);
14 void elim1(Lista* pl, int numero);
15 void elimTutti(Lista *pl, int numero);
16 int lunghezza(Lista l);
17 void stampa(Lista l);

```

TIPI DI
DATO

main() {
 Lista l,
 nuovaLista(&l),

}

I prototipi delle funzioni sono uguali a quelli della lista sequenziale.

Sommario

1 Rappresentazione collegata dell'ADT Lista

2 Operazioni non strutturali

3 Operazioni strutturali fondamentali

4 Operazioni strutturali derivate

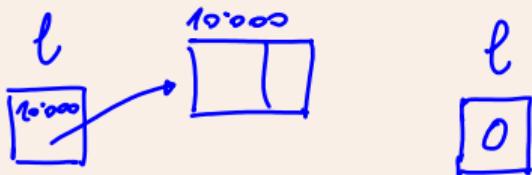
Lista vuota e piena

Una lista collegata è vuota se vale **NULL**, mentre non può mai essere piena.

230_liste_collegate/vuota.c

```

1 #include <stdlib.h>
2 #include "tipi.h"
3
4 int vuota(Lista l) {
5     return l == NULL;
6 }
```



230_liste_collegate/piena.c

```

1 #include "tipi.h"
2
3 int piena(Lista l) {
4     return 0;
5 }
```

ForEach

[2, 0, 4, 5, 8]

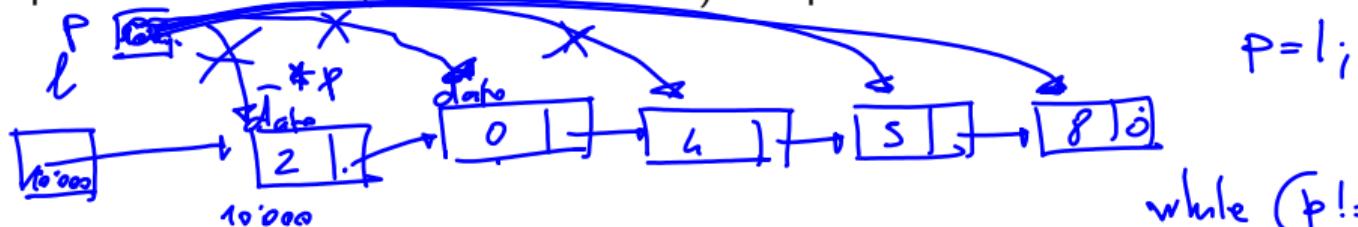
2 0 4 5 8 [0, 0, 0, 0, 0]
~~a [2|0|4|5|8]~~ for (i = 0; i < 5; i++)
~~a[i] = 0..4~~

Il pattern ForEach per su un array **a** si implementa usando un ciclo **for** con un contatore **i** che varia su tutti gli indici di **a**, e ad ogni iterazione si opera sull'elemento **a[i]**.

Con le liste collegate questo non è possibile, perché non si può accedere direttamente all'**i**-esimo elemento. E' necessario seguire la traccia dei puntatori nel campo **next** di tutti gli elementi precedenti.

Per questo si usa un ciclo **while** con un puntatore che

- alla prima iterazione è uguale alla lista
- a ogni iterazione viene sostituito dal campo **next** del nodo a cui punta
- quando vale **NULL** (cioè la lista vuota) comporta l'uscita dal ciclo.



2 0 4 5 8

while (**p** != **NULL**) {
 accenna **p** -> **dato** testa
~~**p** = **p** -> **next**; } 11 / 37~~

ForEach in array e liste collegate

i indice

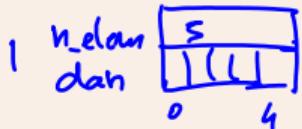
	Lista sequenziale	Lista collegata
Inizializzazione	.i = 0	"p = l ."
Condizione di permanenza	i < dl	p!=NULL (o p)
Elemento a ogni iterazione	a[i]	(*p).dato
Passaggio all'elemento successivo	i++	p = p->next

lista non
vuota
p->dato
p=coda dip

230_liste_collegate/stampa-seq.c

```

1 #include <stdio.h>
2 #include "lista-seq.h"
3 void stampa(Lista l) {
4     int i;
5     for (i = 0; i < (l.n_elementi); i++)
6         printf("%d ", l.dat[i]);
7     printf("\n");
8 }
```



230_liste_collegate/stampa.c

```

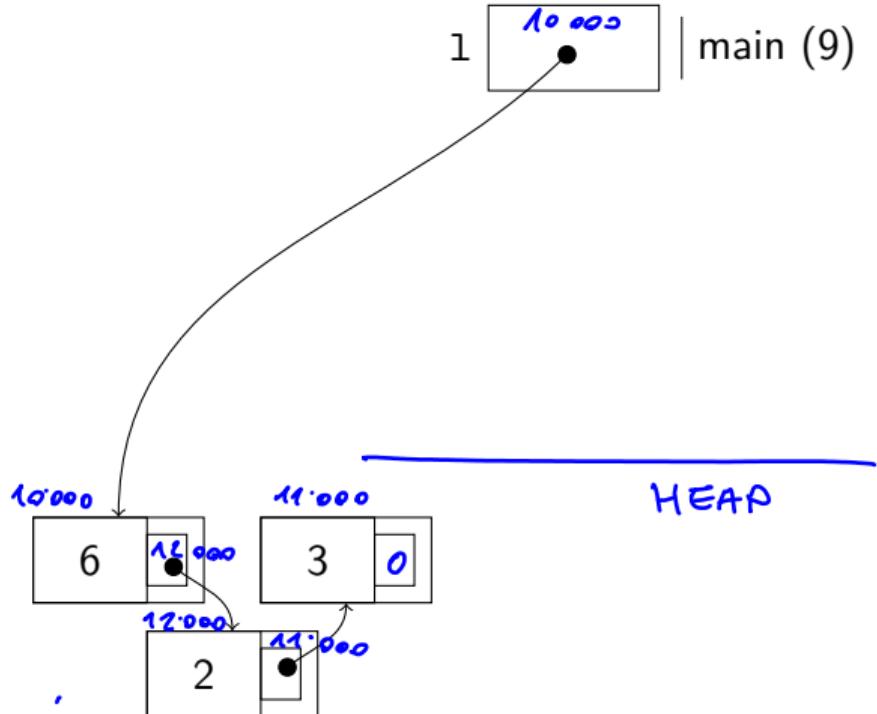
1 #include <stdio.h>
2 #include "tipi.h"
3
4 void stampa(Lista l) {
5     while (l != NULL) {
6         printf("%d ", l->dato);
7         l = l->next;
8     }
9     printf("\n");
10}
```

Annotations on the code:

- Handwritten note: "Nodo + p = l;"
- Handwritten note: "while(p) { ... } p = p->next;"
- Handwritten note: "testa di lista"

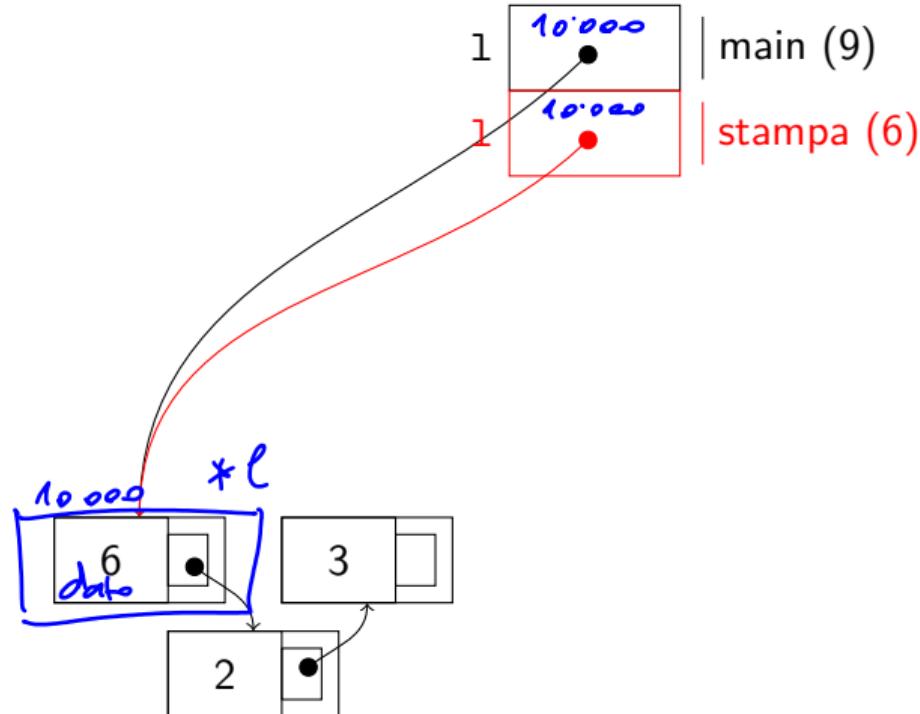
Stampa

```
1 #include <stdio.h>
2 #include "tipi.h"
3 #include "generatoreListe.h"
4 #include "stampa.h"
5
6 int main(void) {
7     Lista l;
8     listaNonOrdinata(&l, 3);
9     stampa(l); 6 2 3
10    return 0;
11 }
```



Stampa

```
1 #include <stdio.h>
2 #include "tipi.h"
3
4 void stampa(Lista l) {
5     while (l) { (*l) dato
6         printf("%d ", l->dato);
7         l = l->next; TESTA di l
8     }
9     printf("\n");
10 }
```

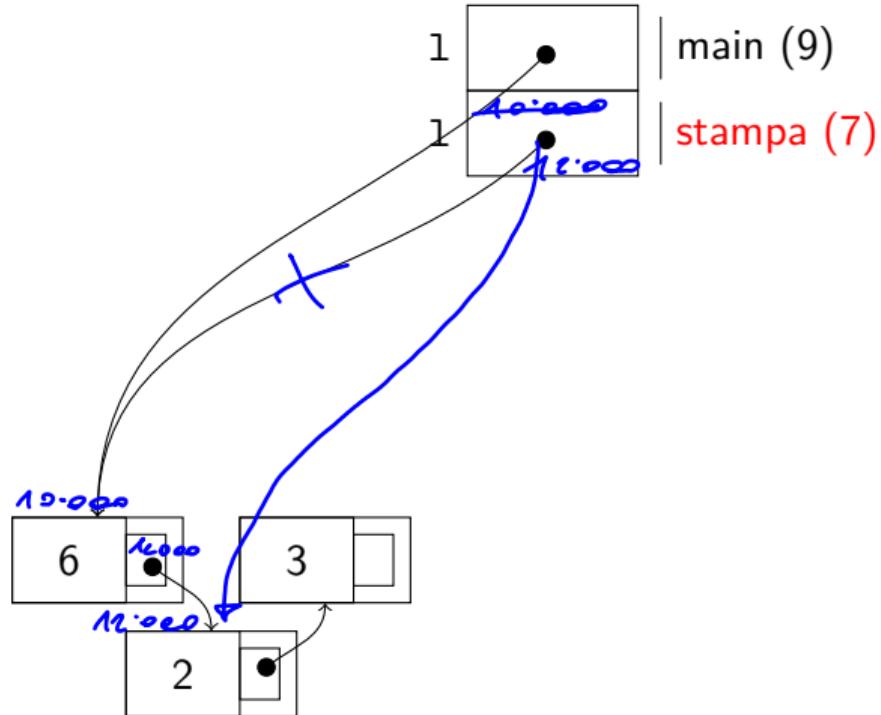


Stampa

```
1 #include <stdio.h>
2 #include "tipi.h"
3
4 void stampa(Lista l) {
5     while (l) {
6         printf("%d ", l->dato);
7         l = l->next;
8     }
9     printf("\n");
10 }
```

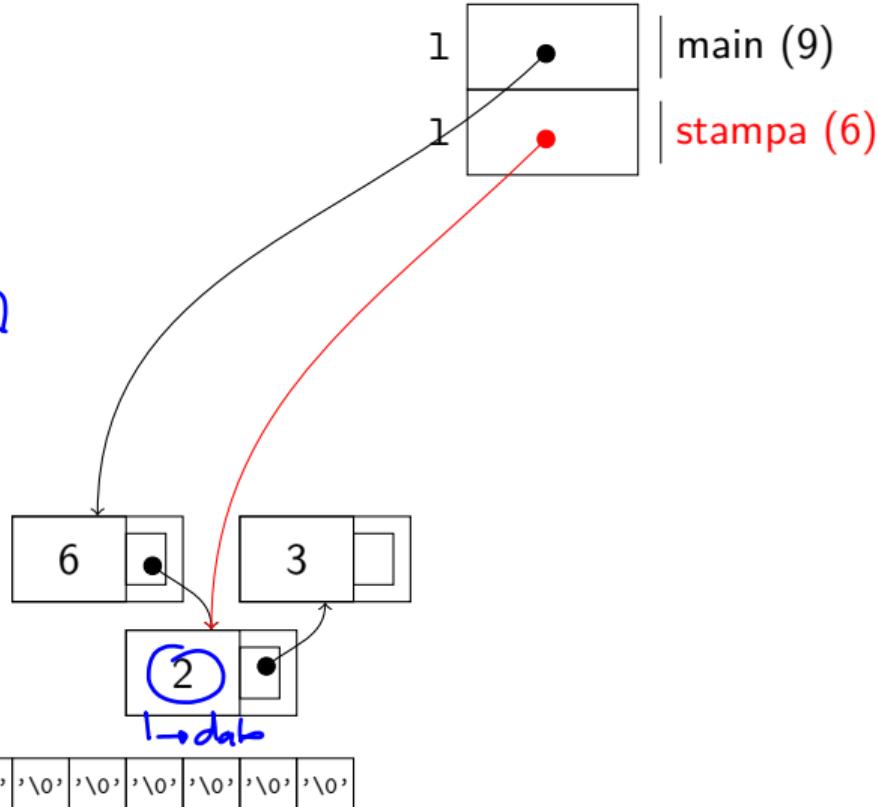
↓

'6'	,	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
stdout																											



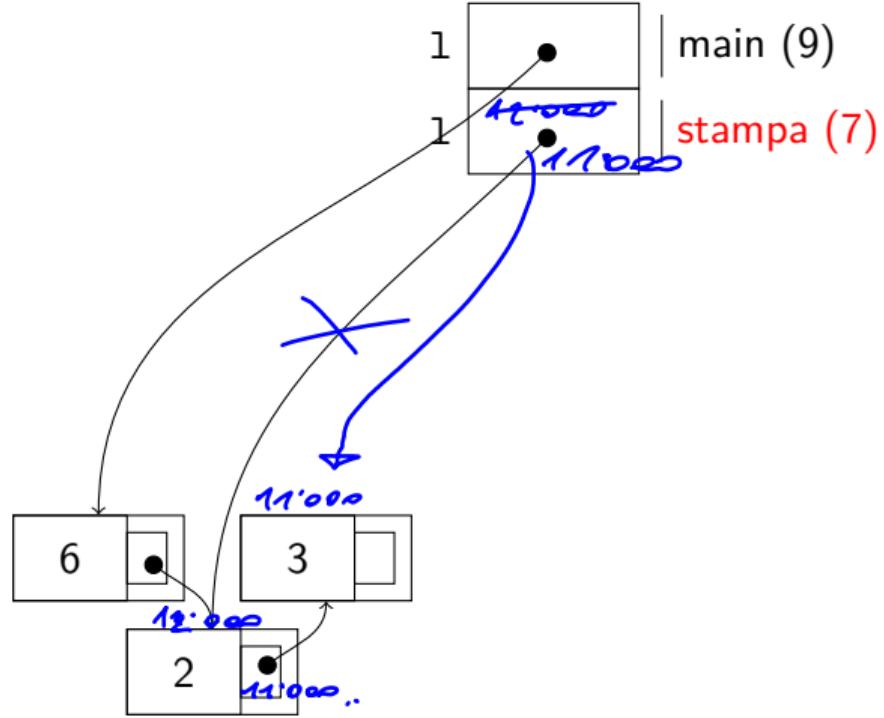
Stampa

```
1 #include <stdio.h>
2 #include "tipi.h"
3
4 void stampa(Lista l) {
5     while (l) {
6         printf("%d ", l->dato);
TUTTA DI [2,3]
7         l = l->next;
8     }
9     printf("\n");
10 }
```



Stampa

```
1 #include <stdio.h>
2 #include "tipi.h"
3
4 void stampa(Lista l) {
5     while (l) {
6         printf("%d ", l->dato);
7         l = l->next; 11'000
8     } coda di l
9     printf("\n");
10 }
```

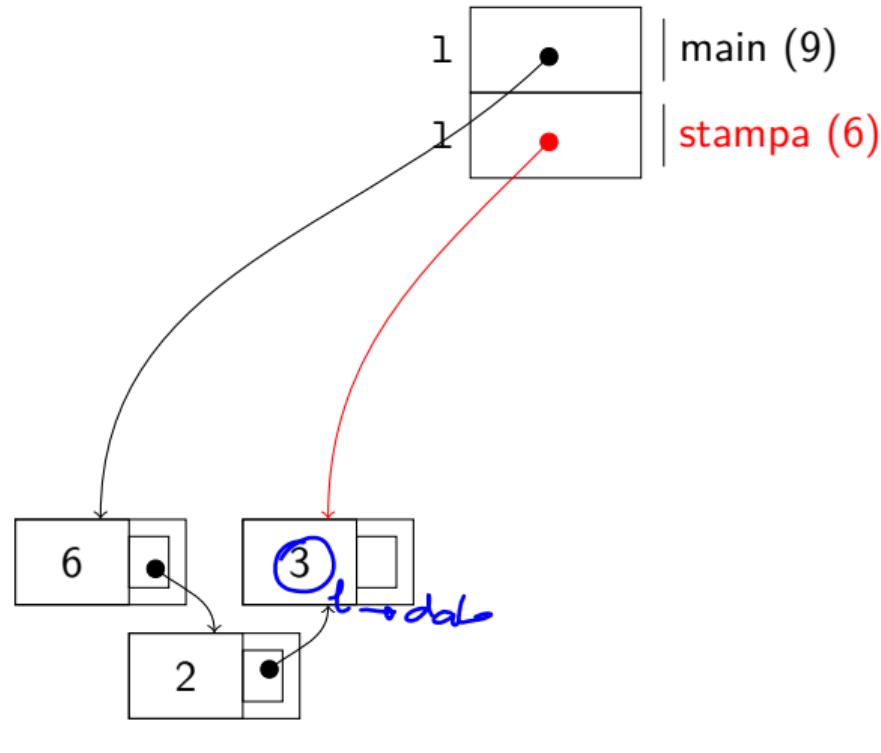


'6'	,	'2'	' '	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
-----	---	------------	------------	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

stdout

Stampa

```
1 #include <stdio.h>
2 #include "tipi.h"
3
4 void stampa(Lista l) {
5     while (l) {
6         printf("%d ", l->dato);
7         l = l->next;
8     }
9     printf("\n");
10 }
```



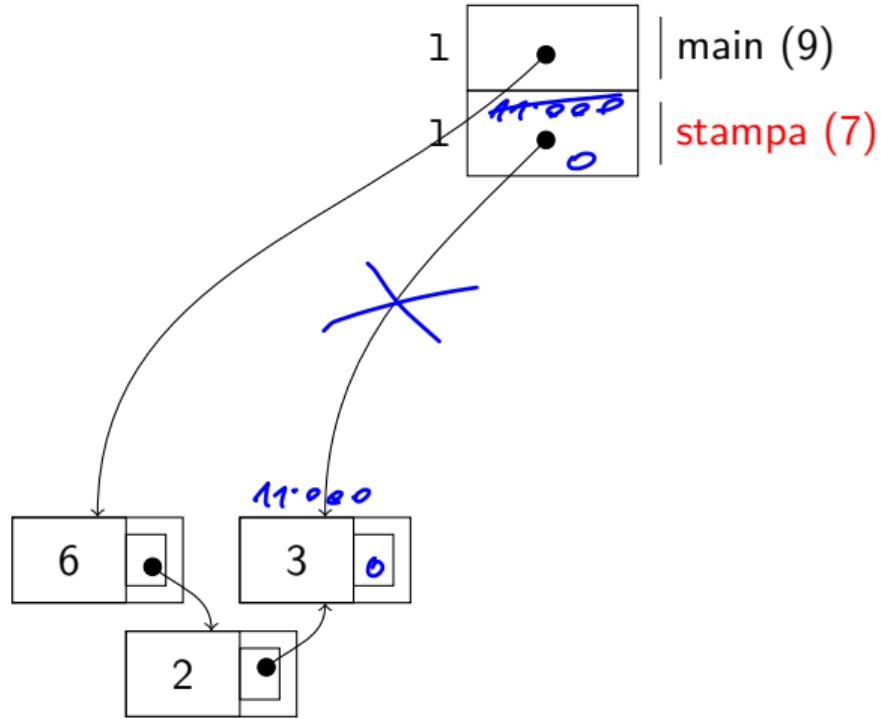
↓

'6'	,	'2'	,	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
-----	---	-----	---	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

stdout

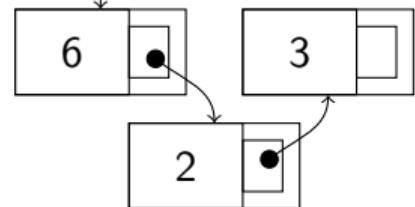
Stampa

```
1 #include <stdio.h>
2 #include "tipi.h"
3
4 void stampa(Lista l) {
5     while (l) {
6         printf("%d ", l->dato);
7         l = l->next;
8     }
9     printf("\n");
10 }
```



Stampa

```
1 #include <stdio.h>
2 #include "tipi.h"
3
4 void stampa(Lista l) {
5     while (l) {
6         printf("%d ", l->dato);
7         l = l->next;
8     }
9     printf("\n");
10 }
```



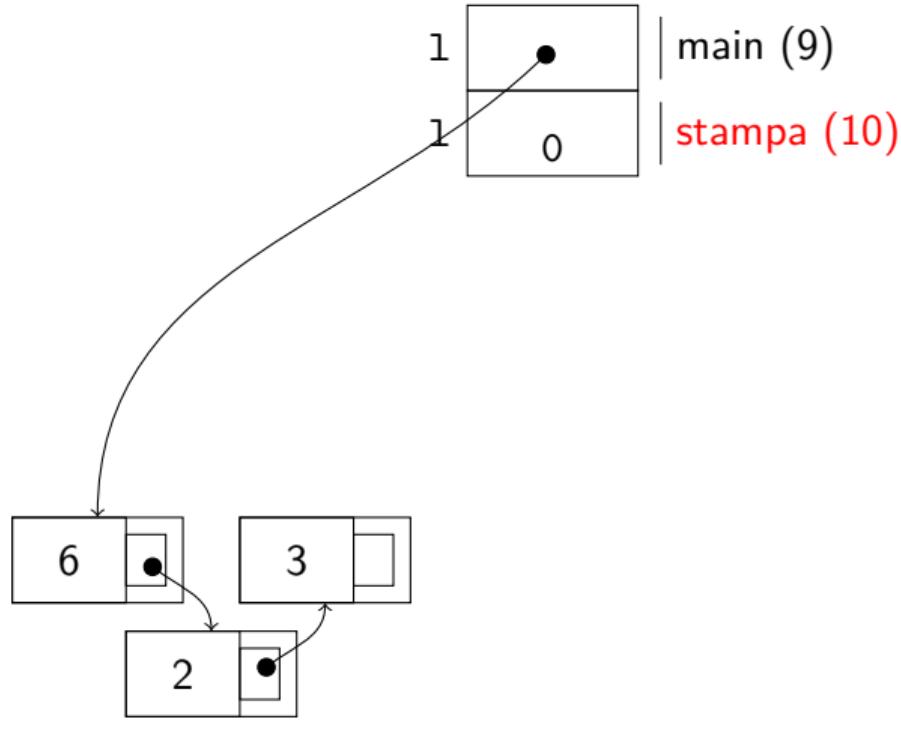
↓

'6'	,	'2'	,	'3'	,	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
-----	---	-----	---	-----	---	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

stdout

Stampa

```
1 #include <stdio.h>
2 #include "tipi.h"
3
4 void stampa(Lista l) {
5     while (l) {
6         printf("%d ", l->dato);
7         l = l->next;
8     }
9     printf("\n");
10 }
```



Esercizio

void azzeri(Lista &L);

Azzeramento lista

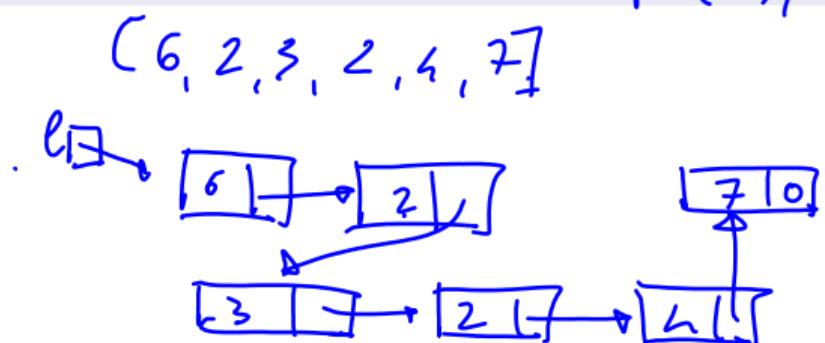
Scrivere una funzione che azzeri tutti gli elementi della lista di interi che riceve come parametro.

Utilizzarla in un programma che

- crei una lista non ordinata di sei elementi;
- azzeri tutti gli elementi della Lista;
- stampi la lista così ottenuta.

generatoreLista.c

stampa(L);



Pattern Reduce (FoldL)

reduce $(+, 0, e)$ $\xrightarrow{[6, 24]} \text{acc } 0 \otimes 88(12)$

Analogamente, il pattern Reduce si ottiene dalla versione per liste sequenziali cambiando solo la modalità di scorrimento della lista e di accesso all'elemento corrente. Questa funzione calcola la somma degli elementi di una lista.

230_liste_collegate/reduce.c

```
1 #include <stdio.h>
2 #include "tipi.h"
3
4 int somma(Lista l) {
5     int s = 0; ACCUMULATORE
6     while (l) {
7         s += l->dato; ELEMENTO CORRENTE (TESTA di e)
8         l = l->next; anche a[i] anziché i++
9     }
10    return s;
11 }
```

Esercizio

numero degli elementi

Lunghezza di una lista

Scrivere una funzione che restituisca la lunghezza (cioè il numero di elementi) di una lista.



reduce (++ , 0 , l)

lunghezza(l)

```
int c = 0,  
while(l) {
```

```
  c++;  
  l = l->next;
```

}

Esercizio

$$l = [2, 6, 4, 0, 5] \quad \text{massimo}(l) \rightarrow 6$$

Massimo di una lista

Scrivere una funzione che restituisca l'elemento massimo di una lista di interi.

VETTORE	LISTA COLLEGATA
elemento contiene	$a[i]$
passaggio	$i++$
elemento	$l \rightarrow \text{dato}$
successivo	$l = l \rightarrow \text{next}$

Esercizio

$$l = [2, 6, 4, 1, 0]$$

List To Array

Scrivere una funzione che, ricevuta come parametro una lista di interi, restituisca l'indirizzo di un vettore, allocato dinamicamente, contenente tutti gli elementi della lista.

int * listToArray (Liste c)

listToArray(l) → 10000



Sommario

1 Rappresentazione collegata dell'ADT Lista

2 Operazioni non strutturali

3 Operazioni strutturali fondamentali

4 Operazioni strutturali derivate

Modifiche strutturali

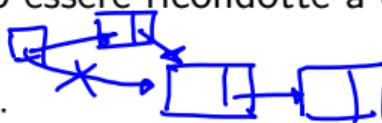
Finora abbiamo visto esempi di operazioni su liste che le lasciano invariate o, al massimo, modificano il campo **dato** dei nodi.

Sono invece dette **strutturali** le modifiche di una lista che cambiano il numero o l'ordine dei nodi. Ad esempio

- non è strutturale la modifica che cambia il valore del primo nodo della lista;
- è strutturale la modifica che elimina il terzo nodo della lista.

Molte modifiche strutturali di una lista possono essere ricondotte a due operazioni elementari:

- Inserimento di un dato in testa a una lista;
- Eliminazione del primo nodo di una lista.



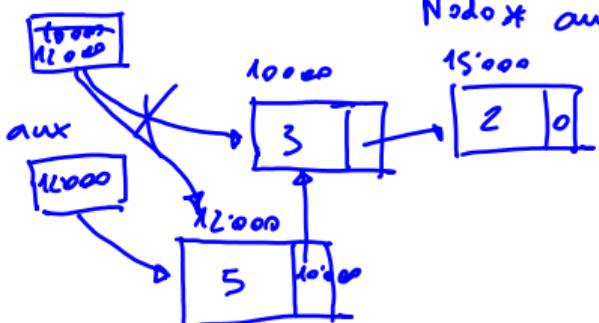
Inserimento in testa s

$$l = [s, z] \\ = [s, s, z]$$

Vogliamo inserire il dato d in testa alla lista l .

Passi:

- ① Creazione di un nodo con `malloc`, assegnando il valore di ritorno a un puntatore `aux`.
- ② Assegnamento di d ad `aux->dato`
- ③ Assegnamento di l ad `aux->next`
- ④ Assegnamento di `aux` a l .



`v ad instTesta (Lista * pl, Dato d)`

$*pl = aux;$

}

`main() {` l `=` `listCreate();`
`instTesta (&l, 5);`

`Nodo * aux = (Nodo*) malloc (sizeof(Nodo));`
`aux->dato = d;`
`aux->next = l;`
`. l = aux;`

Inserimento in testa

Le procedure che operano modifiche strutturali hanno bisogno dell'indirizzo della lista (perché possono dover modificare il valore della lista), quindi il parametro che rappresenta la lista è di tipo **Lista***.

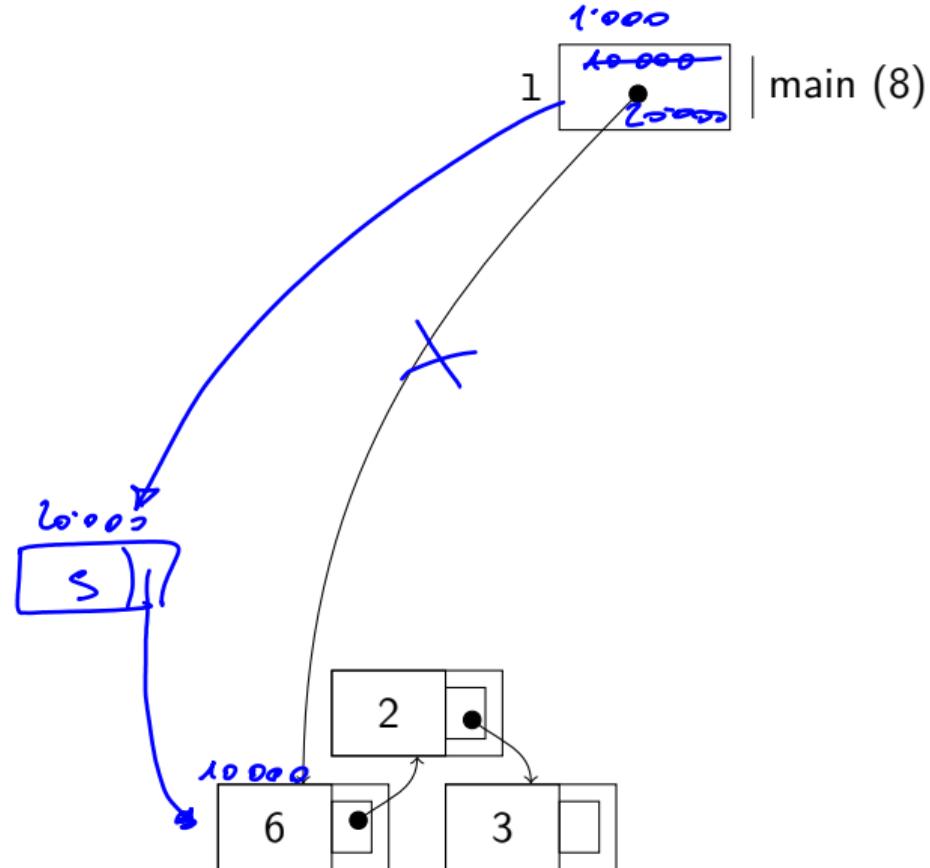
230_liste_collegate/insTesta.c

```
1 #include <stdlib.h>
2 #include "tipi.h"
3
4 void insTesta(Lista* pl, Dato d) {
5     Nodo* aux = (Nodo*)malloc(sizeof(Nodo));
6     if (aux == NULL) exit(100); // allocazione non riuscita
7     aux->dato = d;
8     aux->next = *pl;
9     *pl = aux;
10 }
```

Inserimento in testa

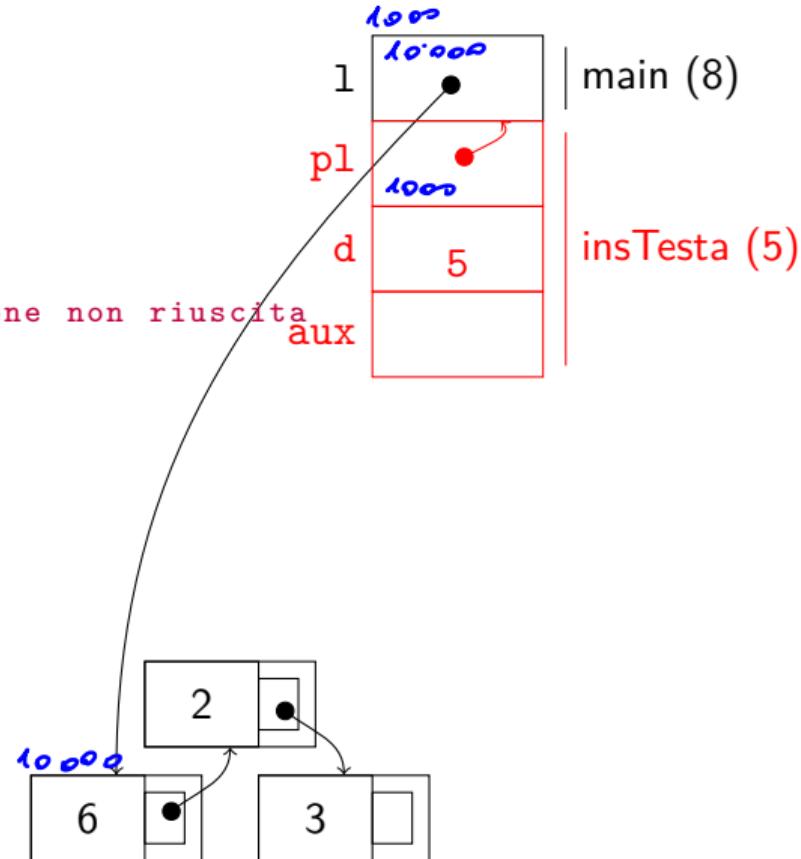
$$l = [6, 2, 3] \rightarrow [5, 6, 2, 3]$$

```
1 #include "tipi.h"
2 #include "generatoreListe.h"
3 #include "insTesta.h"
4
5 int main(void) {
6     Lista l;
7     listaNonOrdinata(&l, 3);
8     insTesta(&l, 5);
9     return 0;
10 }
```



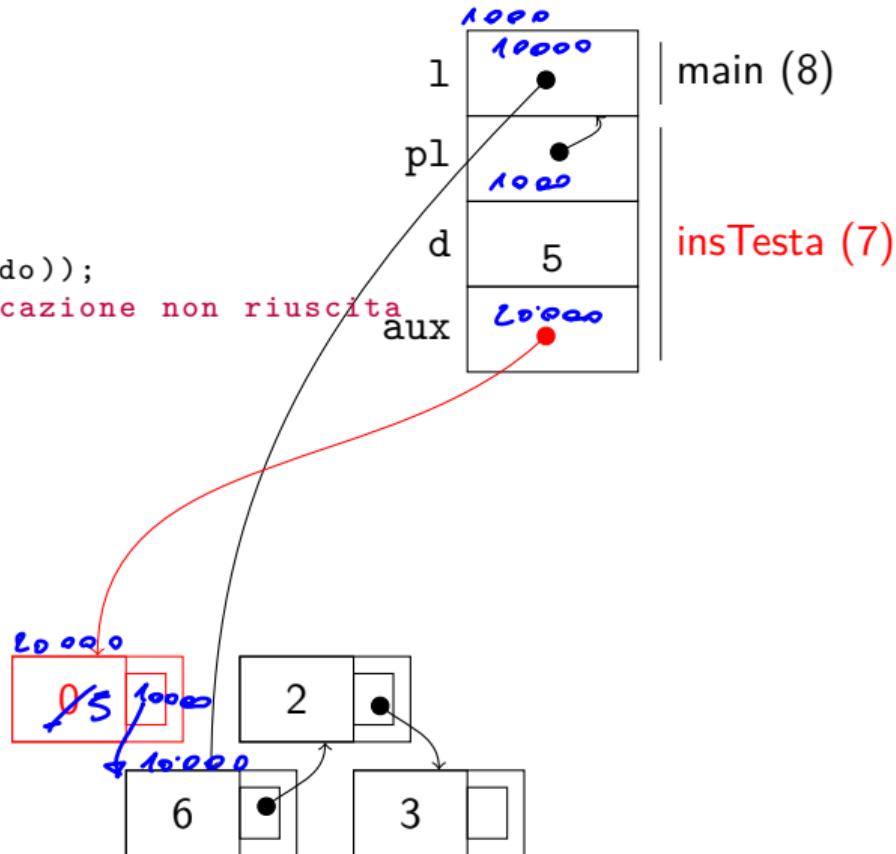
Inserimento in testa

```
1 #include <stdlib.h>
2 #include "tipi.h"
3
4 void insTesta(Lista* 1000 pl, Dato s d) {
5     Nodo* aux = (Nodo*)malloc(sizeof(Nodo));
6     if (aux == NULL) exit(100); // allocazione non riuscita
7     aux->dato = d;
8     aux->next = *pl;
9     *pl = aux;
10 }
```



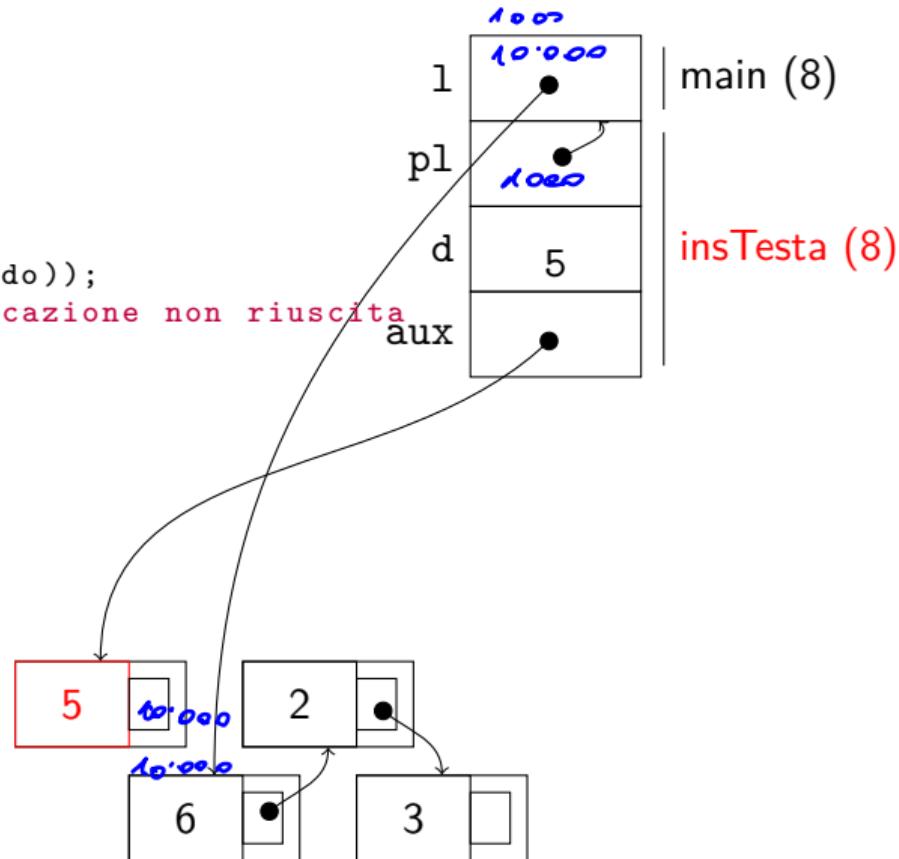
Inserimento in testa

```
1 #include <stdlib.h>
2 #include "tipi.h"
3
4 void insTesta(Lista* pl, Dato d) {
5     Nodo* aux = (Nodo*)malloc(sizeof(Nodo));
6     if (aux == NULL) exit(100); // allocazione non riuscita
7     aux->dato = 5;
8     aux->next = *pl;
9     *pl = aux;
10 }
```



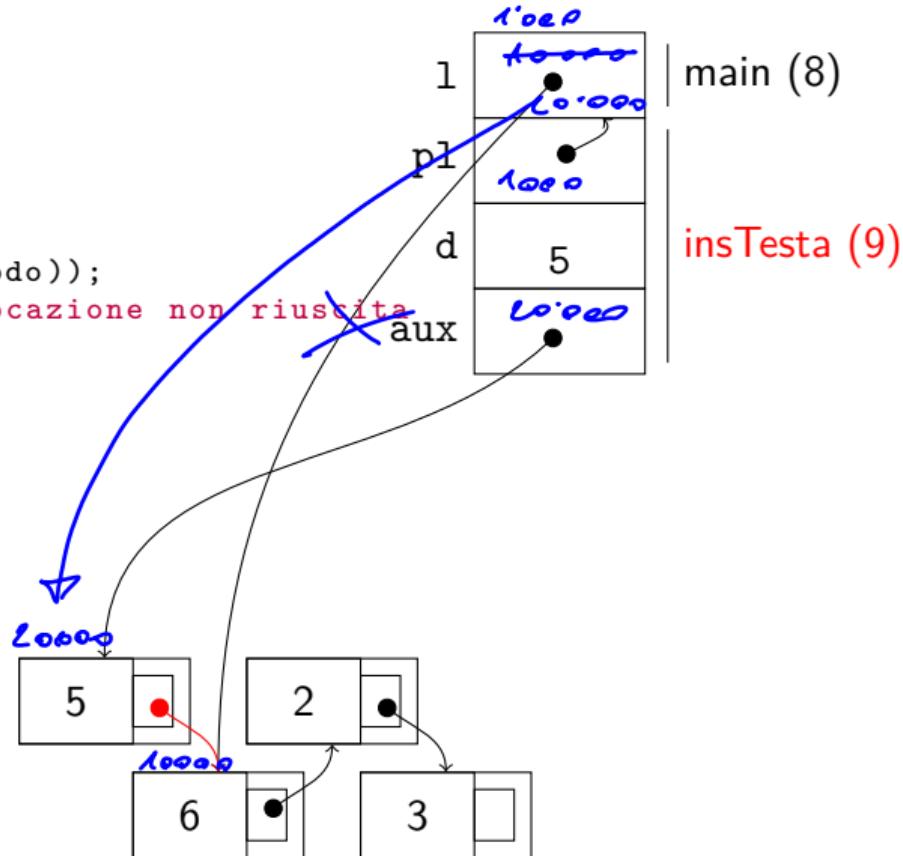
Inserimento in testa

```
1 #include <stdlib.h>
2 #include "tipi.h"
3
4 void insTesta(Lista* pl, Dato d) {
5     Nodo* aux = (Nodo*)malloc(sizeof(Nodo));
6     if (aux == NULL) exit(100); // allocazione non riuscita
7     aux->dato = d;
8     aux->next = *pl
9     *pl = aux;
10 }
```



Inserimento in testa

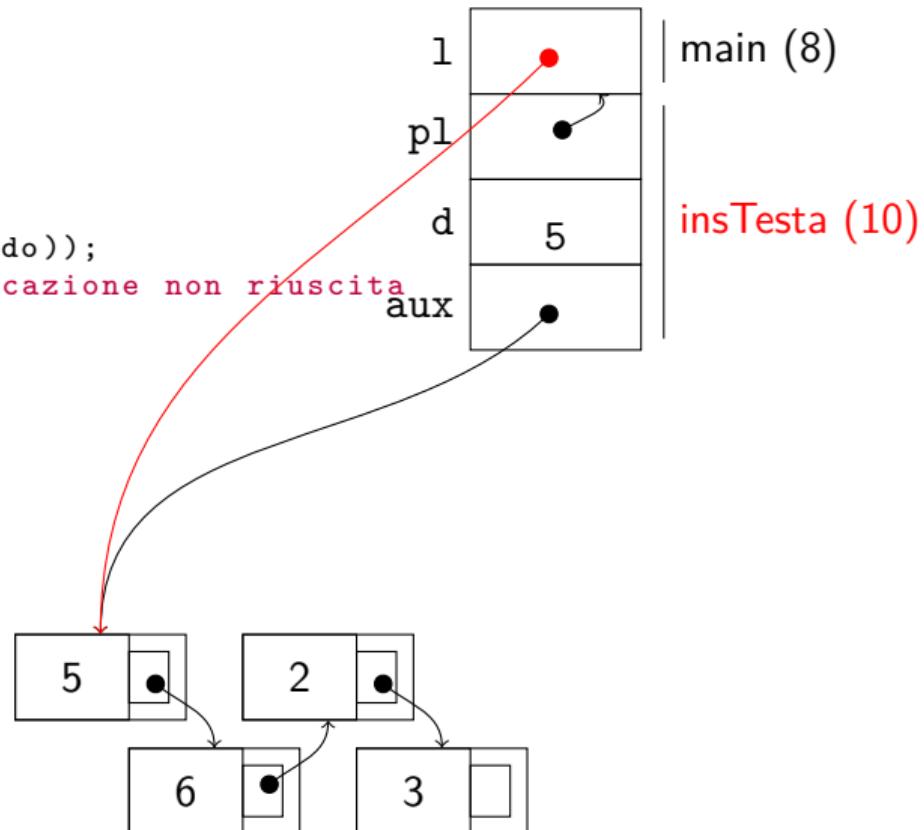
```
1 #include <stdlib.h>
2 #include "tipi.h"
3
4 void insTesta(Lista* pl, Dato d) {
5     Nodo* aux = (Nodo*)malloc(sizeof(Nodo));
6     if (aux == NULL) exit(100); // allocazione non riuscita
7     aux->dato = d;
8     aux->next = *pl;
9     *pl = aux;
10 } e 10.000
```



Inserimento in testa

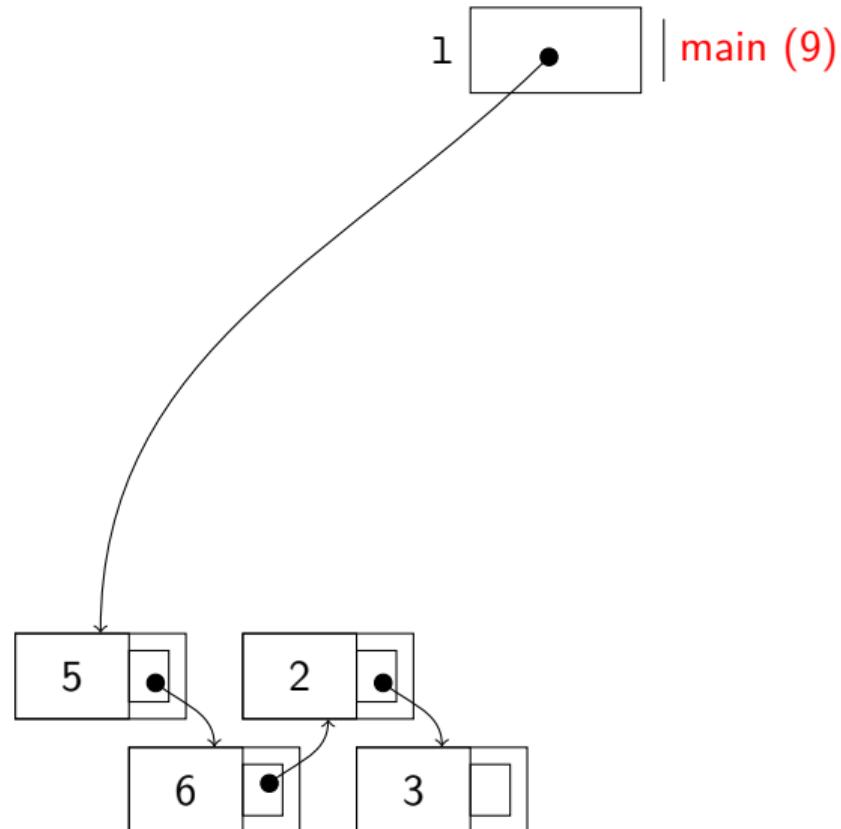
$$l = [5, 6, 2, 3]$$

```
1 #include <stdlib.h>
2 #include "tipi.h"
3
4 void insTesta(Lista* pl, Dato d) {
5     Nodo* aux = (Nodo*)malloc(sizeof(Nodo));
6     if (aux == NULL) exit(100); // allocazione non riuscita
7     aux->dato = d;
8     aux->next = *pl;
9     *pl = aux;
10 }
```



Inserimento in testa

```
1 #include "tipi.h"
2 #include "generatoreListe.h"
3 #include "insTesta.h"
4
5 int main(void) {
6     Lista l;
7     listaNonOrdinata(&l, 3);
8     insTesta(&l, 5);
9     return 0;
10 }
```



Esercizio $e_1 \quad ? \quad 0 \quad 4 \quad 3$

$$e_2 \quad [] \rightarrow [2] \rightarrow [0, 2] \\ \rightarrow [6, 0, 1, 0] \rightarrow [3, 4, 0, 2]$$

Reverse

Scrivere una funzione di prototipo `void reverse(Lista *pl1, Lista *pl2)` che faccia sì che ~~*pl2 sia la lista inversa di pl1~~, cioè contenga gli stessi elementi ma in ordine inverso. Ad esempio, l'inversa di $[2,0,4,3]$ è $[3,4,0,2]$.

E' sufficiente:

- inizializzare `*pl2; []`
- inserire in testa a `*pl2` gli elementi di `*pl1`, dal primo all'ultimo.

`ForEach(iusTesta(pl2, e1->data), e1)`

Utilizzare la funzione `reverse` in un programma che

- crei una lista non ordinata di sei elementi;
- calcoli la sua inversa;
- le stampi entrambe.

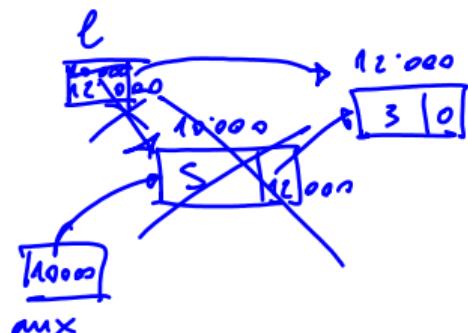
Eliminazione in testa

$$l = [5, 3] \quad l \rightarrow [3]$$

Vogliamo eliminare il primo elemento della lista l , cioè far sì che l assuma il valore della sua coda.

Ovviamente ha senso solo se la lista non è vuota.

Non bisogna dimenticare di liberare la memoria occupata dal primo elemento, che non sarà più raggiungibile; per questo il valore di l deve essere salvato prima di cambiarlo.



$$\text{Node} * \text{aux} = l,$$

$l = \text{e} \rightarrow \text{next},$
 $\text{free}(\text{aux}),$

void elimTesta(Lista *pl)
 {
 Node *aux = *pl;
 *pl = (*pl) \rightarrow next;
 free(aux);
 }

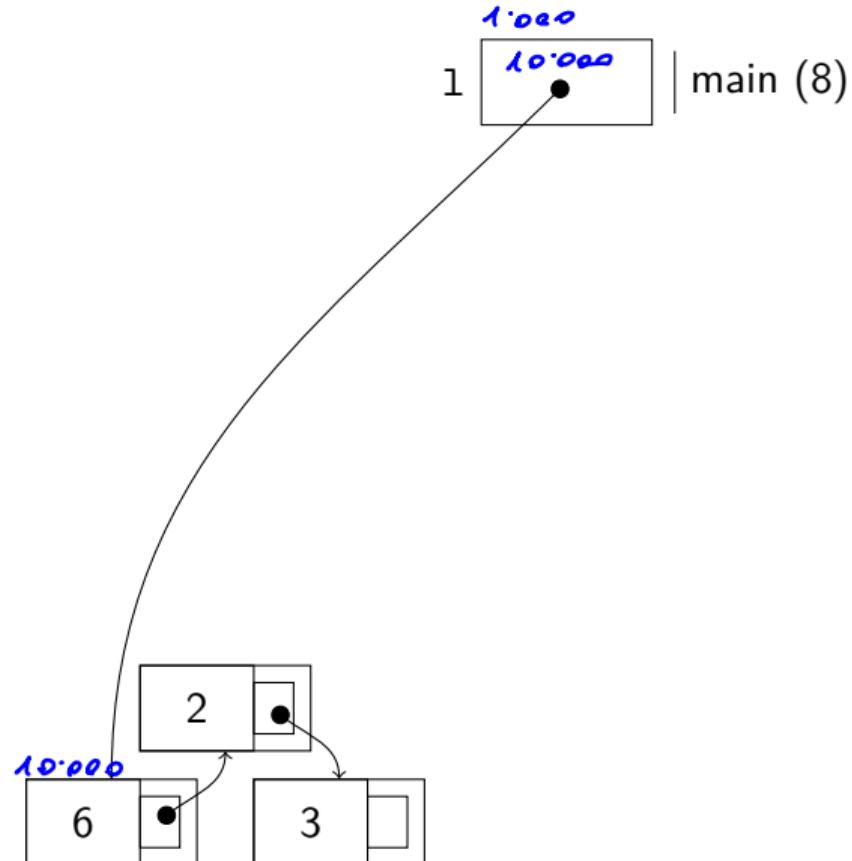
Eliminazione in testa

230_liste_collegate/elimTesta.c

```
1 #include <malloc.h>
2 #include "tipi.h"
3
4 void elimTesta(Lista* pl) {
5     Nodo* aux = *pl;
6     *pl = (*pl)->next;
7     free(aux);
8 }
```

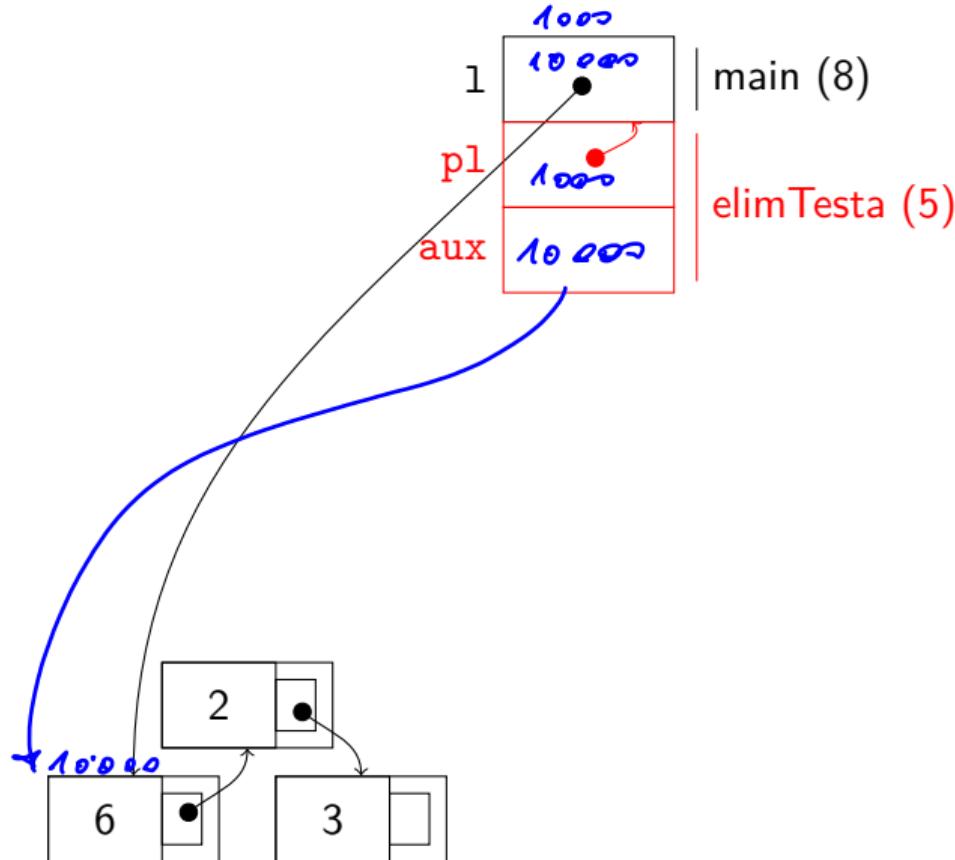
Eliminazione in testa

```
1 #include "tipi.h"
2 #include "generatoreListe.h"
3 #include "elimTesta.h"
4
5 int main(void) {
6     Lista l;
7     listaNonOrdinata(&l, 3);
8     elimTesta(&l);
9     return 0;
10 }
```



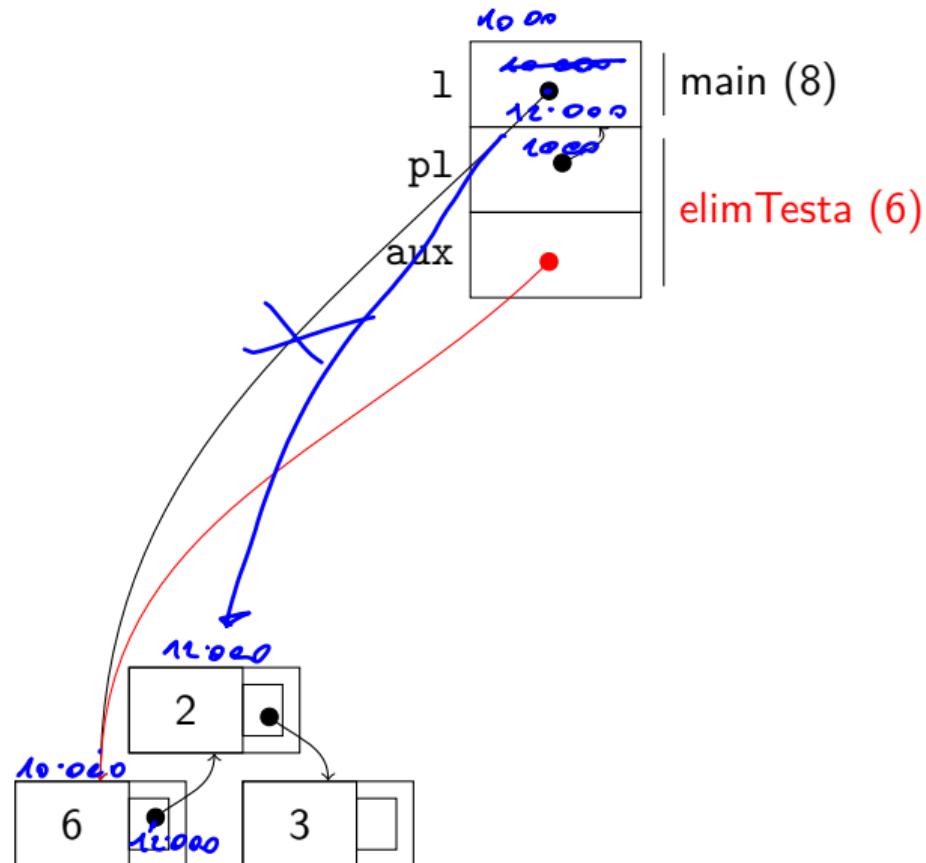
Eliminazione in testa

```
1 #include <malloc.h>
2 #include "tipi.h"
3
4 void elimTesta(Lista* pl) {
5     Nodo* aux = *pl;
6     *pl = (*pl)->next;
7     free(aux);
8 }
```



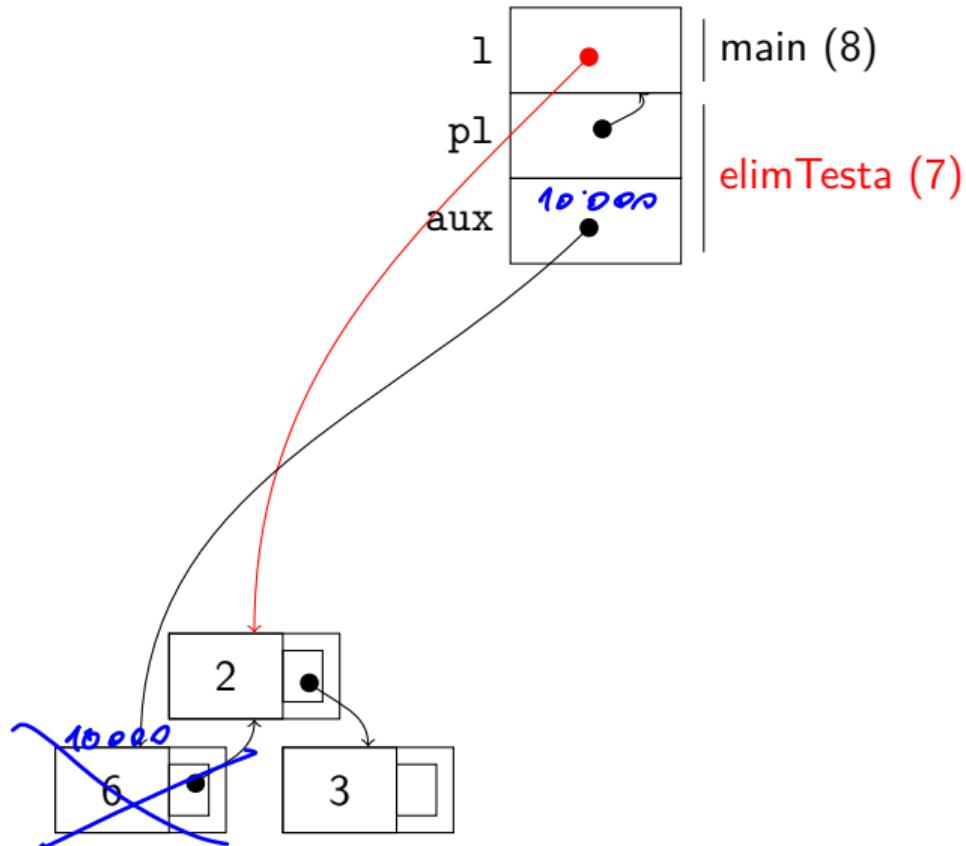
Eliminazione in testa

```
1 #include <malloc.h>
2 #include "tipi.h"
3
4 void elimTesta(Lista* pl) {
5     Nodo* aux = *pl;
6     *pl = (*pl)->next;
7     free(aux);
8 }
```



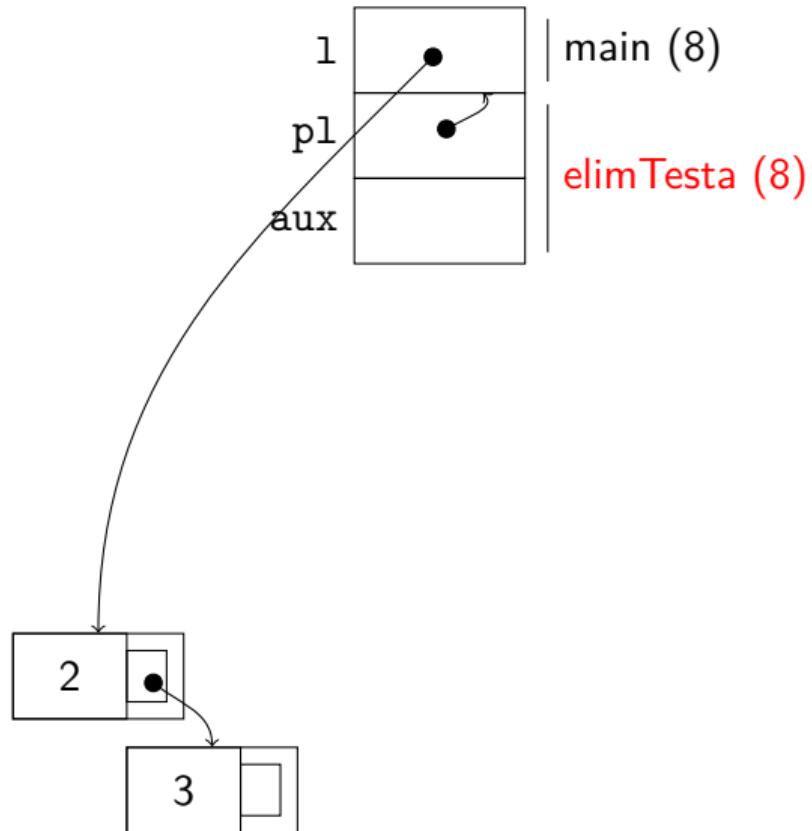
Eliminazione in testa

```
1 #include <malloc.h>
2 #include "tipi.h"
3
4 void elimTesta(Lista* pl) {
5     Nodo* aux = *pl;
6     *pl = (*pl)->next;
7     free(aux);
8 }
```



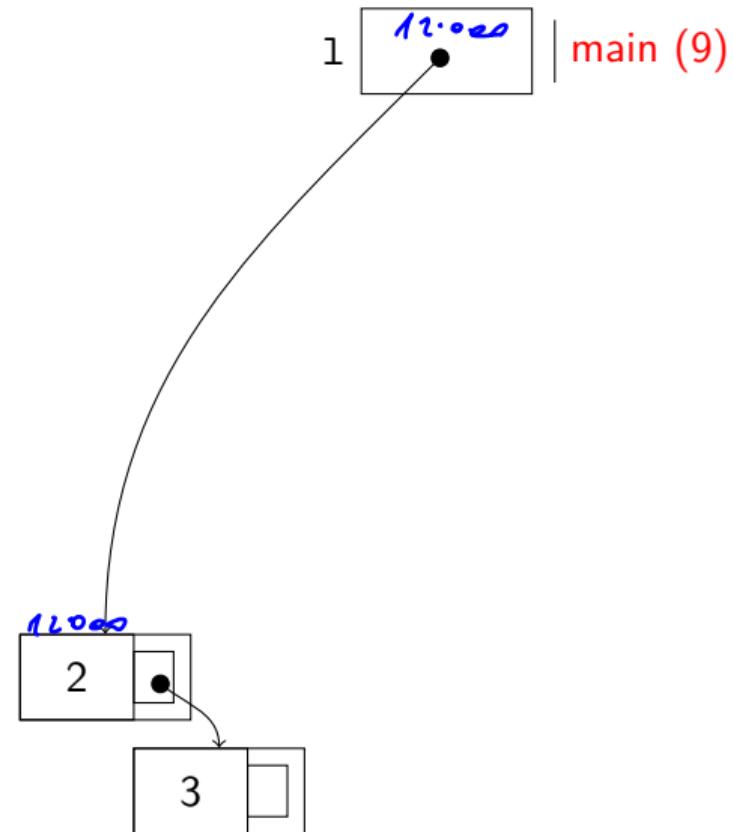
Eliminazione in testa

```
1 #include <malloc.h>
2 #include "tipi.h"
3
4 void elimTesta(Lista* pl) {
5     Nodo* aux = *pl;
6     *pl = (*pl)->next;
7     free(aux);
8 }
```



Eliminazione in testa

```
1 #include "tipi.h"
2 #include "generatoreListe.h"
3 #include "elimTesta.h"
4
5 int main(void) {
6     Lista l;
7     listaNonOrdinata(&l, 3);
8     elimTesta(&l);
9     return 0;
10 }
```



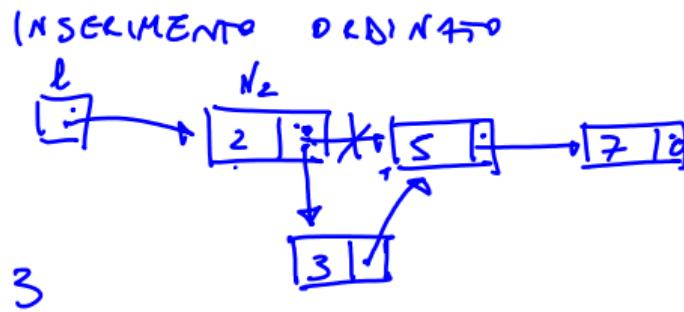
Sommario

1 Rappresentazione collegata dell'ADT Lista

2 Operazioni non strutturali

3 Operazioni strutturali fondamentali

4 Operazioni strutturali derivate



Inizializzazione di una lista

Per inizializzare una lista, la si pone uguale **NULL**.

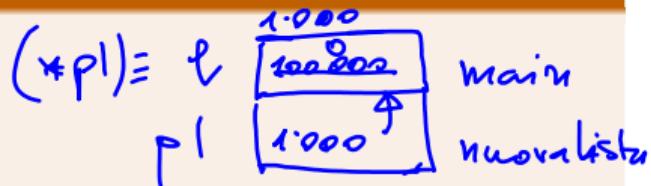
230_liste_collegate/nuovaLista.c

```

1 #include <stdlib.h>
2 #include "tipi.h"
3
4 void nuovaLista(Lista* pl) {
5   *pl = NULL;
6 }
```

```

main () {
  Lista l;
  l=NULL; nuovaLista(&l);
}
}
```



Naturalmente, nelle funzioni che implementano modifiche a liste il parametro che rappresenta la lista è di tipo **Listastar**.

Liste parziali



insTesta(Lista *p,
Dato d)



Ricordiamo che il campo **next** di ogni nodo può essere visto come una lista (eventualmente nulla).

Molte operazioni (per esempio l'inserimento di un elemento in ordine, o l'eliminazione degli elementi con un determinato dato) possono essere viste come l'applicazione delle operazioni elementari su una sotto-lista, in due fasi:

- ① ricerca della lista su cui agire;
- ② applicazione di una delle operazioni elementari alla lista trovata.

Che cosa ricercare dipende dall'operazione da svolgere: se vogliamo eliminare il numero **2** da una lista cercheremo la lista la cui testa è uguale a **2**; se vogliamo operare un inserimento ordinato di **5** in una lista ordinata in modo crescente cercheremo la prima lista la cui testa è maggiore o uguale a **5**.

Poiché ~~sulla~~ sulla lista trovata dovranno essere eseguite modifiche strutturali, la ricerca deve restituire l'indirizzo della lista.

Ricerca lista

void inserisci(Lista* pl, Dato d) {

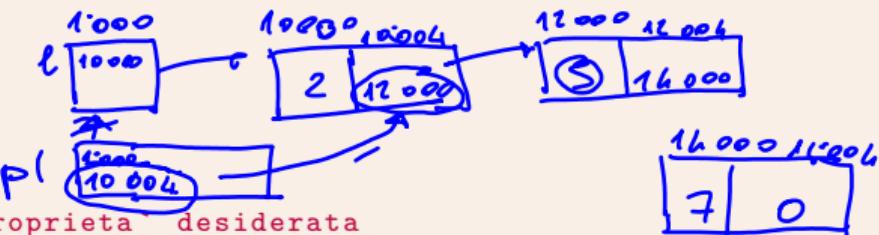
Restituisce l'indirizzo della prima lista la cui testa ha la proprietà desiderata, o della lista vuota se tale nodo non c'è.



230_liste_collegate/ricerca.c

```

1 #include "tipi.h"
2
3 Lista* ricerca(Lista* pl, Dato d) {
4     // finche' la lista ha elementi
5     while (*pl) {
6         // se l'elemento corrente ha la proprietà desiderata
7         if ((*pl)->dato == d)
8             // esco
9             break;
10        // altrimenti passo all'elemento successivo
11        pl = &(*pl)->next;
12    }
13    // qui l'e' l'indirizzo della lista desiderata, o di quella vuota
14    return pl; 10004
15 }
```



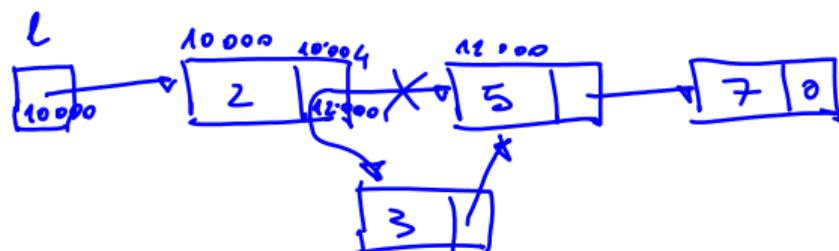
elimTesta (10004)

Inserimento ordinato

Supponendo che gli elementi di una lista l siano in un determinato ordine (ad esempio crescente, se l è una lista di interi), l'**inserimento ordinato** di un dato d in l produce una lista, ordinata allo stesso modo, che contiene un nuovo nodo con dato d .

Passi:

- ① Ricerca della prima lista l_1 la cui testa non preceda d ;
- ② Inserimento di d in testa a l_1 .



l
 $pl = 10\cdot004$

$\text{insTesta}(pl, d)$

Inserimento ordinato

230_liste_collegate/insOrd.c

```

1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }

```

3

*condizione sulla testa di *pl che individua la lista per l'inserimento in testa*

10004 3

Inserimento Ordinato

1 | main (6)

```
1 #include "tipi.h"
2 #include "insOrd.h"
3
4 int main(void) {
5     Lista l;
6     nuovaLista(&l);
7     insOrd(&l, 3);
8     insOrd(&l, 1);
9     insOrd(&l, 4);
10    insOrd(&l, 2);
11    return 0;
12 }
```

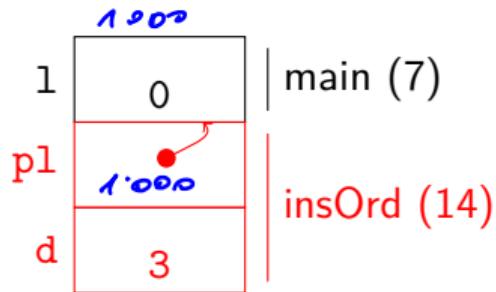
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insOrd.h"
3
4 int main(void) {
5     Lista l;
6     nuovaLista(&l);
7     insOrd(&l, 3);
8     insOrd(&l, 1);
9     insOrd(&l, 4);
10    insOrd(&l, 2);
11    return 0;
12 }
```



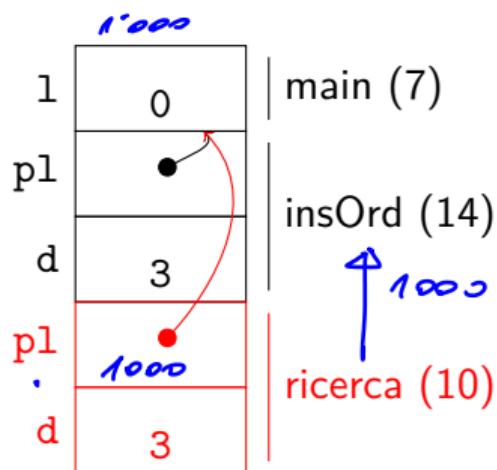
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d); 3
16 }
```



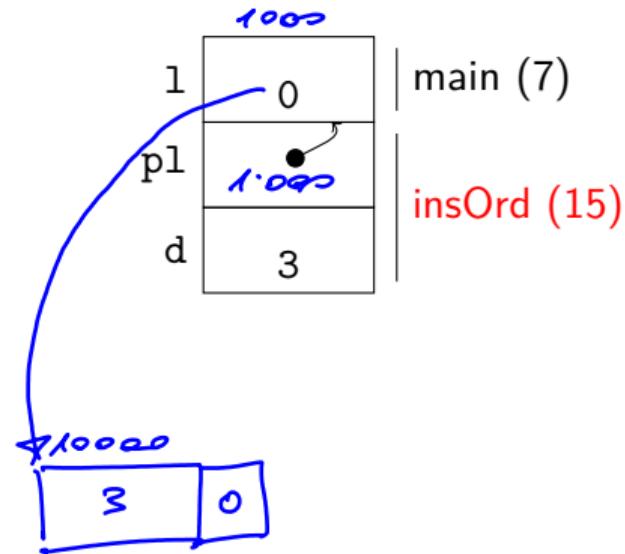
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



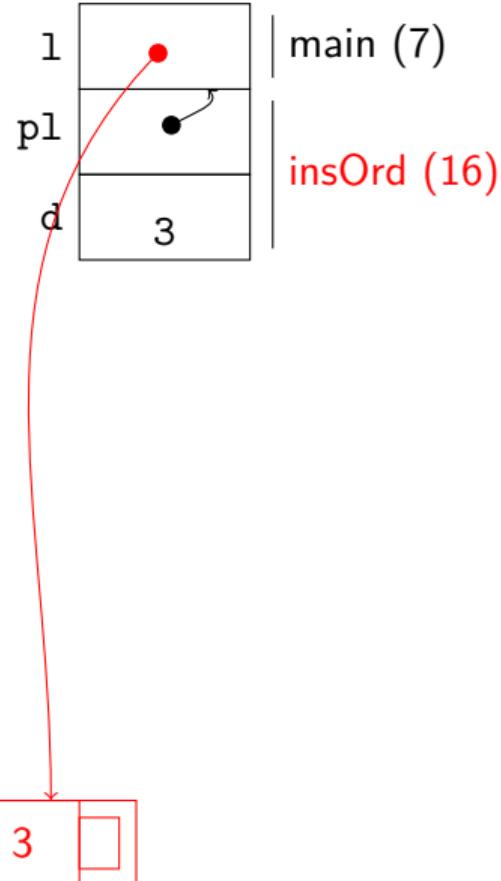
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



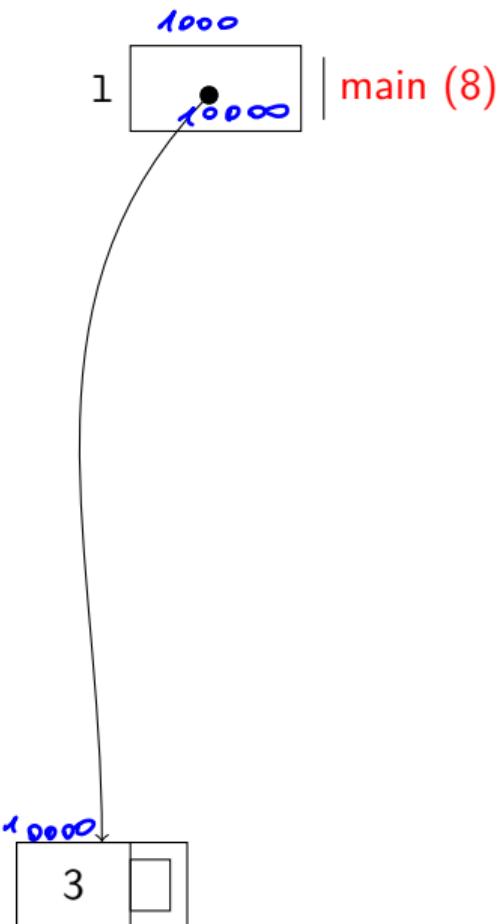
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



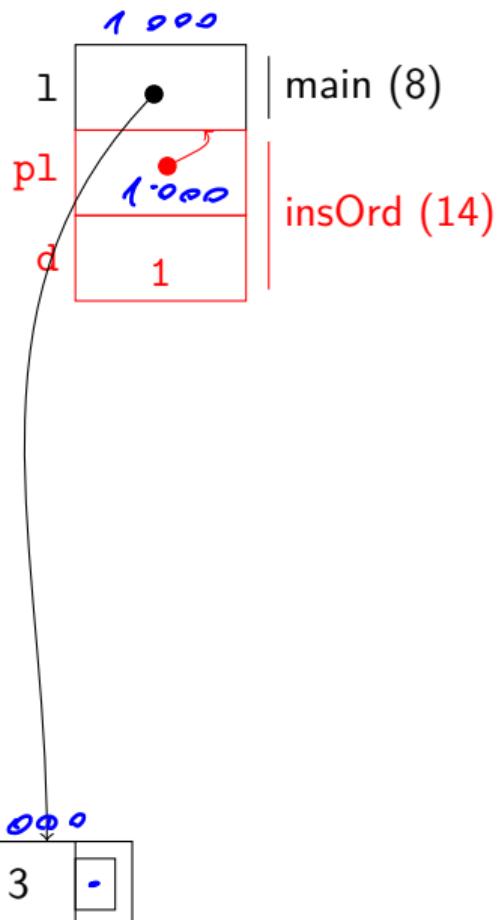
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insOrd.h"
3
4 int main(void) {
5     Lista l;
6     nuovaLista(&l);
7     insOrd(&l, 3);
8     insOrd(&l, 1);
9     insOrd(&l, 4);
10    insOrd(&l, 2);
11    return 0;
12 }
```



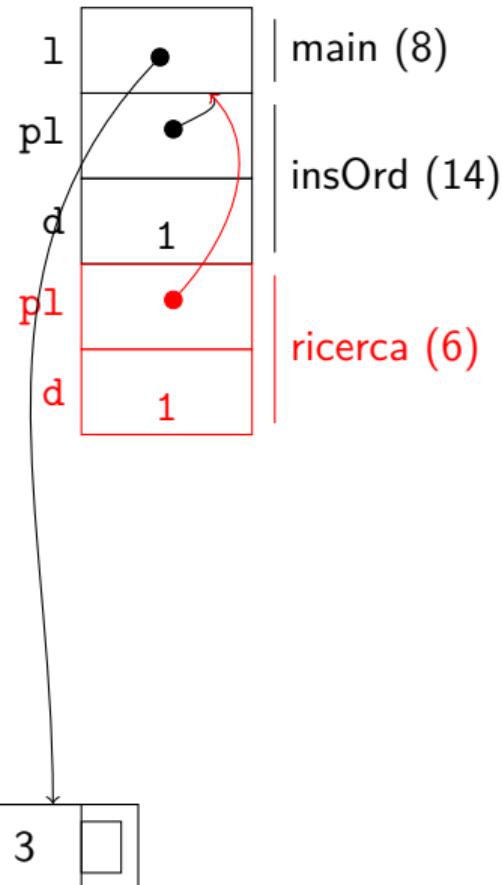
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



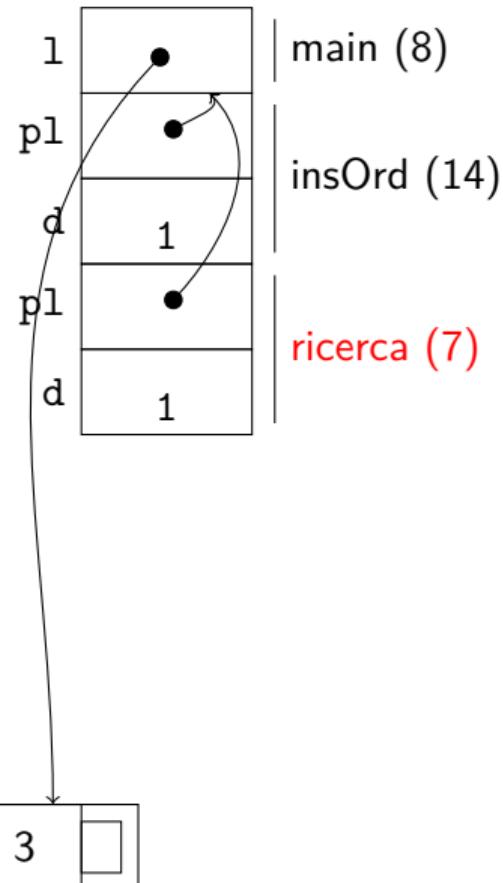
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl; ← 1000
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



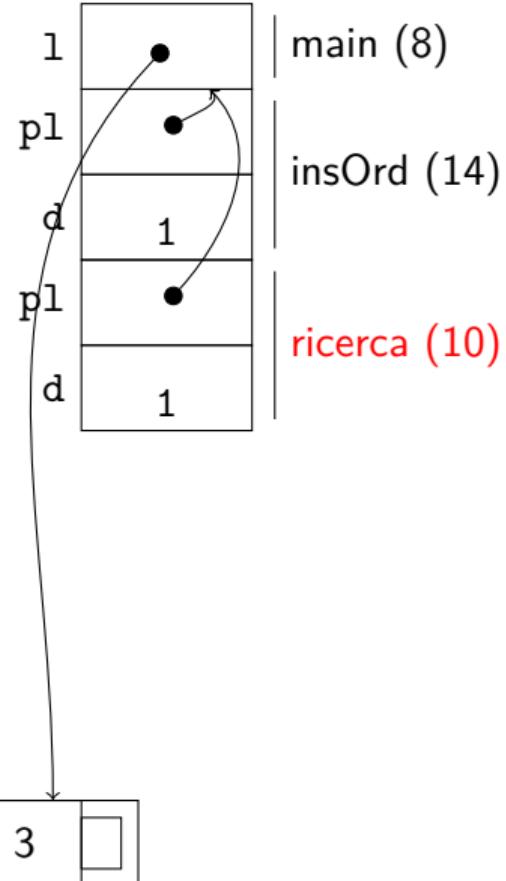
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



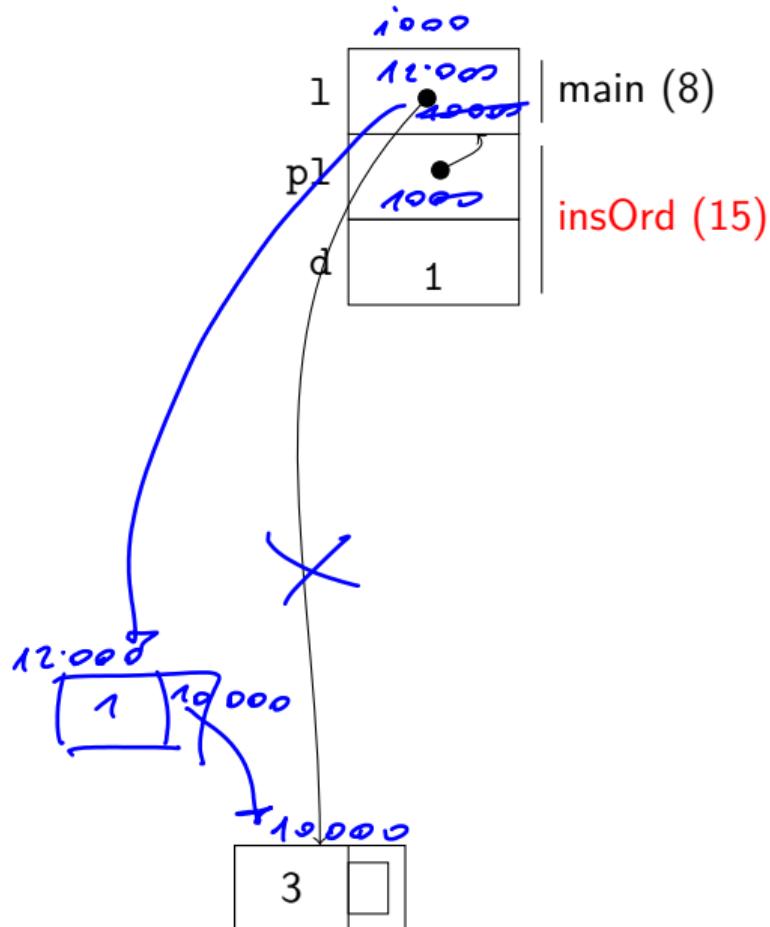
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d); 1000
15     insTesta(pl, d);
16 }
```



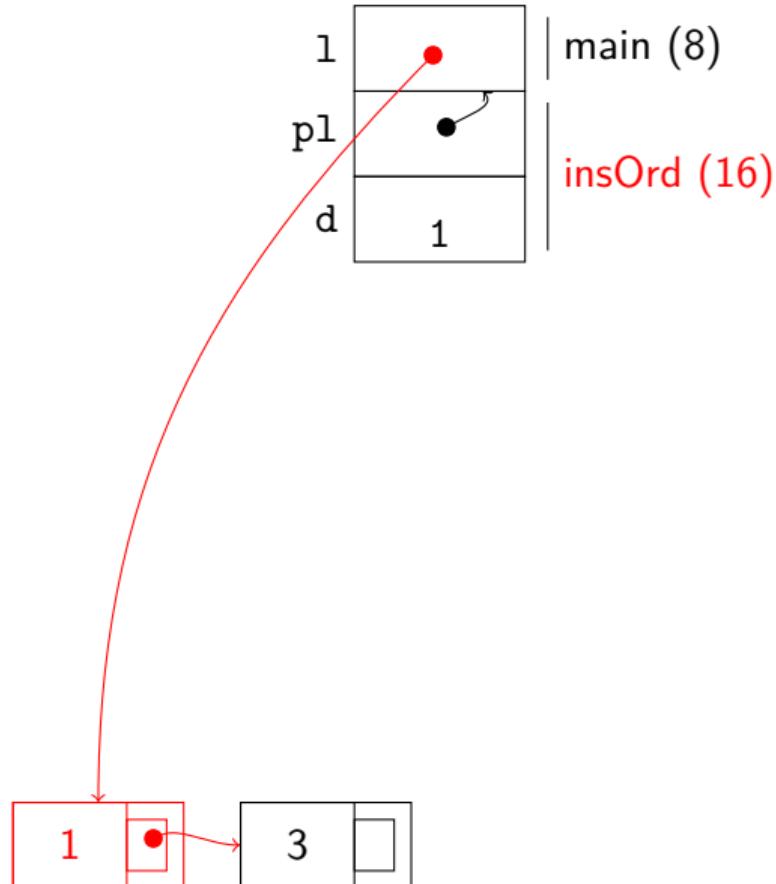
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



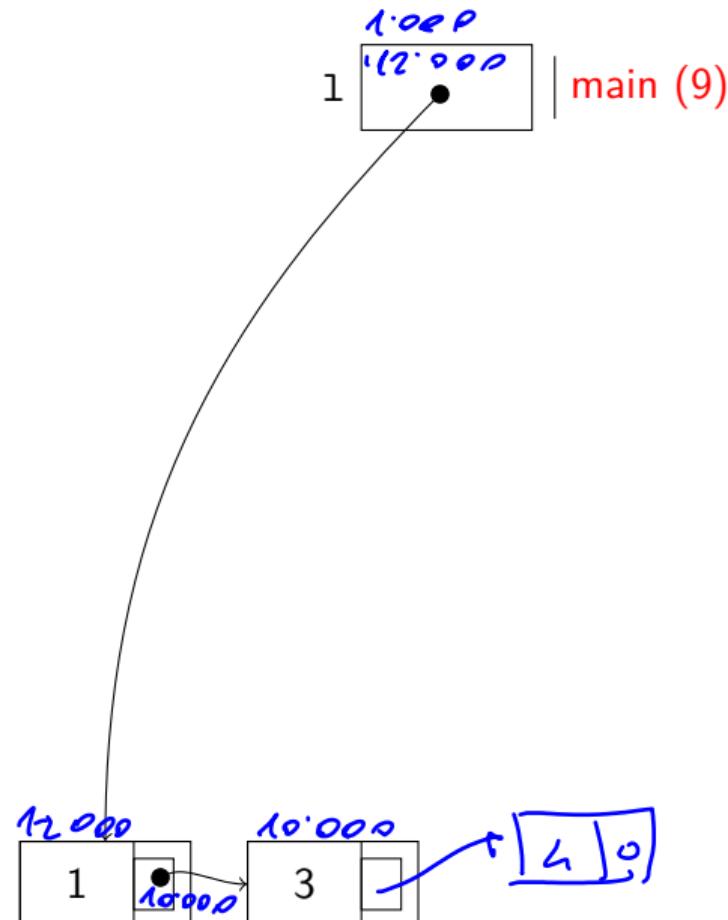
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



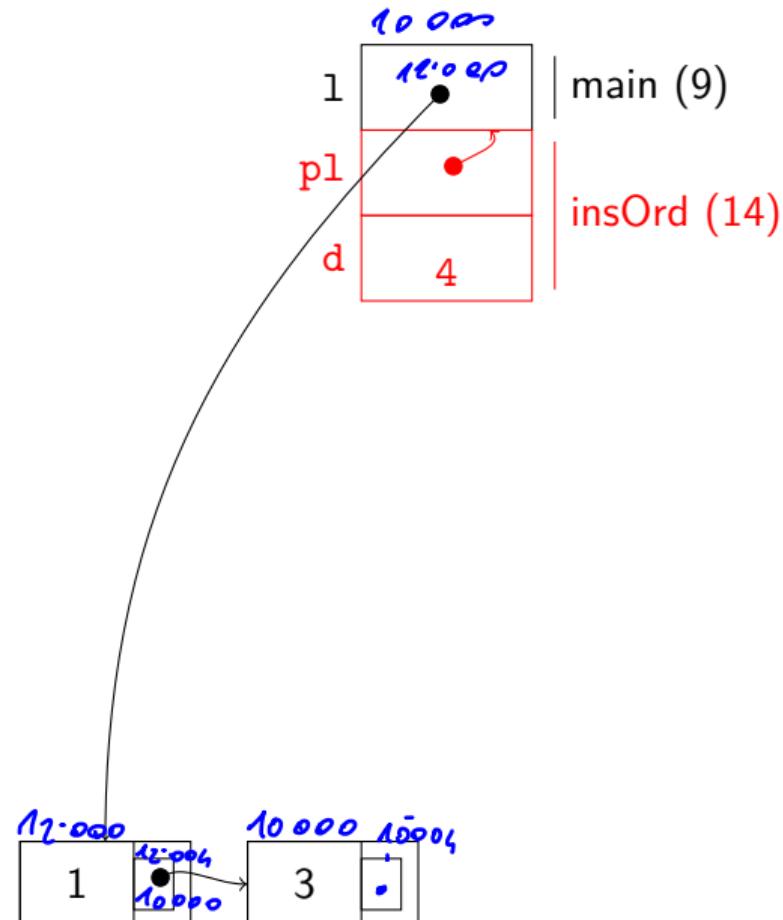
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insOrd.h"
3
4 int main(void) {
5     Lista l;
6     nuovaLista(&l);
7     insOrd(&l, 3);
8     insOrd(&l, 1);
9     insOrd(&l, 4);
10    insOrd(&l, 2);
11    return 0;
12 }
```



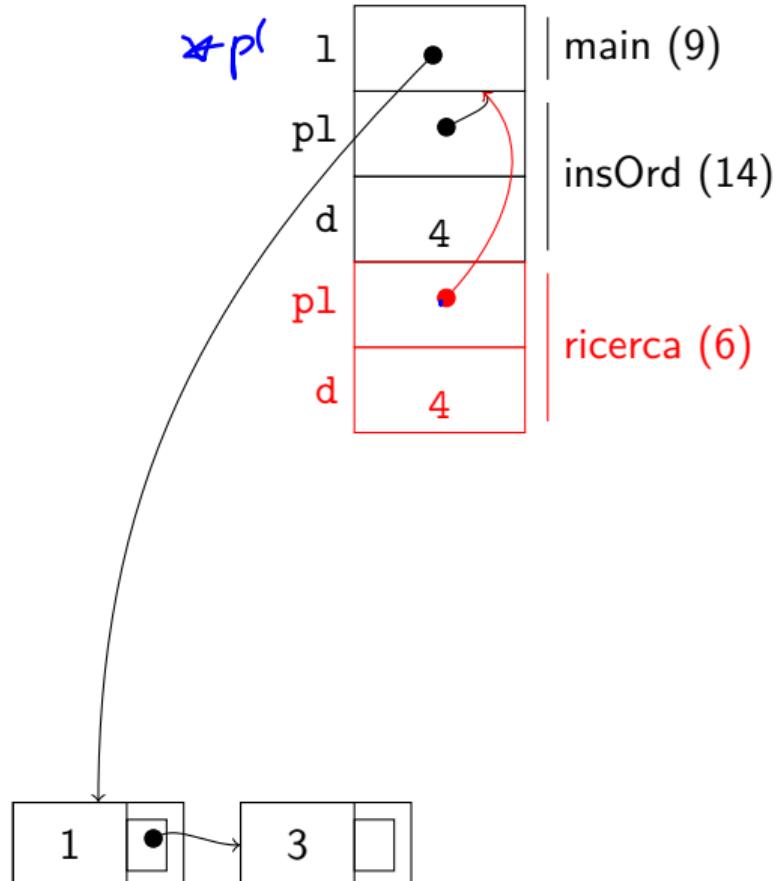
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d); 10'004
16 }
```



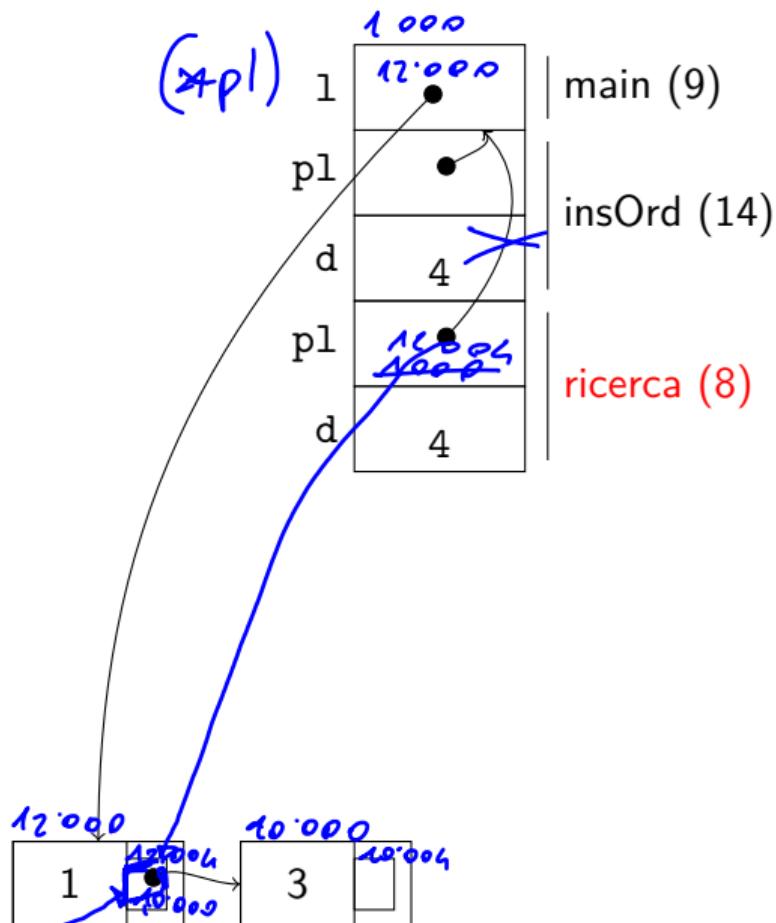
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



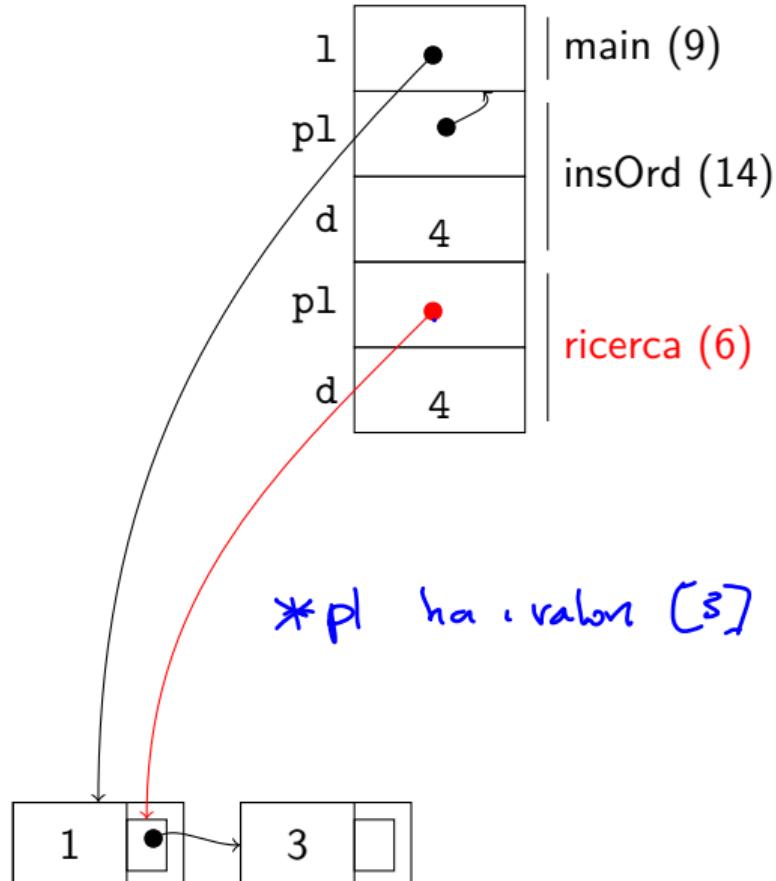
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;      11.004
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



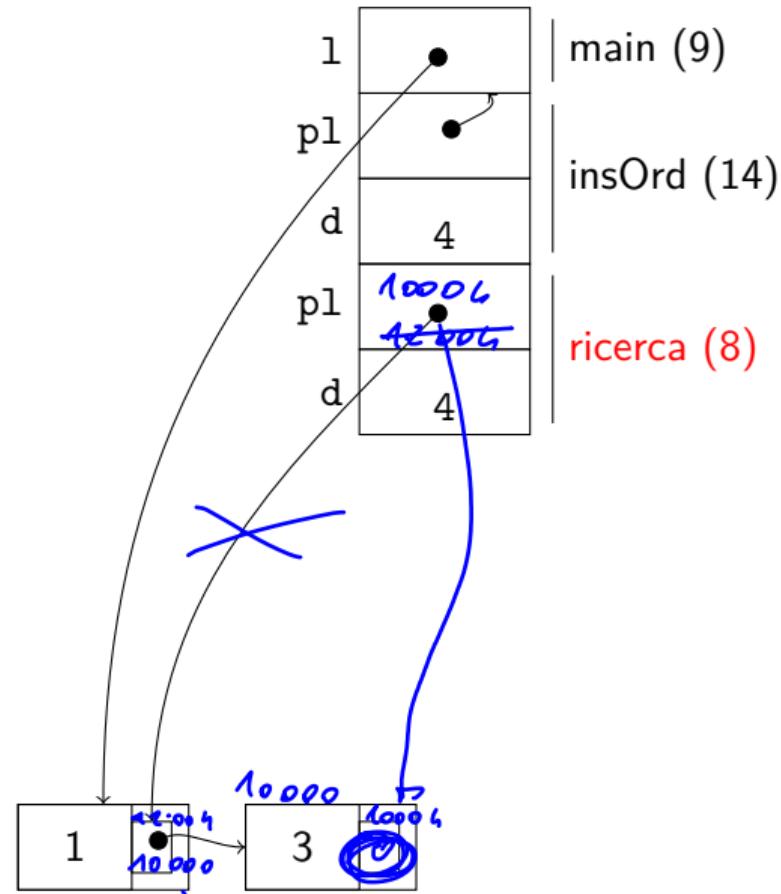
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



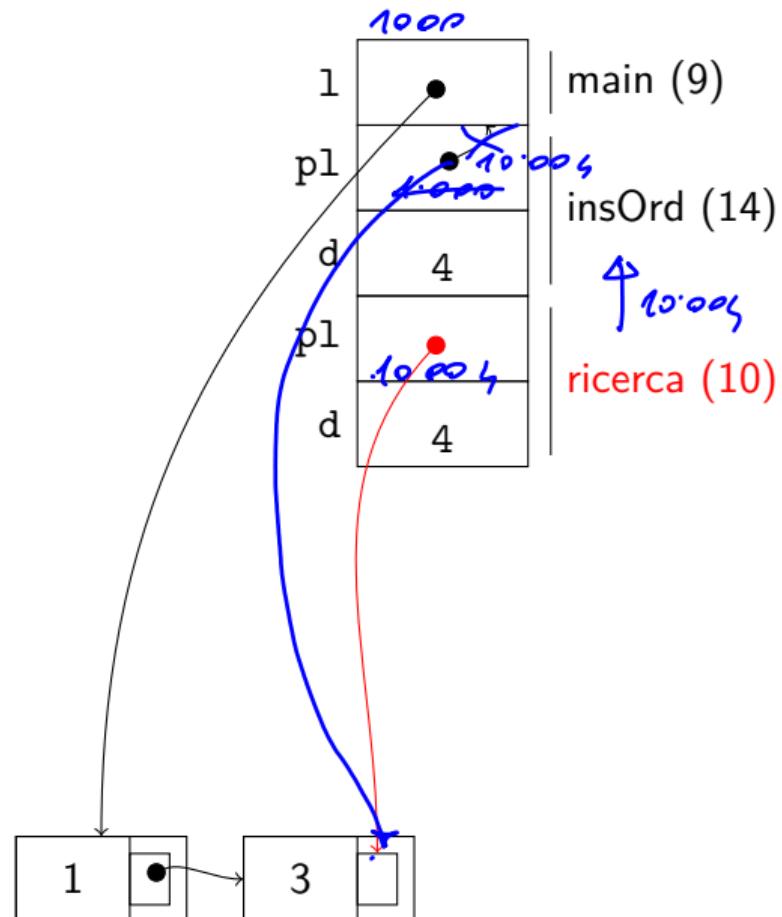
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl; 10000
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



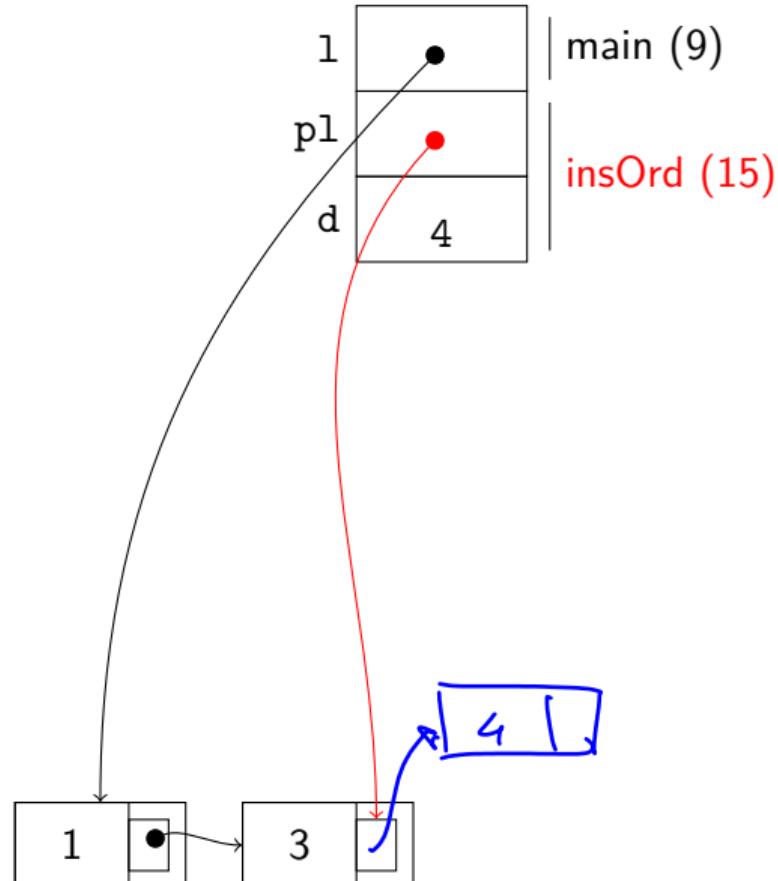
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



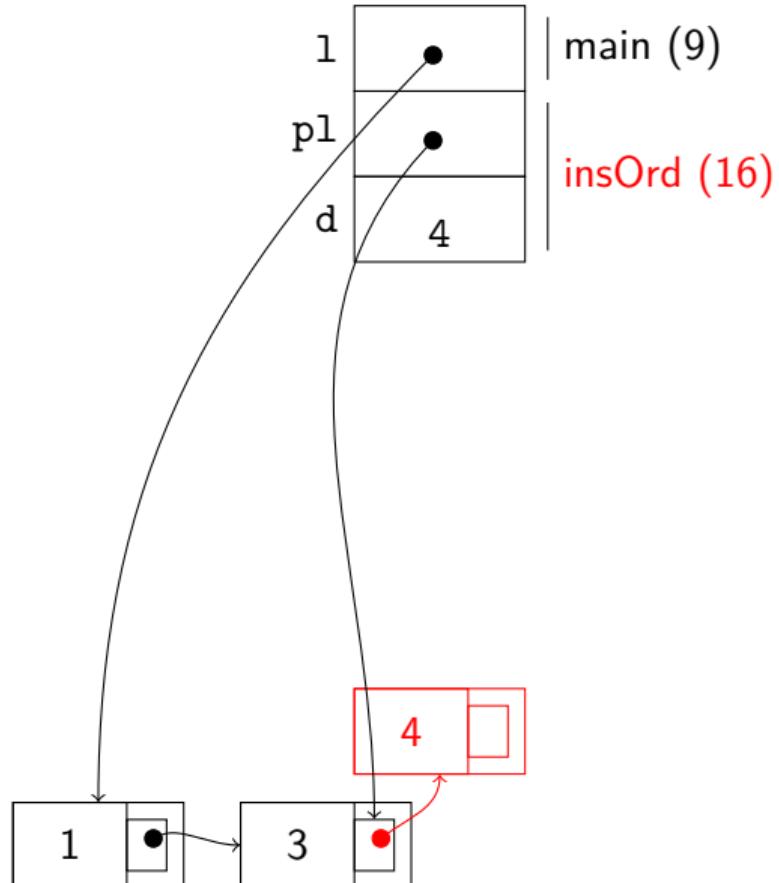
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



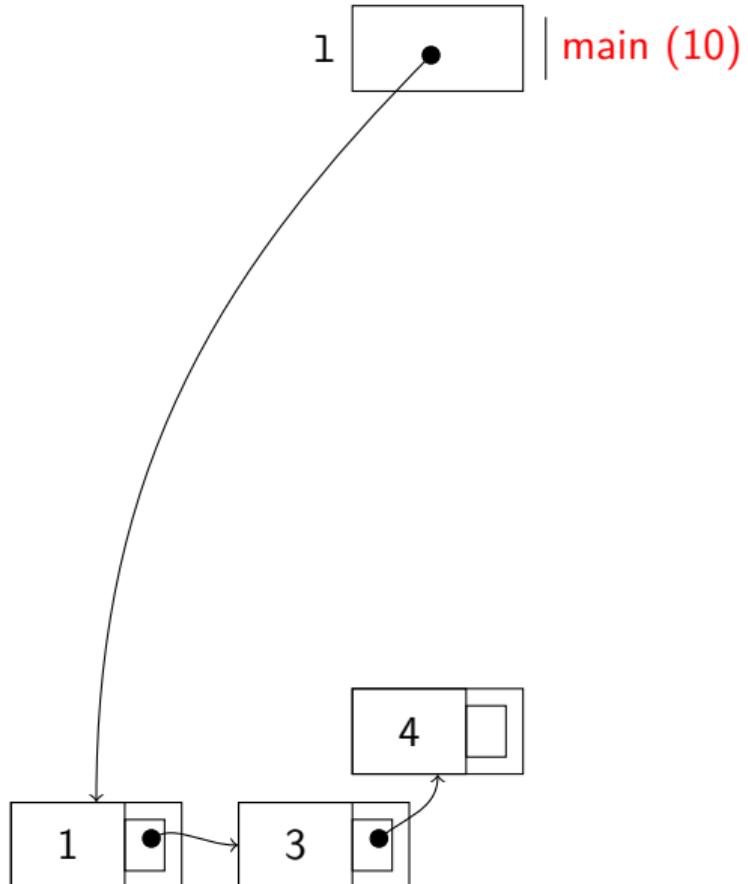
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



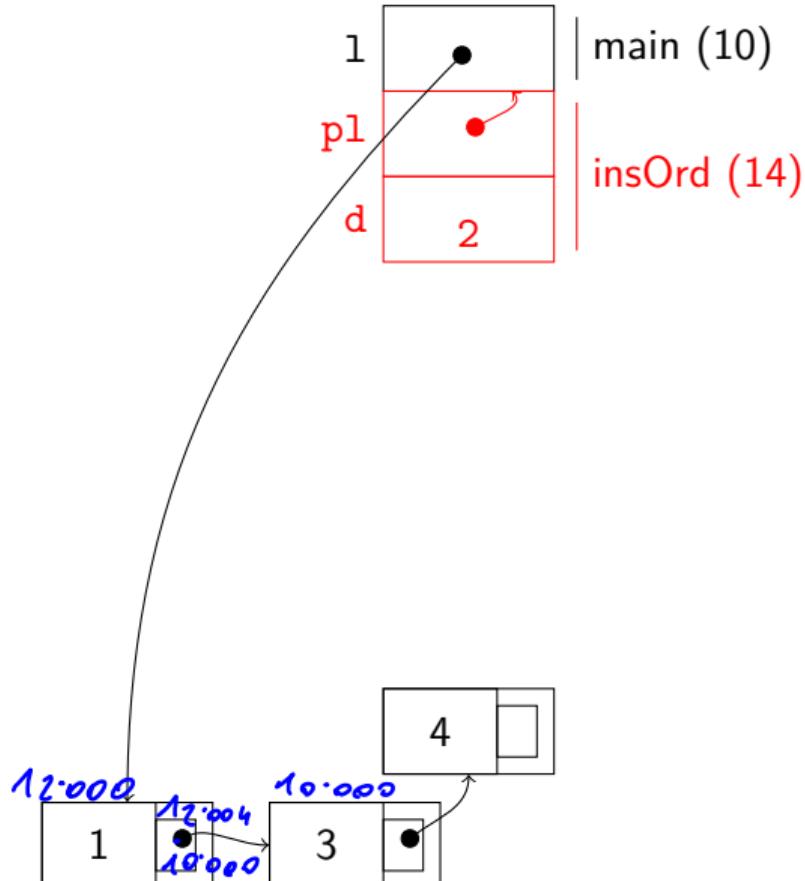
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insOrd.h"
3
4 int main(void) {
5     Lista l;
6     nuovaLista(&l);
7     insOrd(&l, 3);
8     insOrd(&l, 1);
9     insOrd(&l, 4);
10    insOrd(&l, 2);
11    return 0;
12 }
```



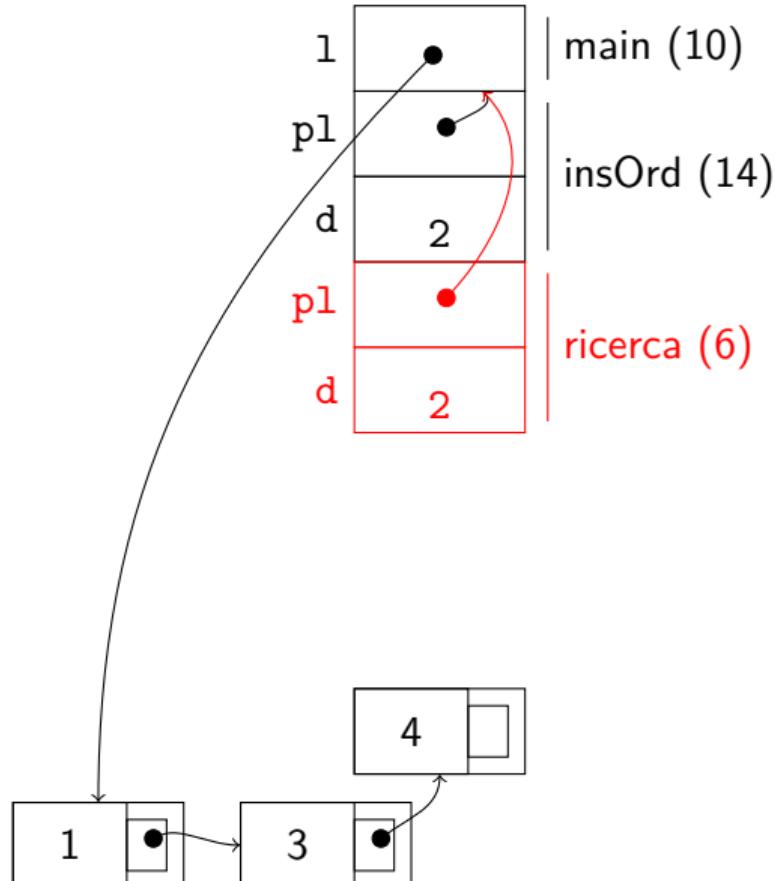
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



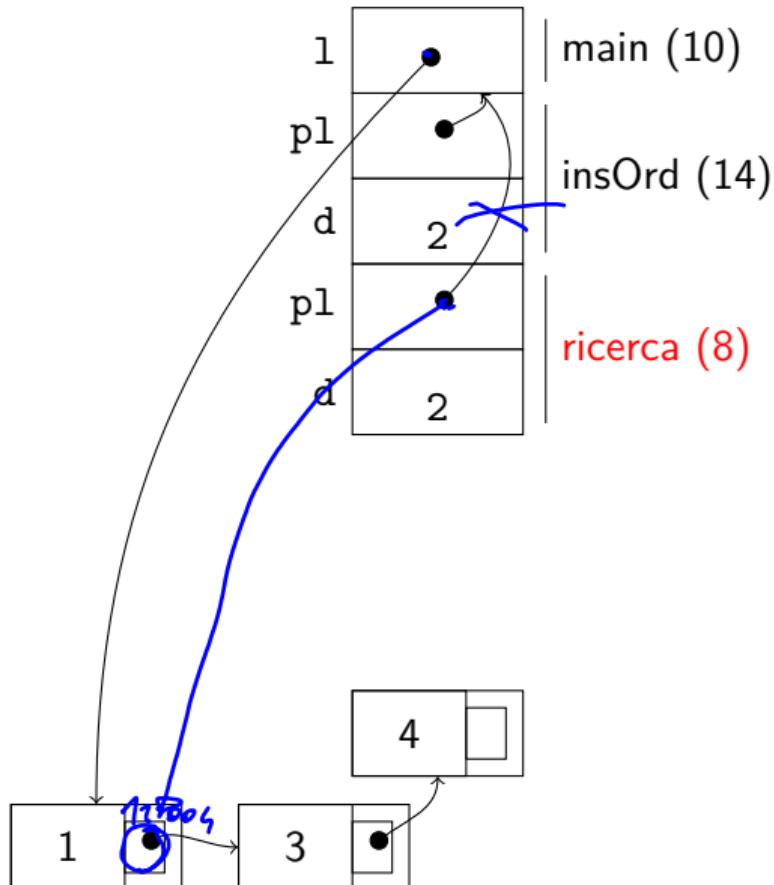
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) { 2
6         if ((*pl)->dato > d) 1
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



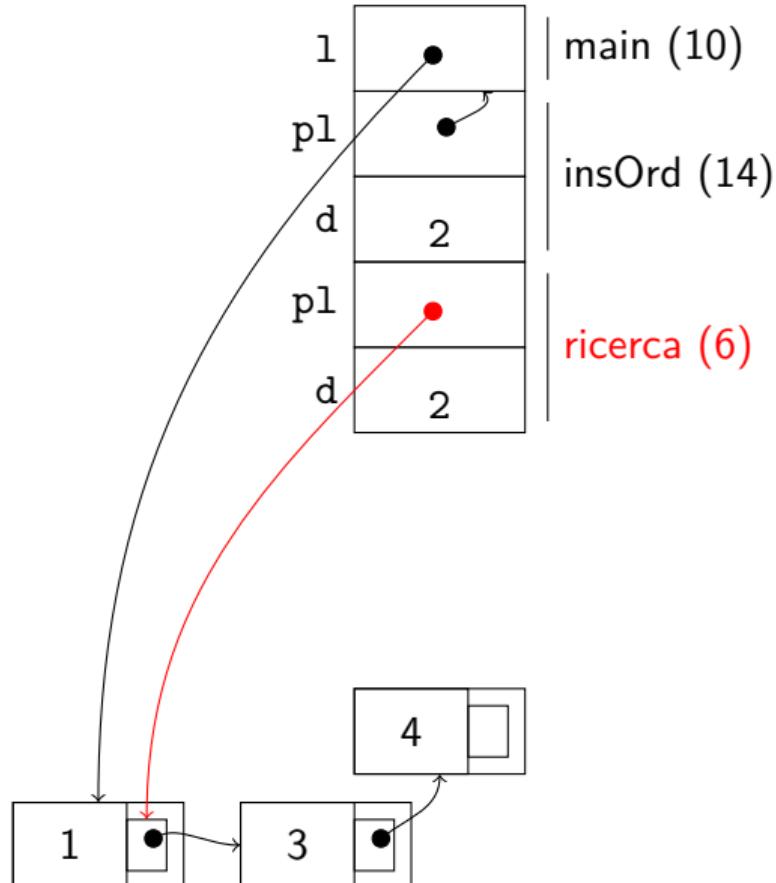
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11}
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16}
```



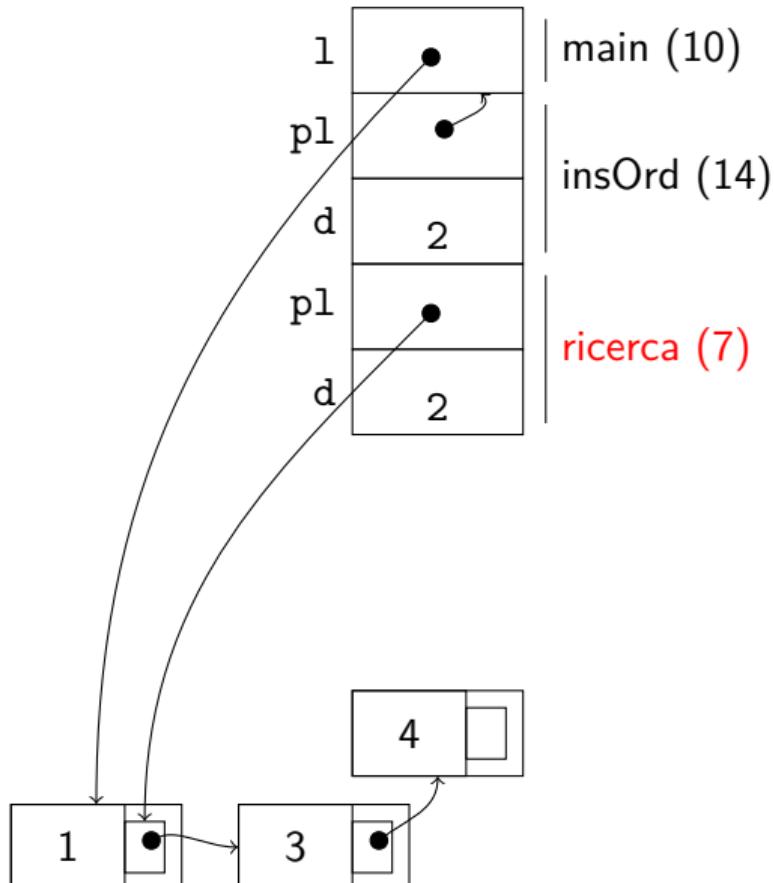
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d) 2
7             break; 3
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



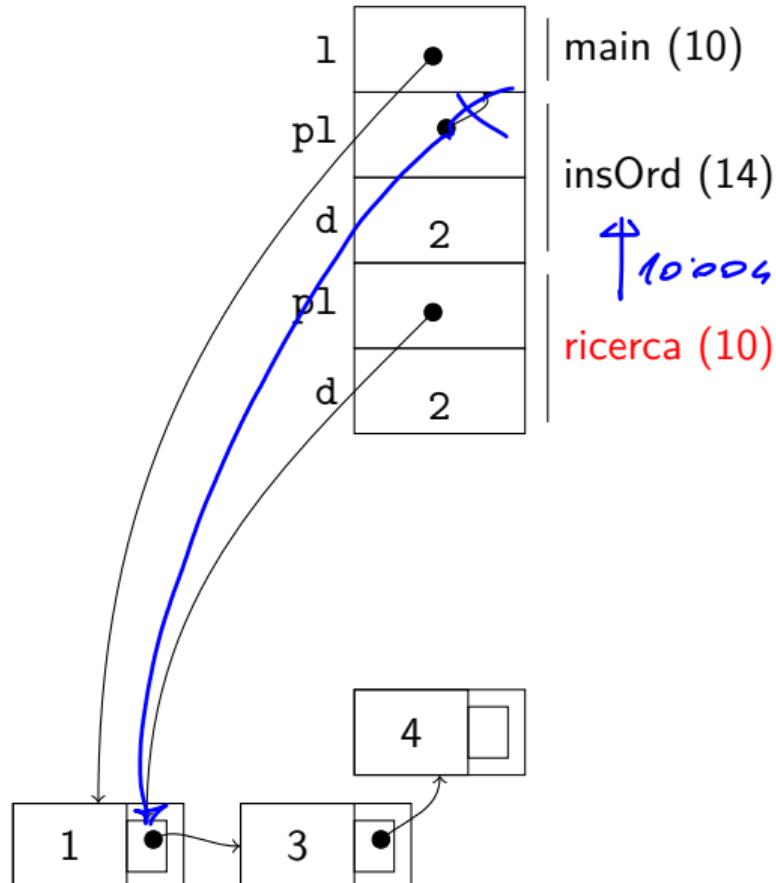
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



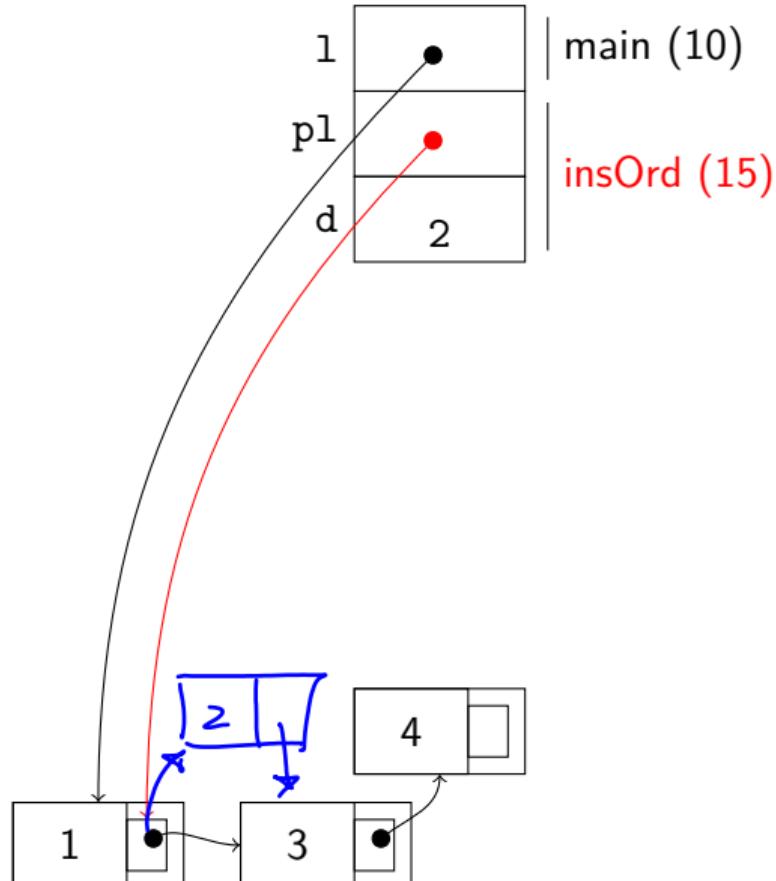
Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d); 1000(
16 }
```



Inserimento Ordinato

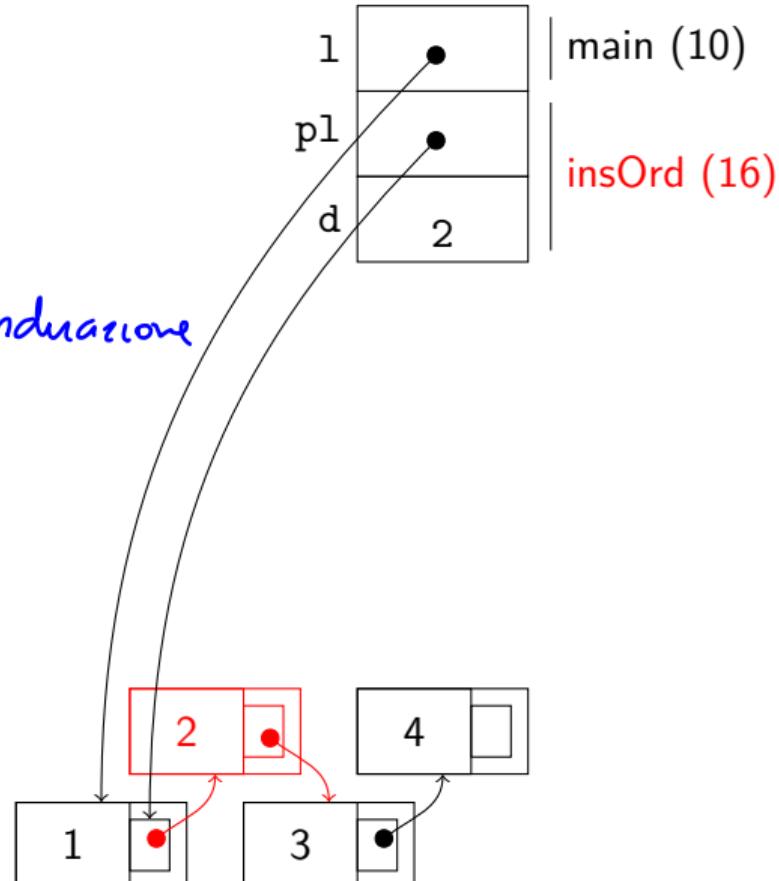
```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```



Inserimento Ordinato

```
1 #include "tipi.h"
2 #include "insTesta.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl) {
6         if ((*pl)->dato > d)
7             break;
8         pl = &(*pl)->next;
9     }
10    return pl;
11 }
12
13 void insOrd(Lista* pl, int d) {
14     pl = ricerca(pl, d);
15     insTesta(pl, d);
16 }
```

* criterio di individuazione
della lista

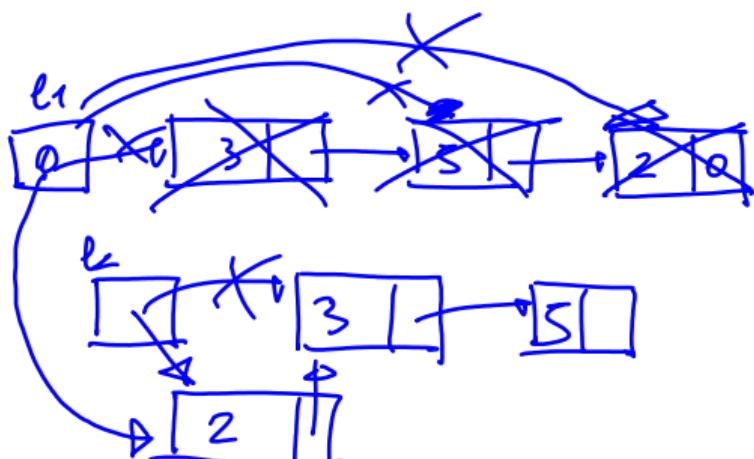


Insertion Sort per liste

Un algoritmo per ordinare una lista collegata 11:

- 1 crea una lista 12
- 2 finché ci sono elementi in 11
 - 2.1 inserisci in 12 la testa di 11
 - 2.2 elimina il primo elemento di 11
- 3 assegna 12 a 11

5	5	2
3	(5)	2
3	5	2
2	3	3

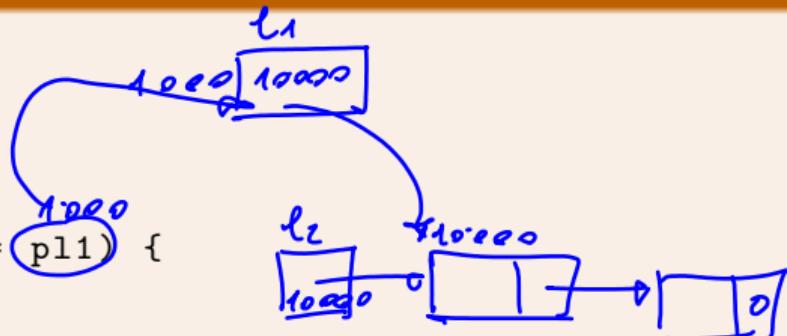


Insertion Sort

230_liste_collegate/insertionSort.c

```

1 #include "tipi.h"
2 #include "elimTesta.h"
3 #include "ins0rd.h"
4
5 void insertionSort(Lista* p11) {
6     Lista l2;
7     nuovaLista(&l2);
8     while (*p11) {
9         ins0rd(&l2, (*p11)->dato);
10        elimTesta(p11);
11    }
12    *p11 = l2;
13 }
```



Inserimento in coda

$$[2, 5, 3] \rightarrow [2, 5, 3, 4]$$

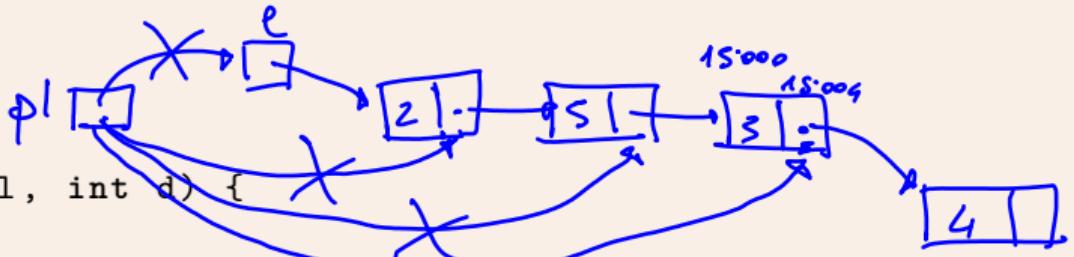
Simile all'inserimento ordinato, ma la lista in cui effettuare l'inserimento è sempre quella vuota (cioè il campo **next** dell'ultimo nodo).

230_liste_collegate/insCoda.c

```

1 #include "insTesta.h"
2 #include "tipi.h"
3
4 Lista* ricerca(Lista* pl, int d) {
5     while (*pl)
6         pl = &(*pl)->next;
7     return pl;
8 }
9
10 void insCoda(Lista* pl, int d) {
11     pl = ricerca(pl, d);
12     insTesta(pl, d);
13 }
14
15:006

```



Esercizio



Array To List

Scrivere una funzione che, dato un array di interi e la sua dimensione logica, restituisca una lista di interi contenente gli stessi elementi.

Lista $\text{arrayToList}(\text{int } v[], \text{int } dl)$ $\rightarrow 10.000$



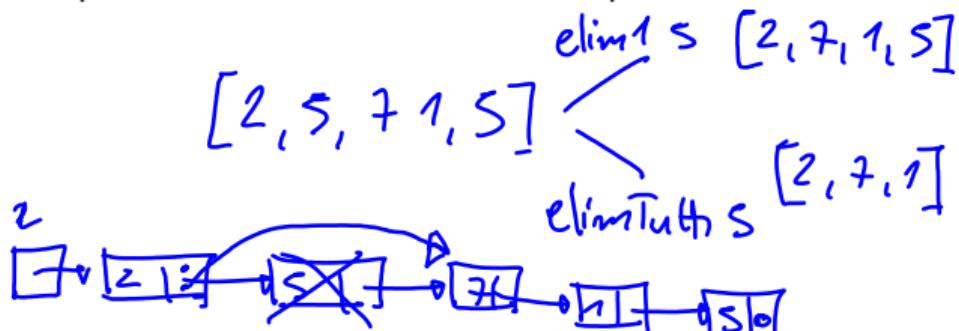
Ricerca ed eliminazione di un elemento

Per eliminare da una lista un elemento che ha una determinata proprietà (ad esempio avere valore **d**):

- 1 ricerca la lista **11** la cui testa ha la proprietà desiderata (ad esempio campo **dato** uguale a **d**);
- 2 se esiste, elimina il primo elemento di **11**.

In generale una lista può contenere più elementi con la proprietà ricercata. Si può decidere se eliminare solo la prima occorrenza e fermarsi, o se continuare fino alla fine della lista per eliminarli tutti.

Si può usare il valore di ritorno per indicare se l'elemento è stato eliminato oppure no.



Eliminazione di un elemento ricercato

230_liste_collegate/elim1.c

```

1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 int elim1(Lista* pl, int d) {
15     pl = ricerca(pl, d);
16     if (*pl) {
17         elimTesta(pl);
18         return 1;
19     } else
20         return 0;
21 }
```

*testa di *pl*

uguale a elemento

da eliminare

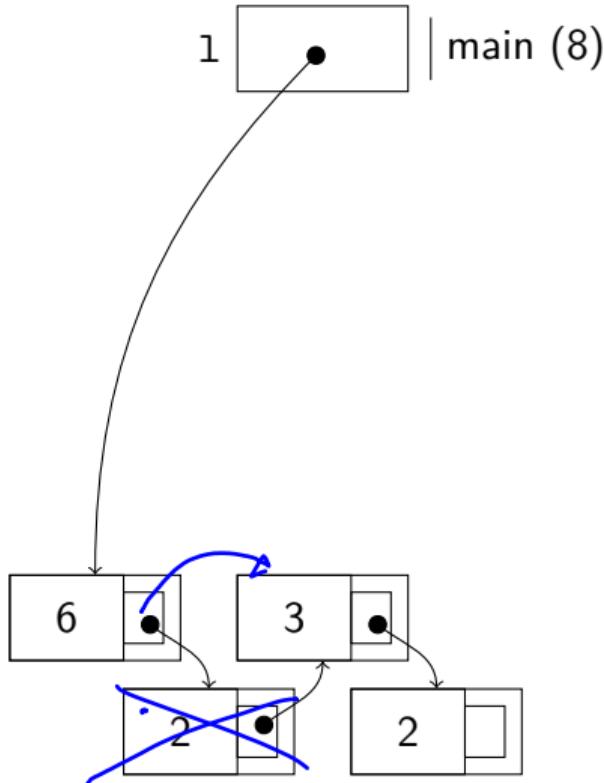
e indirizzo della prima lista

la cui testa

è elem da eliminare

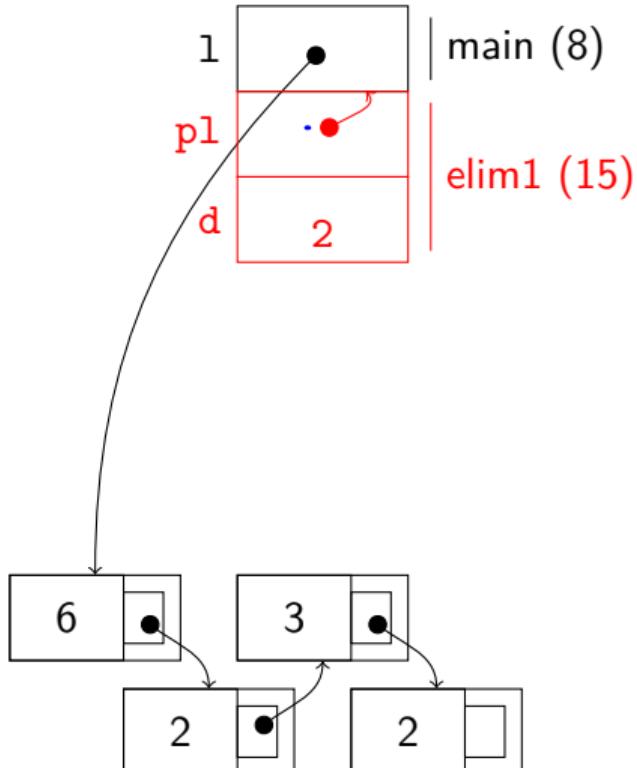
Eliminazione del primo elemento ricercato

```
1 #include "tipi.h"
2 #include "generatoreListe.h"
3 #include "elim1.h"
4
5 int main(void) {
6     Lista l;
7     listaNonOrdinata(&l, 4);
8     elim1(&l, 2);
9     return 0;
10 }
```



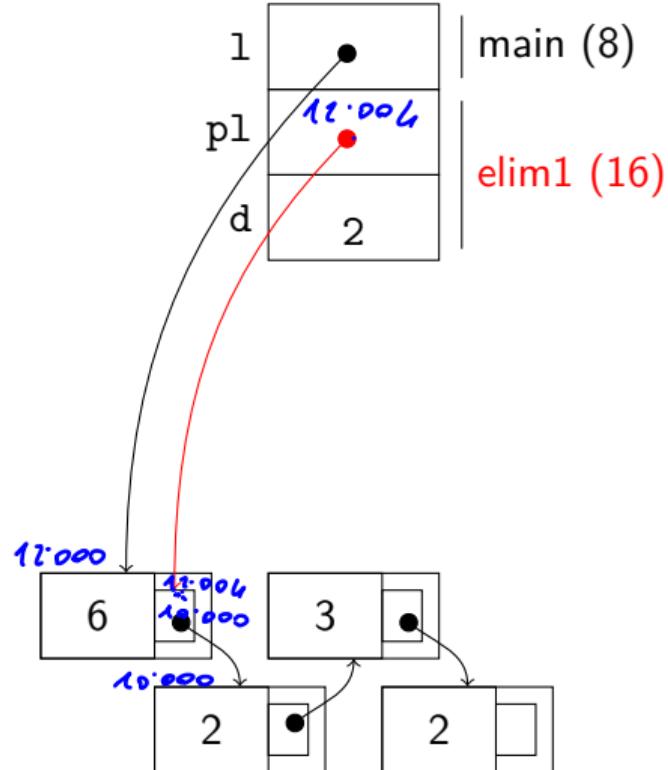
Eliminazione del primo elemento ricercato

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 int elim1(Lista* pl, int d) {
15     pl = ricerca(pl, d);
16     if (*pl) {
17         elimTesta(pl);
18         return 1;
19     } else
20         return 0;
21 }
```



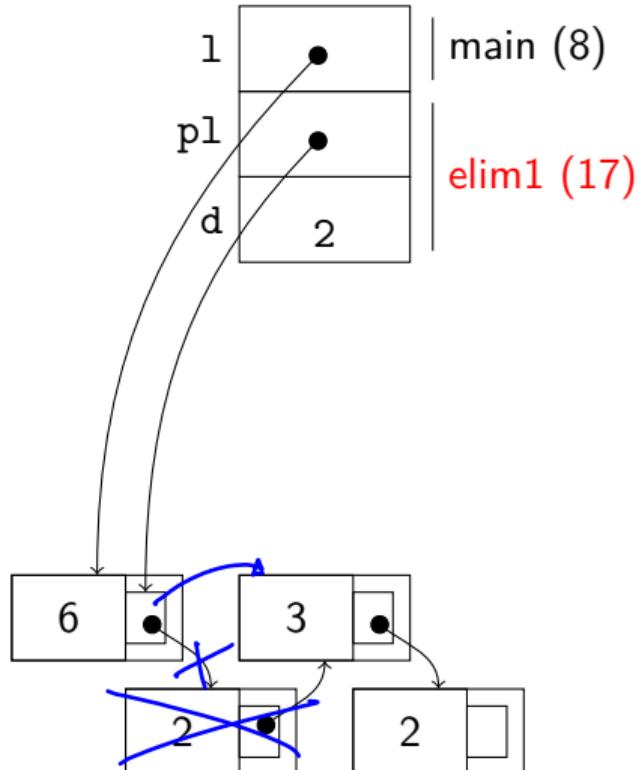
Eliminazione del primo elemento ricercato

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 int elim1(Lista* pl, int d) {
15     . pl = ricerca(pl, d);
16     if (*pl) {
17         elimTesta(pl);
18         return 1;
19     } else
20         return 0;
21 }
```



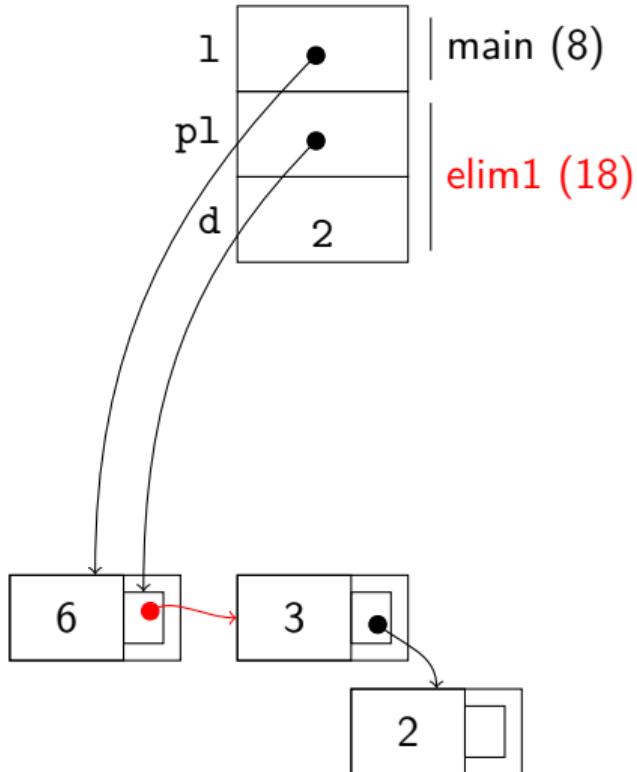
Eliminazione del primo elemento ricercato

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 int elim1(Lista* pl, int d) {
15     pl = ricerca(pl, d);
16     if (*pl) {
17         elimTesta(pl);
18         return 1;
19     } else
20         return 0;
21 }
```



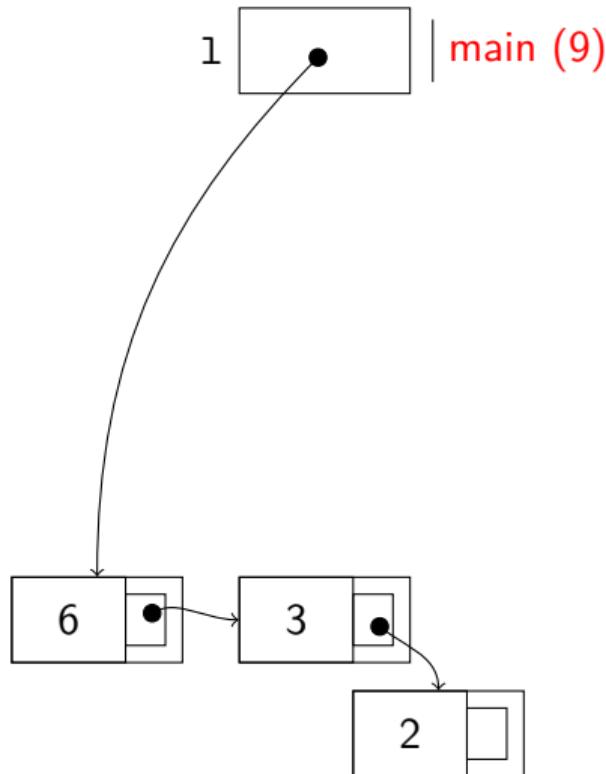
Eliminazione del primo elemento ricercato

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 int elim1(Lista* pl, int d) {
15     pl = ricerca(pl, d);
16     if (*pl) {
17         * elimTesta(pl);
18         return 1;
19     } else
20         return 0;
21 }
```



Eliminazione del primo elemento ricercato

```
1 #include "tipi.h"
2 #include "generatoreListe.h"
3 #include "elim1.h"
4
5 int main(void) {
6     Lista l;
7     listaNonOrdinata(&l, 4);
8     elim1(&l, 2);
9     return 0;
10 }
```



Eliminazione di tutti gli elementi ricercati

230_liste_collegate/elimTutti.c

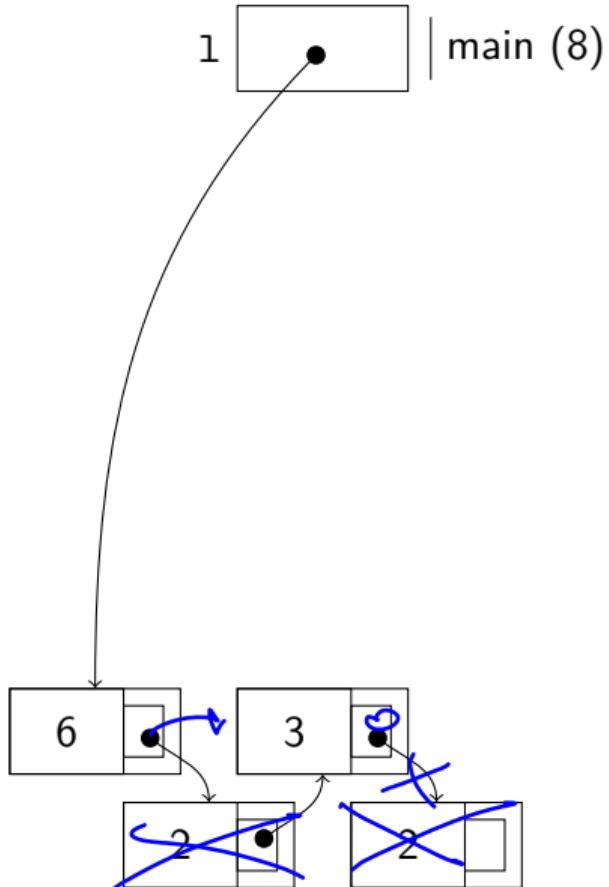
```

1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 void elimTutti(Lista* pl, int d) {
15     while (*pl) {
16         pl = ricerca(pl, d);
17         if (*pl)
18             elimTesta(pl);
19     }
20 }
```



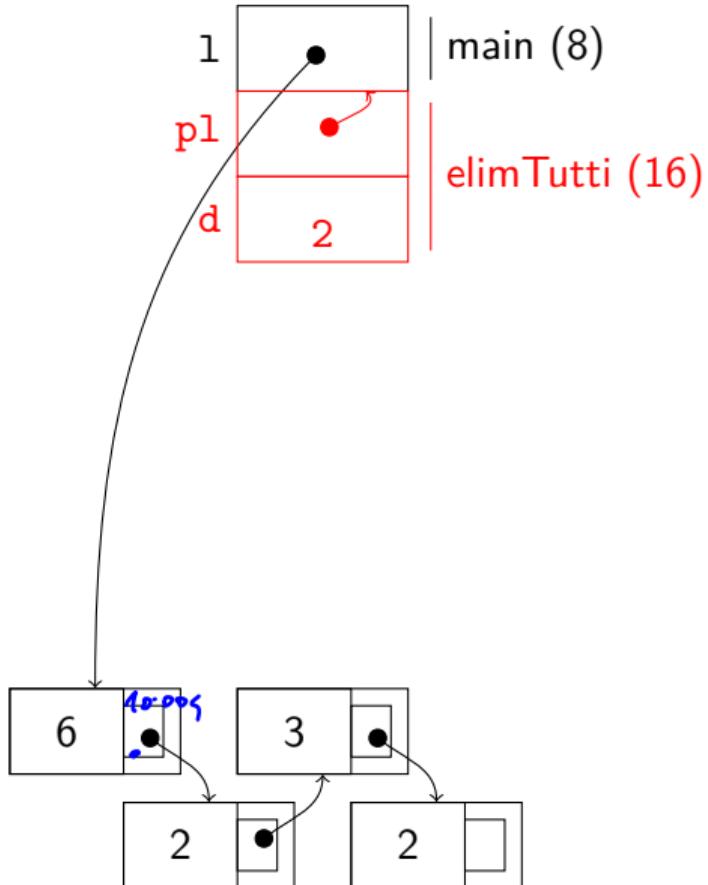
Eliminazione di tutti gli elementi ricercati

```
1 #include "tipi.h"
2 #include "generatoreListe.h"
3 #include "elimTutti.h"
4
5 int main(void) {
6     Lista l;
7     listaNonOrdinata(&l, 4);
8     elimTutti(&l, 2);
9     return 0;
10 }
```



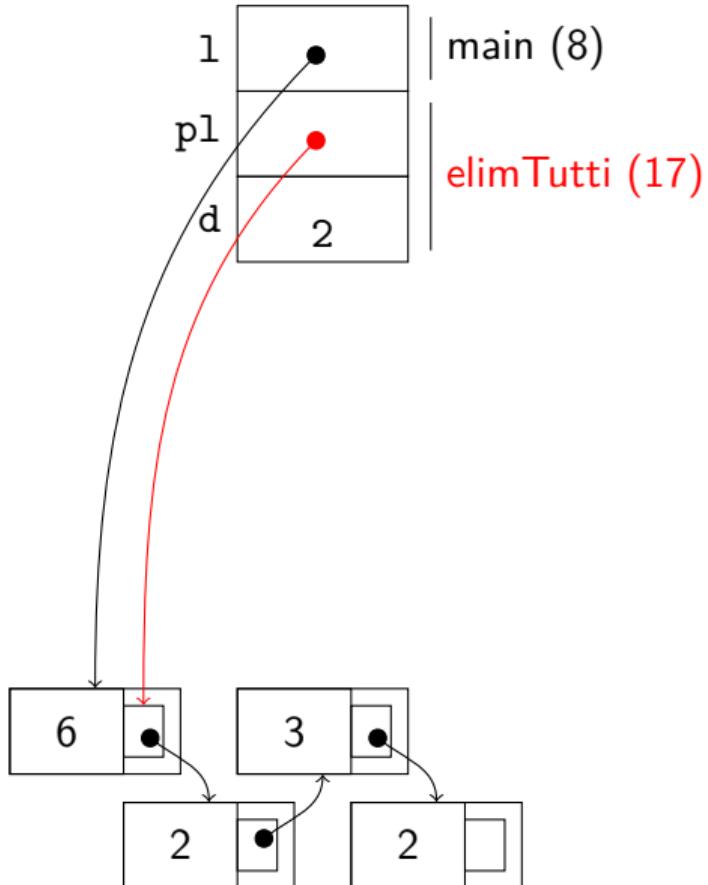
Eliminazione di tutti gli elementi ricercati

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 void elimTutti(Lista* pl, int d) {
15     while (*pl) {
16         pl = ricerca(pl, d);
17         if (*pl)
18             elimTesta(pl);
19     }
20 }
```



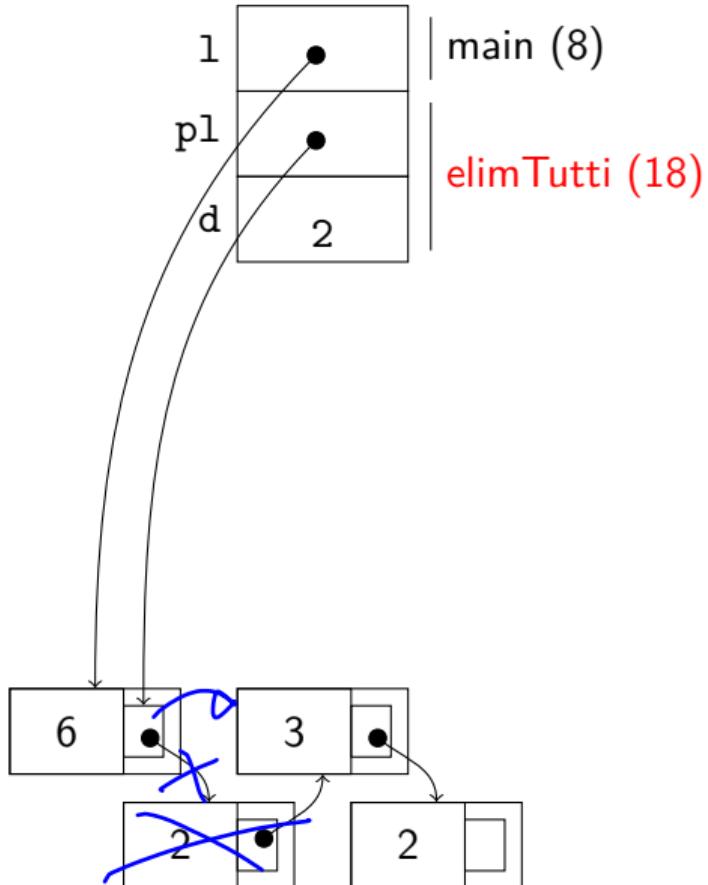
Eliminazione di tutti gli elementi ricercati

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 void elimTutti(Lista* pl, int d) {
15     while (*pl) {
16         pl = ricerca(pl, d);
17         if (*pl)
18             elimTesta(pl);
19     }
20 }
```



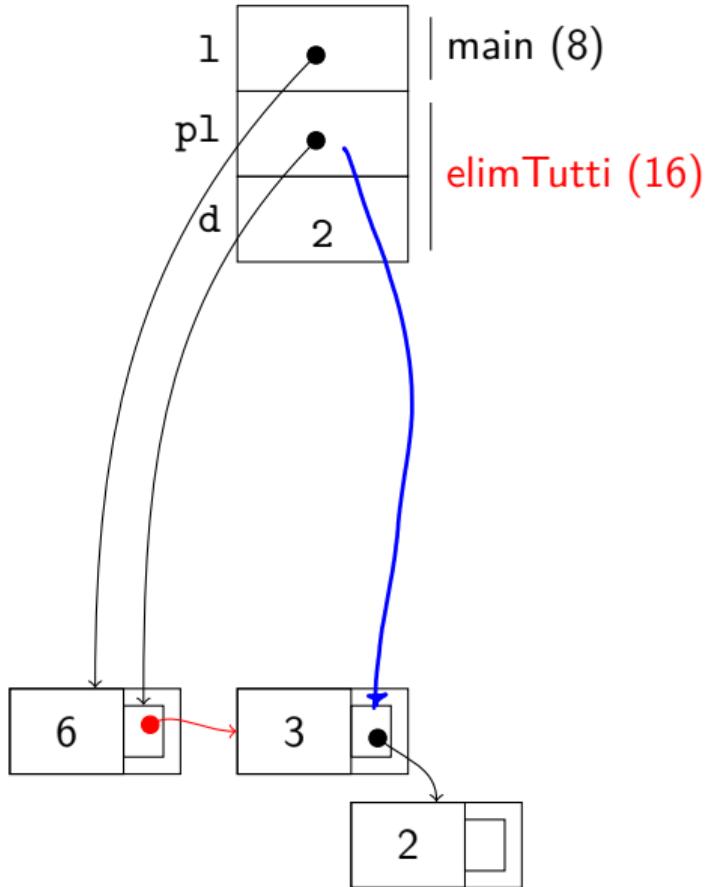
Eliminazione di tutti gli elementi ricercati

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 void elimTutti(Lista* pl, int d) {
15     while (*pl) {
16         pl = ricerca(pl, d);
17         if (*pl)
18             elimTesta(pl);
19     }
20 }
```



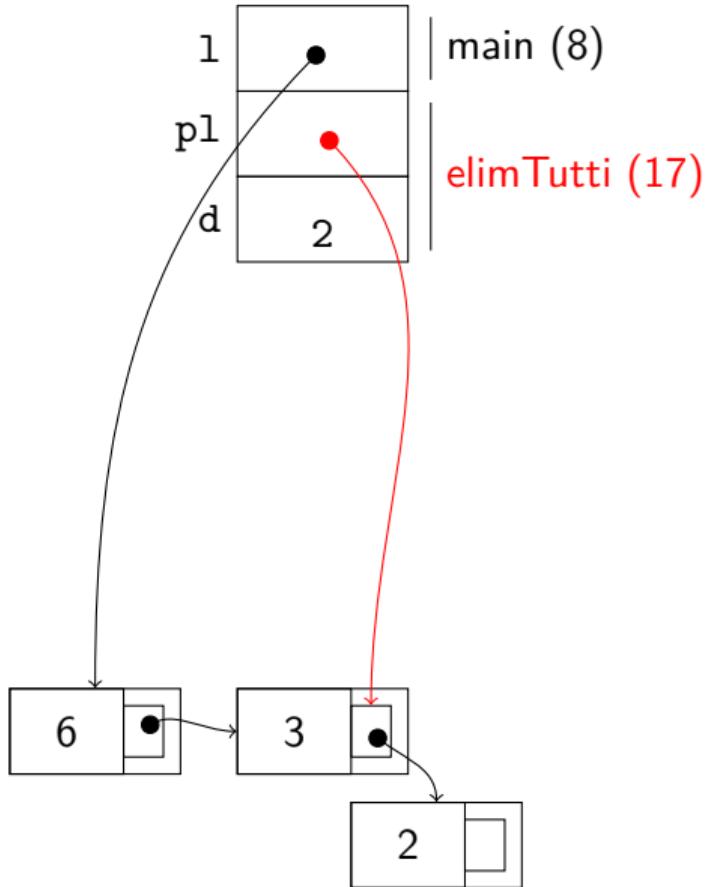
Eliminazione di tutti gli elementi ricercati

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 void elimTutti(Lista* pl, int d) {
15     while (*pl) {
16         pl = ricerca(pl, d);
17         if (*pl)
18             elimTesta(pl);
19     }
20 }
```



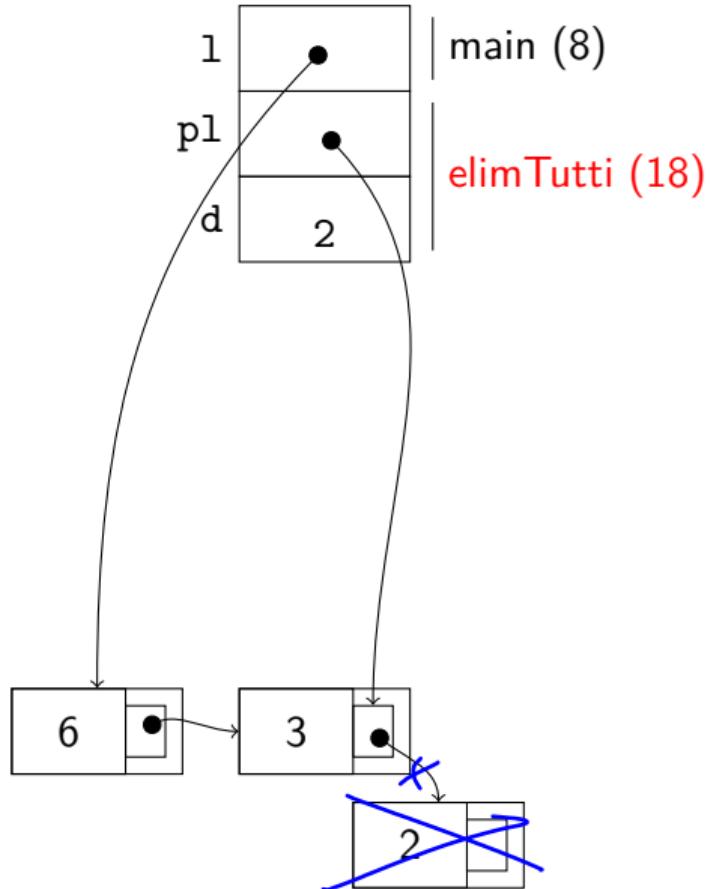
Eliminazione di tutti gli elementi ricercati

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 void elimTutti(Lista* pl, int d) {
15     while (*pl) {
16         pl = ricerca(pl, d);
17         if (*pl)
18             elimTesta(pl);
19     }
20 }
```



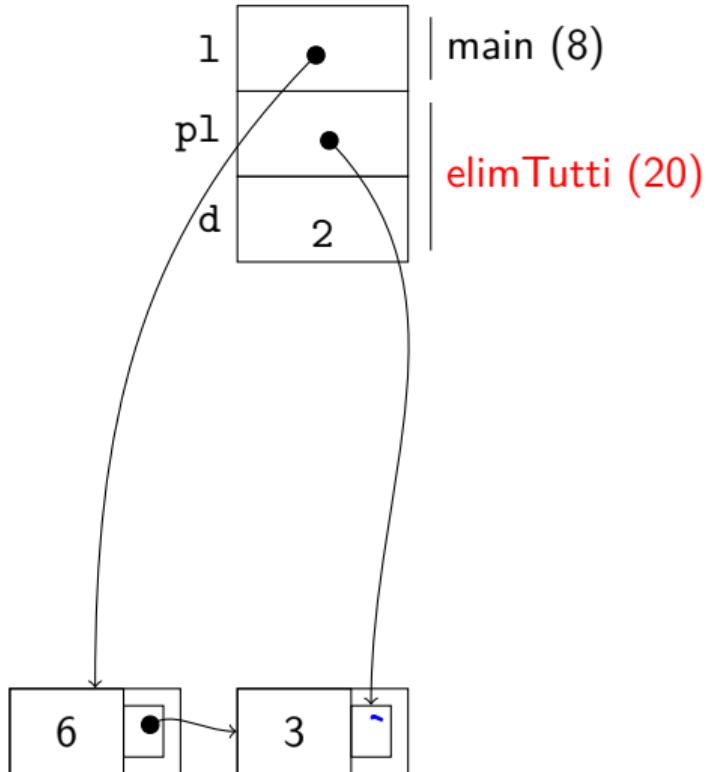
Eliminazione di tutti gli elementi ricercati

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 void elimTutti(Lista* pl, int d) {
15     while (*pl) {
16         pl = ricerca(pl, d);
17         if (*pl)
18             elimTesta(pl);
19     }
20 }
```



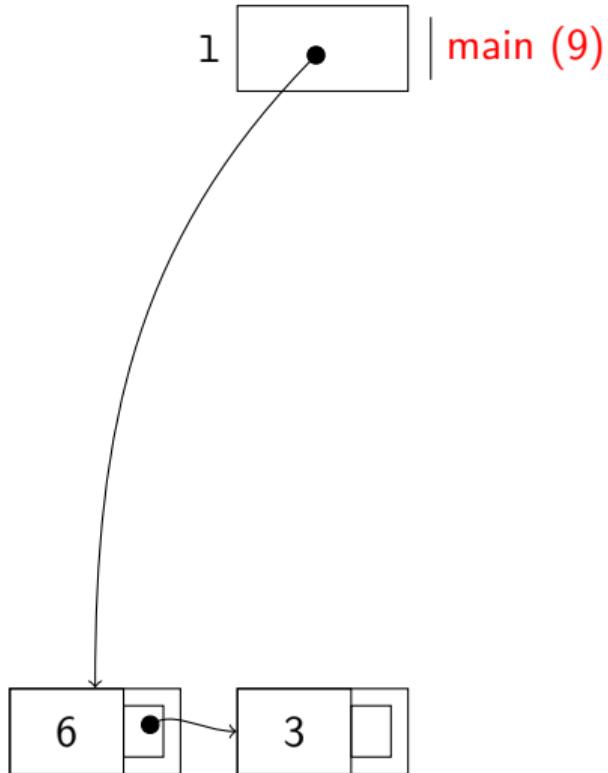
Eliminazione di tutti gli elementi ricercati

```
1 #include <malloc.h>
2 #include "tipi.h"
3 #include "elimTesta.h"
4
5 Lista* ricerca(Lista* pl, int d) {
6     while (*pl) {
7         if ((*pl)->dato == d)
8             break;
9         pl = &(*pl)->next;
10    }
11    return pl;
12 }
13
14 void elimTutti(Lista* pl, int d) {
15     while (*pl) {
16         pl = ricerca(pl, d);
17         if (*pl)
18             elimTesta(pl);
19     }
20 }
```



Eliminazione di tutti gli elementi ricercati

```
1 #include "tipi.h"
2 #include "generatoreListe.h"
3 #include "elimTutti.h"
4
5 int main(void) {
6     Lista l;
7     listaNonOrdinata(&l, 4);
8     elimTutti(&l, 2);
9     return 0;
10 }
```



Esercizio

20170130p1

Modificare la soluzione dell'esercizio del Laboratorio 10 con liste sequenziali, pubblicata al link `230_liste_collegate/..../210_liste_sequenziali/calorie`, sostituendo le liste sequenziali con liste collegate.

I file `main.c` e `Makefile` devono rimanere invariati.

Esercizio

20170622

Modificare la soluzione dell'esercizio all'URL [230_liste_collegate/seq/20170622](https://230.liste.collegate.it/seq/20170622) sostituendo le liste sequenziali con liste collegate.
I file **main.c** e **Makefile** devono rimanere invariati.