Fondamenti di Programmazione (A)

20 - Liste

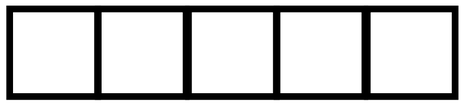
Vincenzo Arceri - Università degli Studi di Parma - vincenzo.arceri@unipr.it

Puntate precedenti

- Funzioni
- Passaggio di parametri
- Call stack e funzioni ricorsive
- Allocazione dinamica della memoria

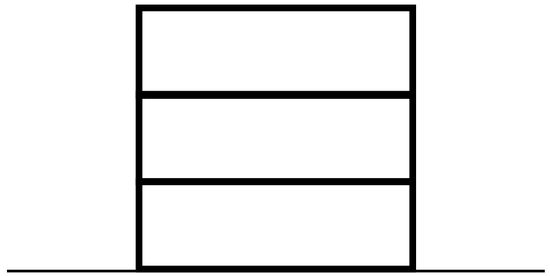
Strutture dati

Array



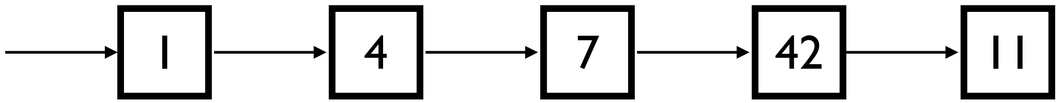
Accesso diretto

Stack

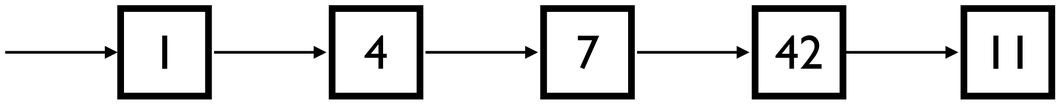


Accesso LIFO

Liste (*linked lists*)

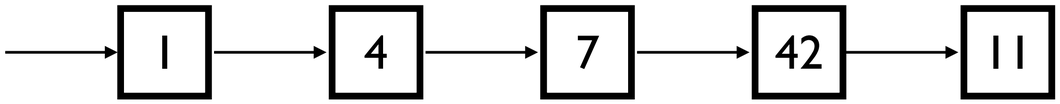


Liste (*linked lists*)



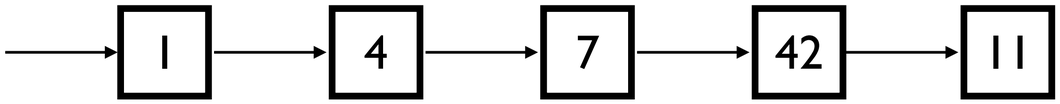
- Ogni elemento è *collegato* al suo successore

Liste (*linked lists*)



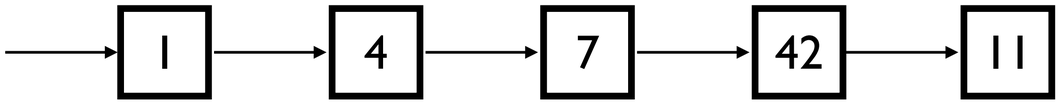
- Ogni elemento è *collegato* al suo successore
- Struttura dati dinamica: la sua dimensione può crescere e diminuire a run-time (a differenza degli array)

Liste (*linked lists*)



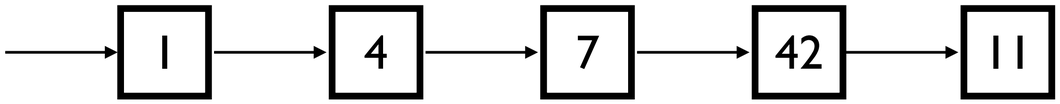
- Ogni elemento è *collegato* al suo successore
- Struttura dati dinamica: la sua dimensione può crescere e diminuire a run-time (a differenza degli array)
- Ogni elemento della lista contiene

Liste (*linked lists*)



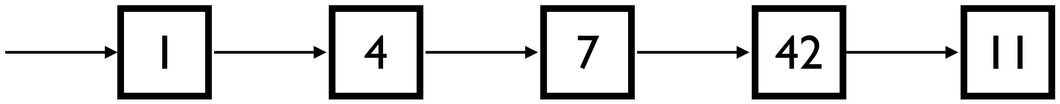
- Ogni elemento è *collegato* al suo successore
- Struttura dati dinamica: la sua dimensione può crescere e diminuire a run-time (a differenza degli array)
- Ogni elemento della lista contiene
  - Dato (es. un intero)

Liste (*linked lists*)



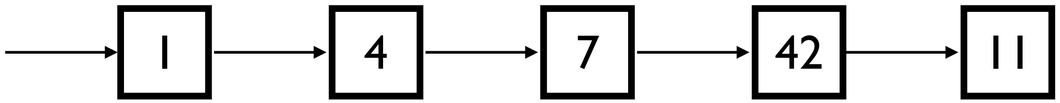
- Ogni elemento è *collegato* al suo successore
- Struttura dati dinamica: la sua dimensione può crescere e diminuire a run-time (a differenza degli array)
- Ogni elemento della lista contiene
  - Dato (es. un intero)
  - Un puntatore all'elemento successivo

Liste (*linked lists*)

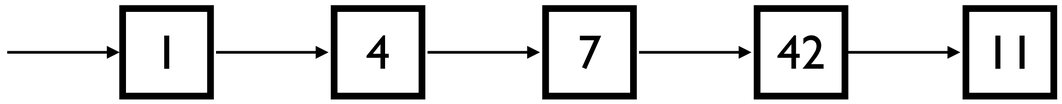


- Ogni elemento è *collegato* al suo successore
- Struttura dati dinamica: la sua dimensione può crescere e diminuire a run-time (a differenza degli array)
- Ogni elemento della lista contiene
  - Dato (es. un intero)
  - Un puntatore all'elemento successivo
- Accesso **sequenziale**

Liste (*linked lists*)

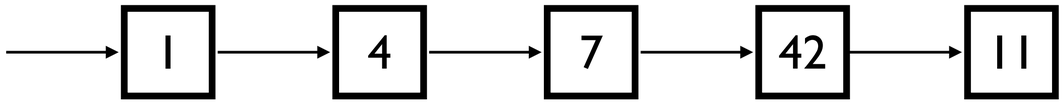


Liste (*linked lists*)

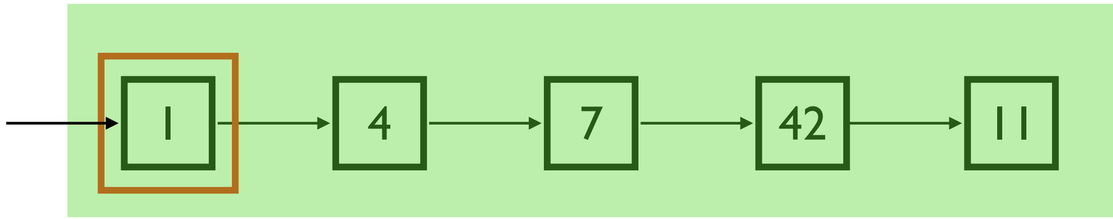


- Formalmente, una **lista** è:
  - (Caso base) Una lista vuota (zero nodi)
  - (Caso ricorsivo) un nodo seguito da una **lista**

Liste (*linked lists*)

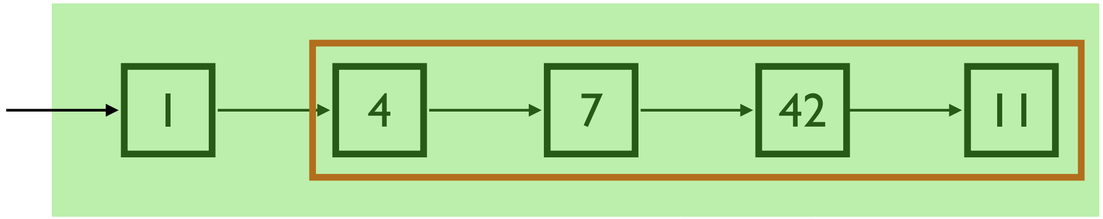


Liste (*linked lists*)

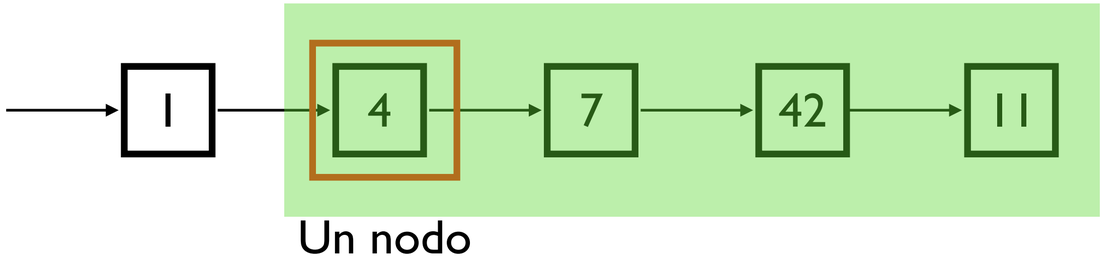


Un nodo

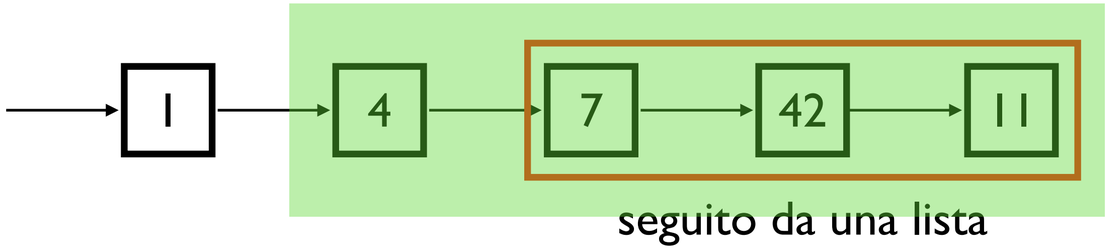
Liste (*linked lists*)



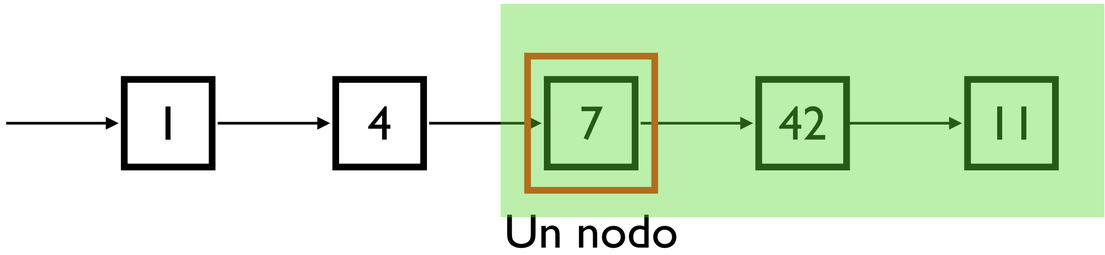
Liste (*linked lists*)



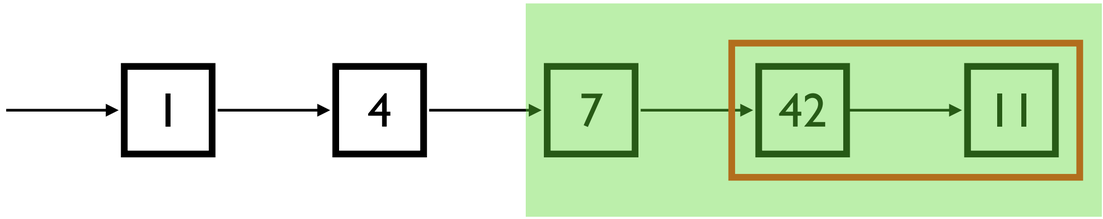
Liste (*linked lists*)



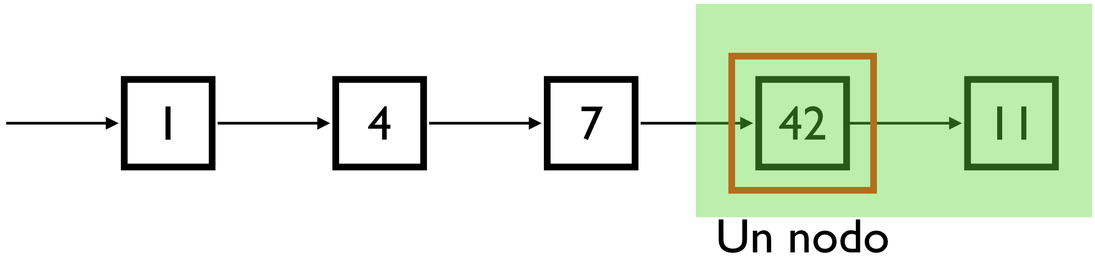
Liste (*linked lists*)



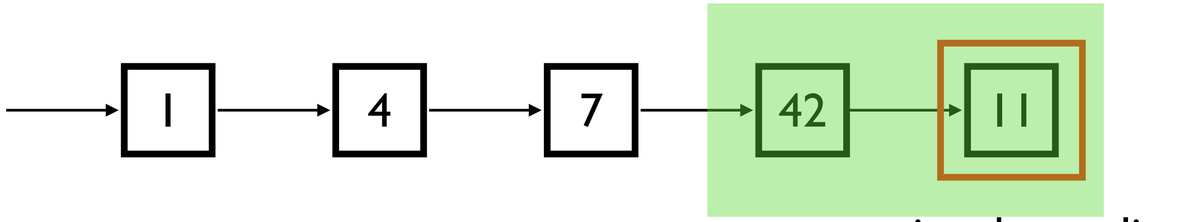
Liste (*linked lists*)



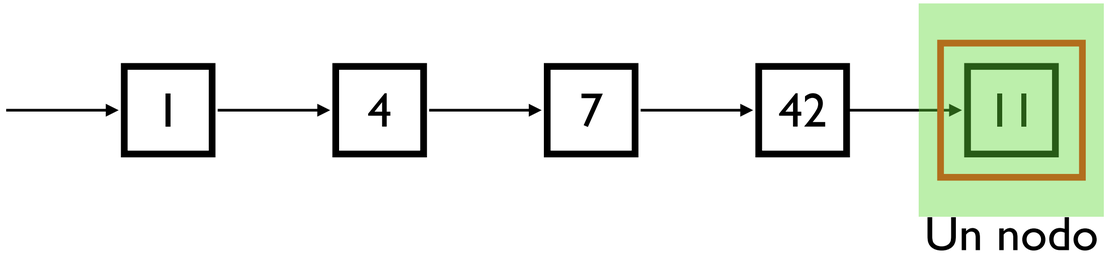
Liste (*linked lists*)



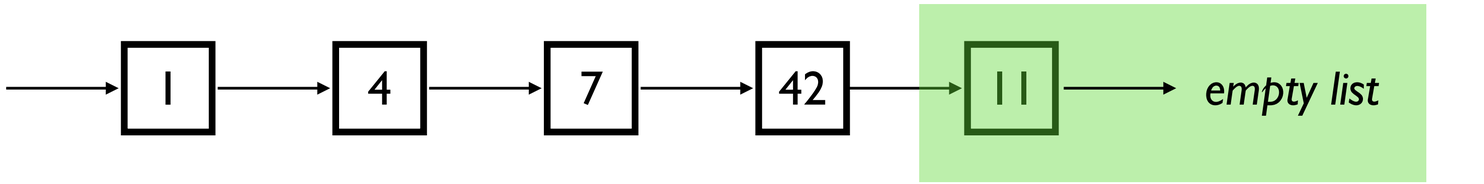
Liste (*linked lists*)



Liste (*linked lists*)

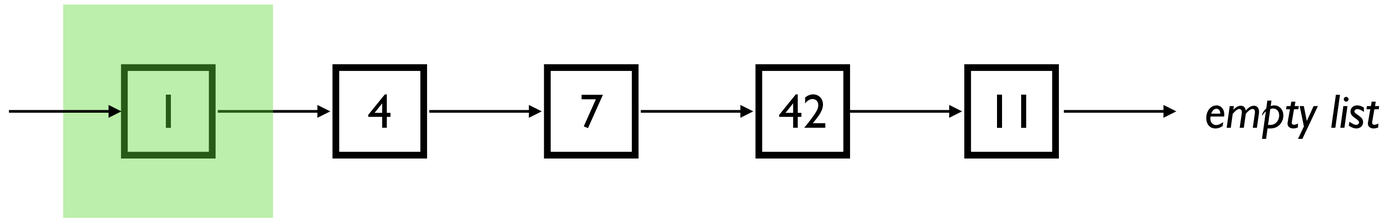


Liste (*linked lists*)



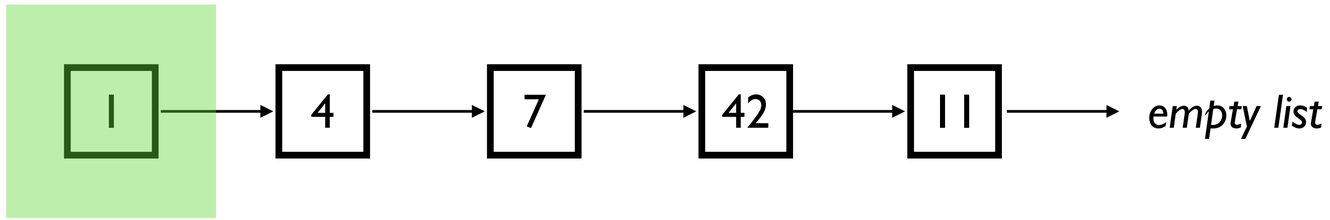
seguito da una lista

Nodo



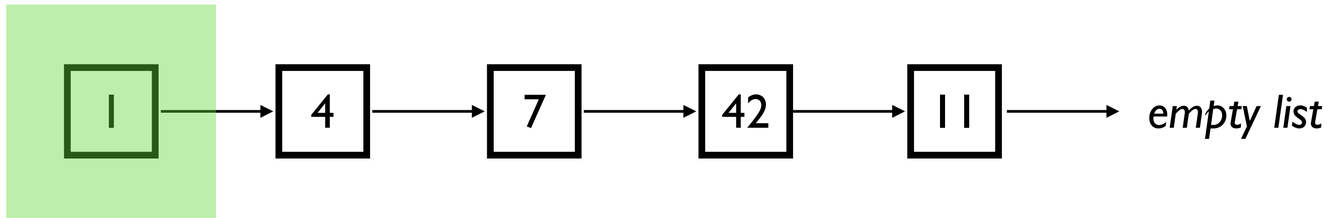

```
node :
```

Nodo



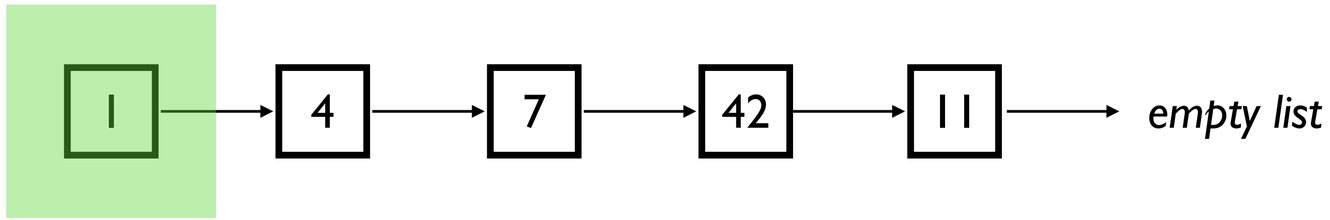

```
node :
    int data;
```

Nodo




```
node :
    int data;
    node * next;
```

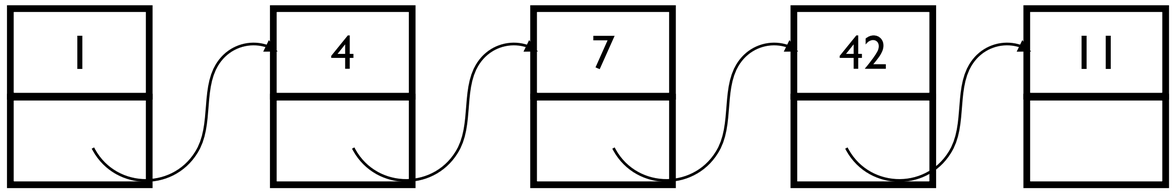
Nodo




```
node :
    int data;
    node * next;
```

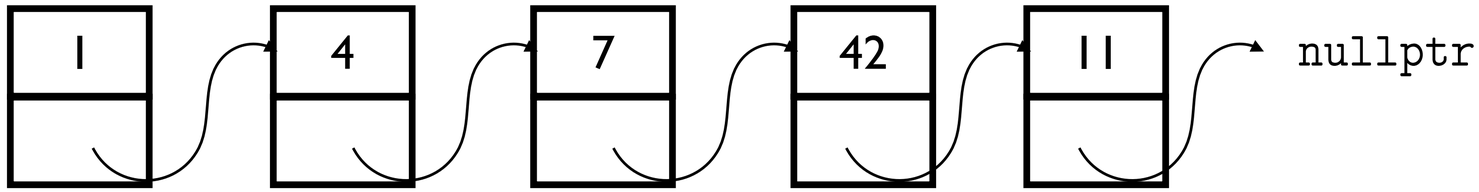
Nodo

```
node :
    int data;
    node * next;
```



Nodo

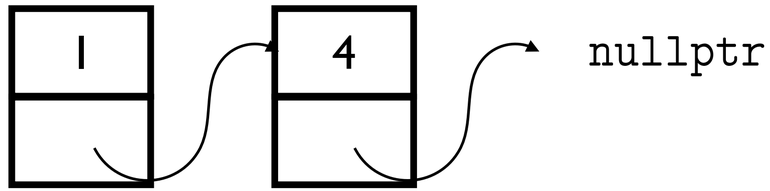
```
node :
    int data;
    node * next;
```



Liste

Esercizio

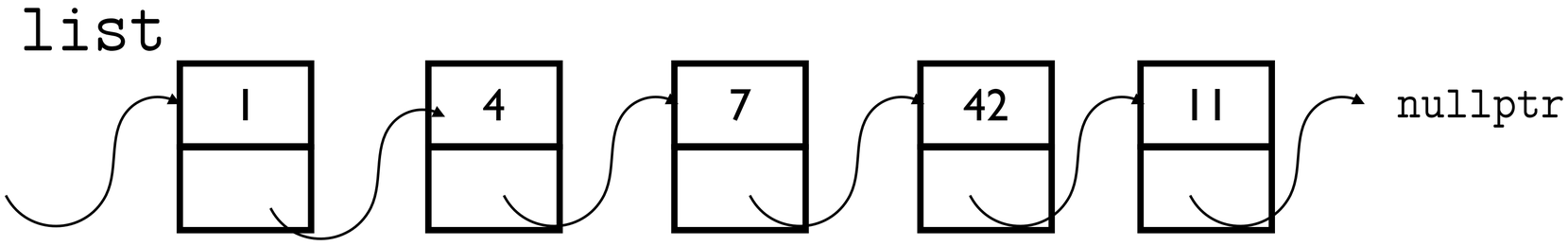
- Creare la lista nell'esempio



Liste

Esercizio

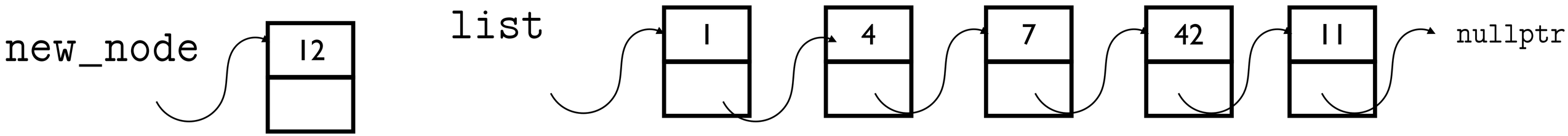
- Aggiungere un nodo in testa



Liste

Esercizio

- Aggiungere un nodo in testa

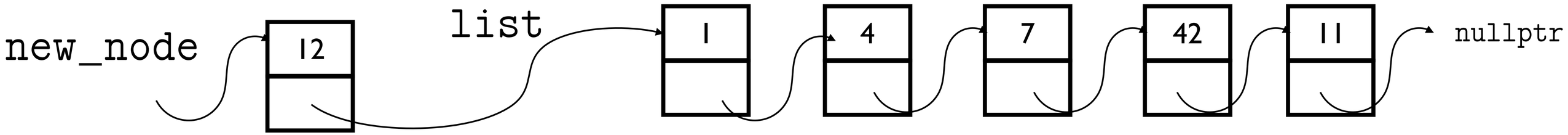


- Creo il nodo

Liste

Esercizio

- Aggiungere un nodo in testa

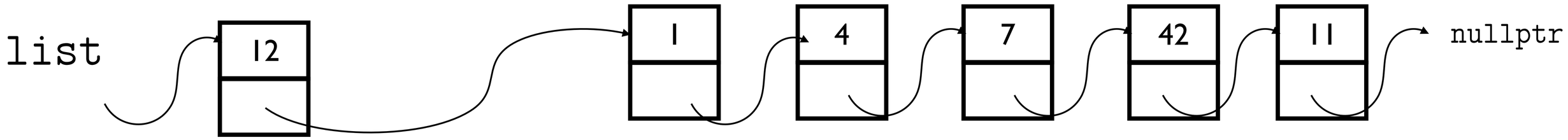


- Creo il nodo
- `next` del nuovo nodo deve puntare alla lista (`new_node -> current = list`)

Liste

Esercizio

- Aggiungere un nodo in testa

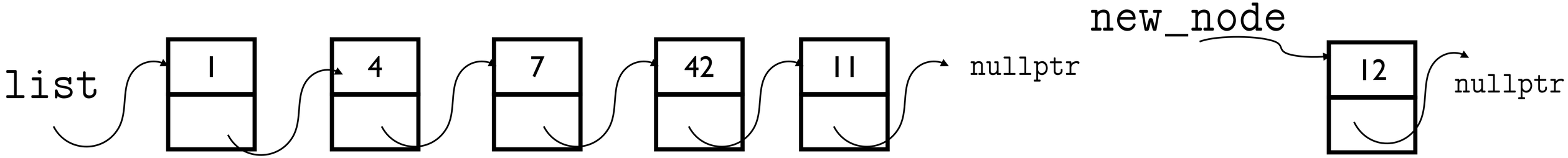


- Creo il nodo
- `next` del nuovo nodo deve puntare alla lista (`new_node -> current = list`)
- `list` diventa `new_node`

Liste

Esercizio

- Aggiungere un nodo in coda

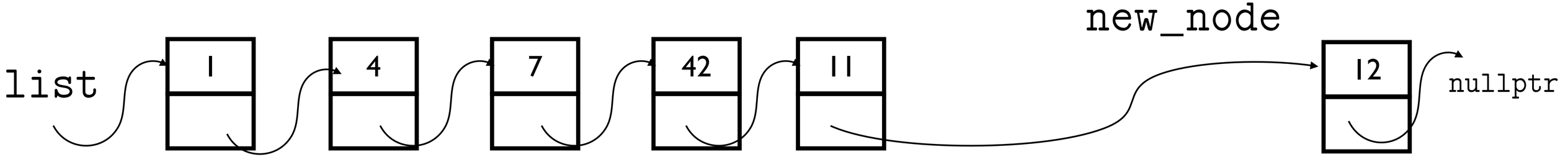


- Creo il nodo

Liste

Esercizio

- Aggiungere un nodo in coda

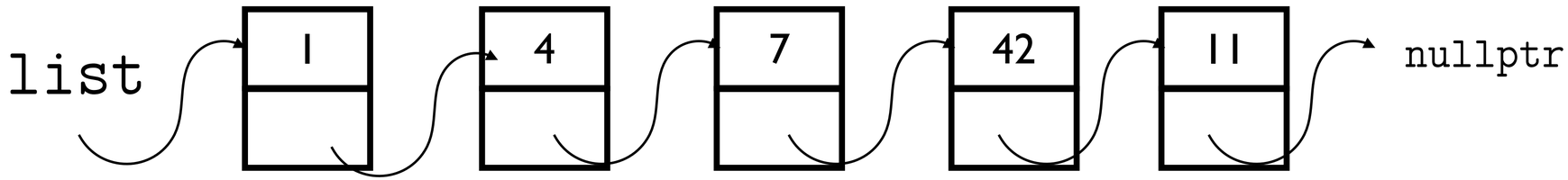


- Creo il nodo
- `next` del nuovo nodo deve puntare a `nullptr` (`new_node -> next = nullptr`)
- `next` dell'ultimo nodo deve puntare a `new_node`

Liste

Esercizio

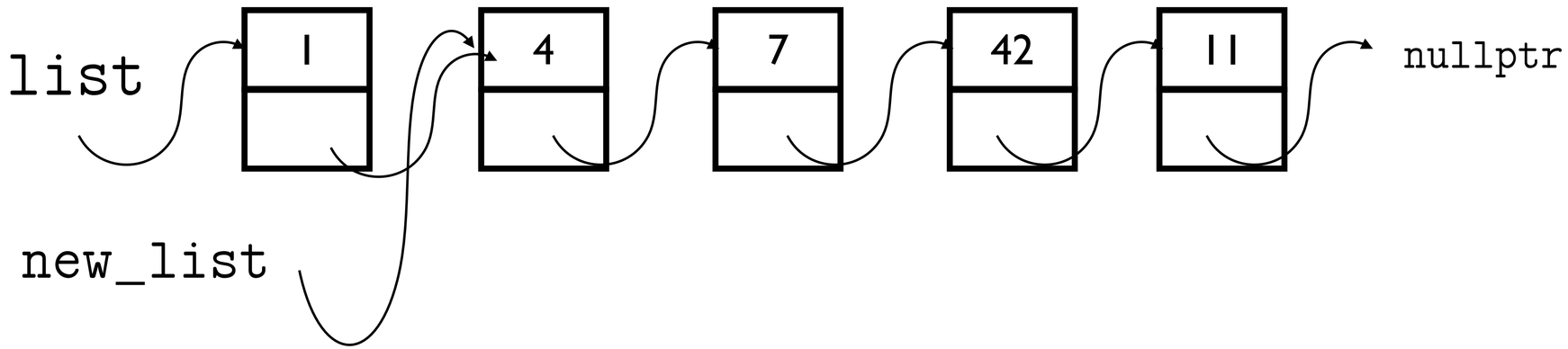
- Rimuovere il primo elemento della lista



Liste

Esercizio

- Rimuovere il primo elemento della lista

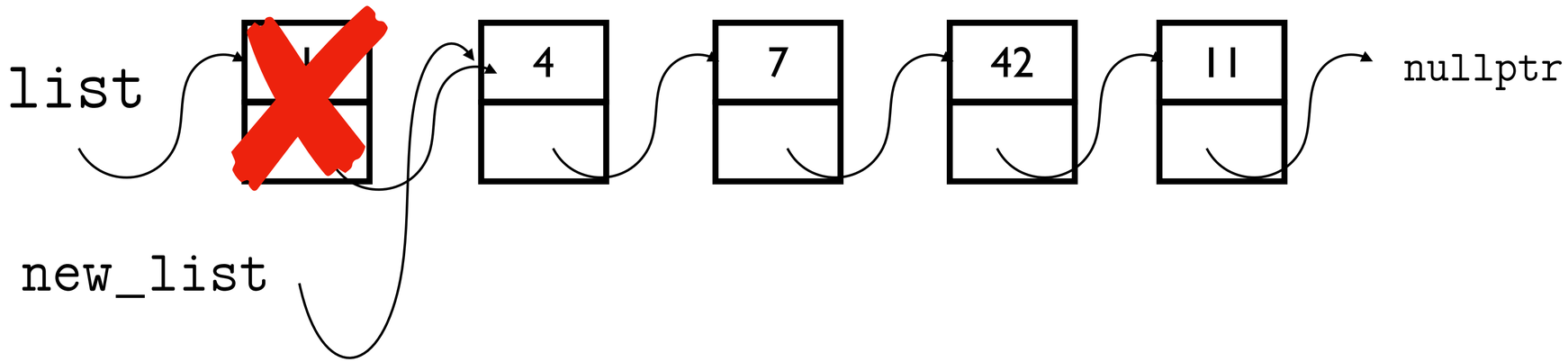


- Prendo `next` del primo elemento della lista (e.g. `new_list`)

Liste

Esercizio

- Rimuovere il primo elemento della lista

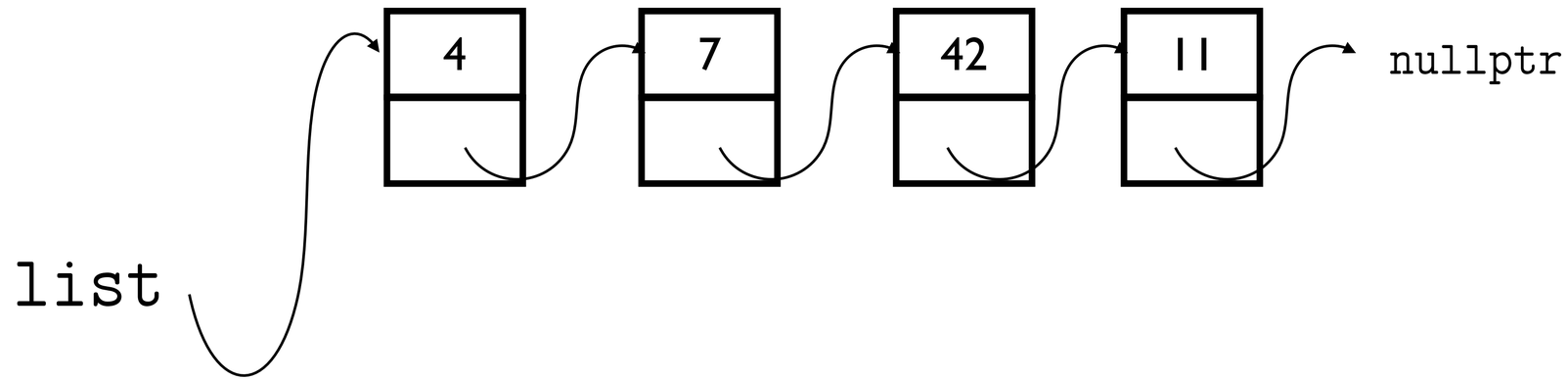


- Prendo `next` del primo elemento della lista (e.g. `new_list`)
- Dealloco la memoria dedicata al primo nodo (`delete list`)

Liste

Esercizio

- Rimuovere il primo elemento della lista

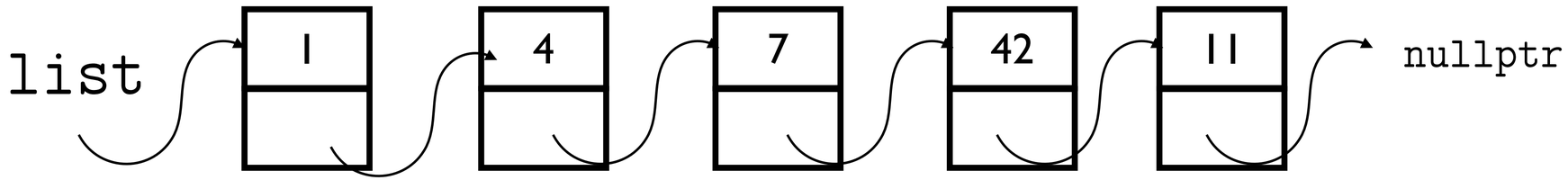


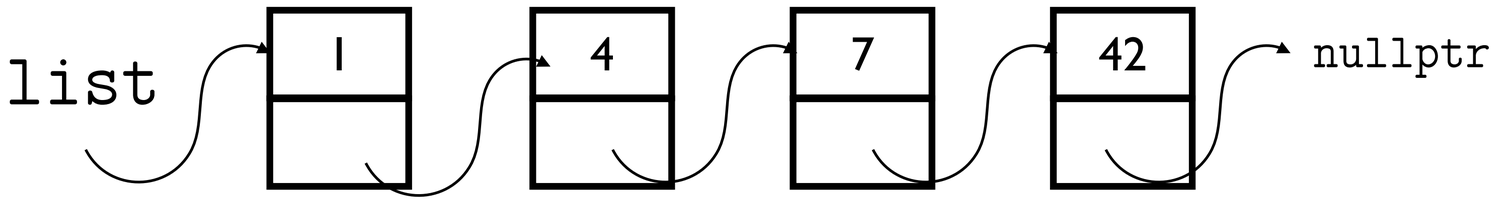
- Prendo `next` del primo elemento della lista (e.g. `new_list`)
- Dealloco la memoria dedicata al primo nodo (`delete list`)
- `list` diventa `new_list` (`list = new_list`)

Liste

Esercizio

- Rimuovere l'ultimo elemento della lista

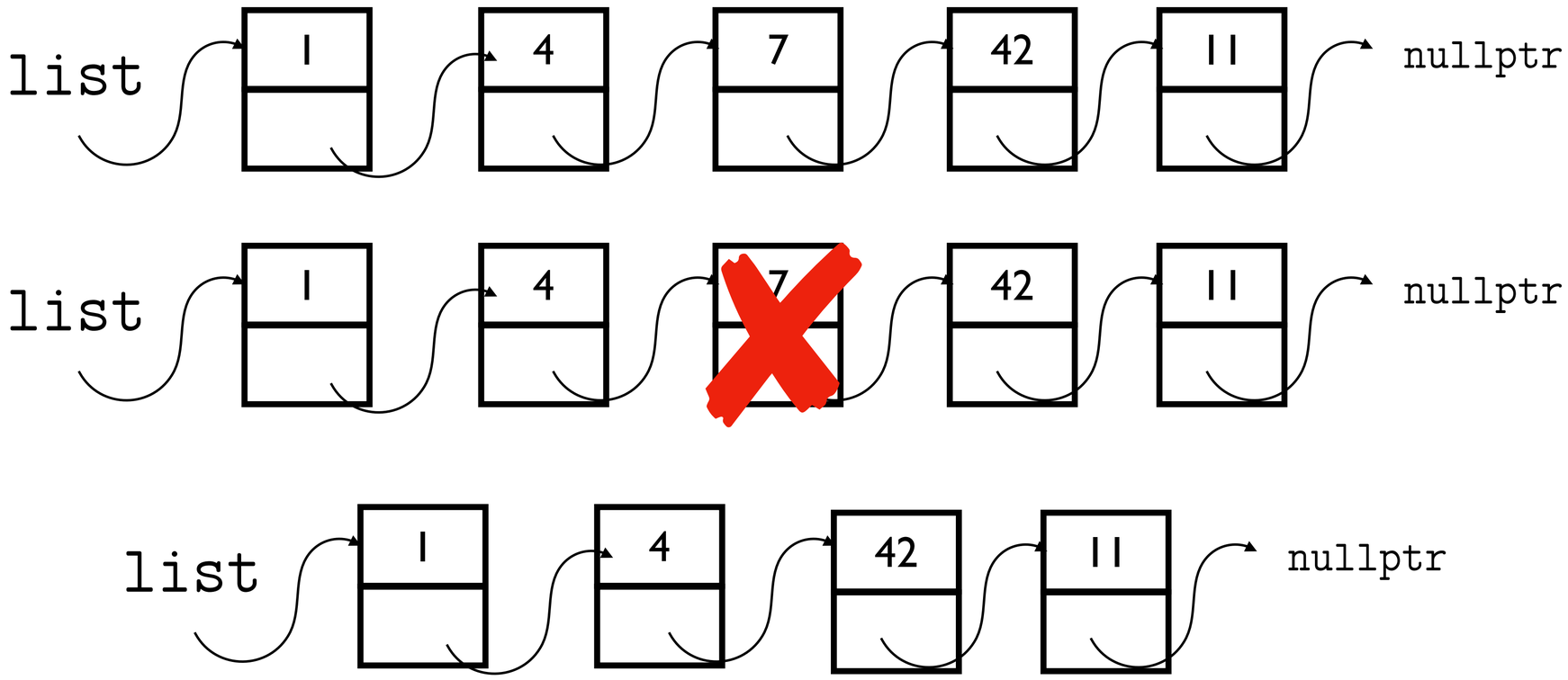




Liste

Esercizio

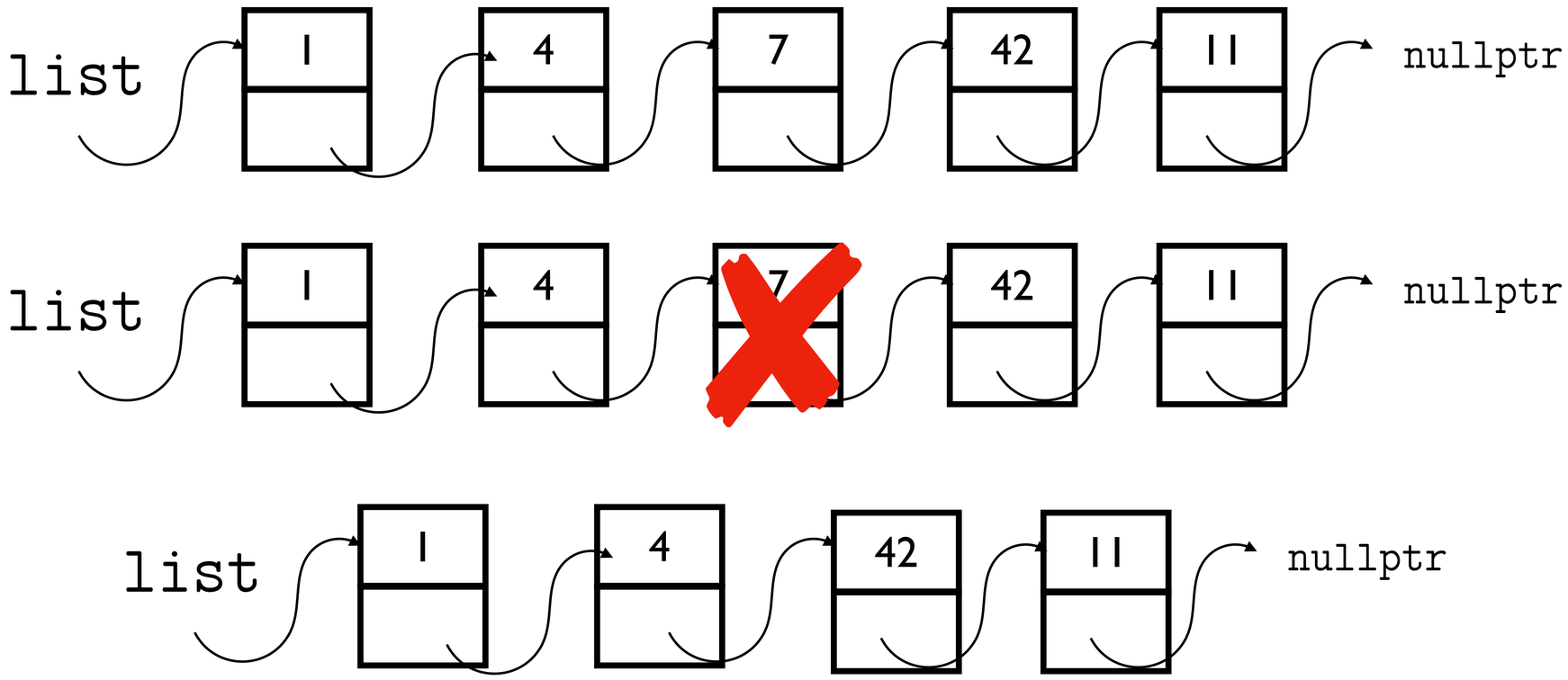
- Rimuovere un elemento della lista



Liste

Esercizio

- Rimuovere un elemento della lista (**ricorsivamente**)



Liste

Esercizi

- Scrivere una funzione che prende in input una lista e calcoli la sua lunghezza (versione **iterativa** e **ricorsiva**)
- Scrivere una funzione che prende in input una lista e calcoli la somma dei suoi valori (versione **iterativa** e **ricorsiva**)
- Scrivere una funzione che prende in input una lista e calcoli il massimo valore della lista (versione **iterativa** e **ricorsiva**)
- Scrivere una funzione che prende in input due liste e controlla se sono uguali (versione **iterative** e **ricorsiva**)
- Scrivere una funzione che prende in input una lista e controlla se la lista è monotona crescente (versione **iterativa** e **ricorsiva**)
- Scrivere una funzione che prende in input una lista e un indice i, e rimuova dalla lista l'elemento in posizione i (versione **iterativa** e **ricorsiva**)