

Lezione 18

Introduzione alle liste

Strutture dati dinamiche

- Vi sono problemi risolvibili efficacemente mediante algoritmi che fanno uso di **strutture dati dinamiche**
 - Ossia che strutture dati cambiano dimensione durante l'esecuzione dell'algoritmo

Problema

- Supponiamo di dover memorizzare e ristampare una successione di valori il cui numero non sia noto a priori
- Supponiamo inoltre che, oltre ad inserirli, sia necessario di tanto in tanto estrarre alcuni elementi

Array dinamico 1/2

- Possibile soluzione: array dinamico riallocato ogni volta che si renda necessario
- Ogni riallocazione ha costo $O(N)$
 - Bisogna ricopiare tutti i valori nella nuova locazione
- Comunque si fa “ogni tanto”, per cui l'inserimento ha costo *ammortizzato* $O(1)$

Array dinamico 2/2

- Però ad ogni estrazione di un elemento che non sia l'ultimo bisogna ricompattare l'array se non si vogliono lasciare 'buchi'
- Questo costa $O(N)$ **tutte le volte**

- Vi viene in mente una soluzione migliore?
- In merito, considerate che, anche se non abbiamo visto come, con l'operatore **new** si può anche allocare un solo oggetto e non necessariamente un array di oggetti

- Perché ogni volta che dobbiamo aggiungere un elemento non lo allochiamo in memoria **da solo**?
- Se e quando dobbiamo estrarlo lo deallochiamo, di nuovo **da solo**

- Dove memorizziamo l'indirizzo dei vari elementi?
- Cominciamo dal primo ...

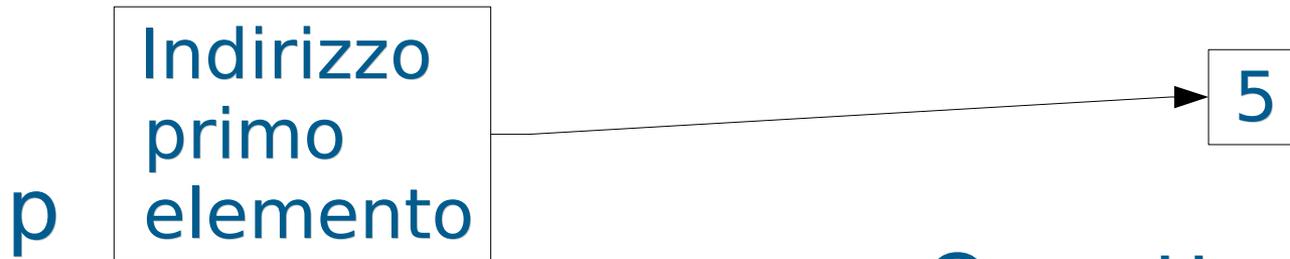
Puntatore al primo elemento

- Potremmo memorizzare in una variabile di tipo puntatore l'indirizzo di tale elemento

Puntatore al primo elemento

- Supponiamo che il primo valore sia 5
 - Allochiamo in memoria spazio per un intero e memorizziamo il valore
 - Ne memorizziamo l'indirizzo in una variabile p di tipo puntatore

Puntatore al primo elemento



Oggetto dinamico

Variabile locale o globale: oggetto automatico o statico

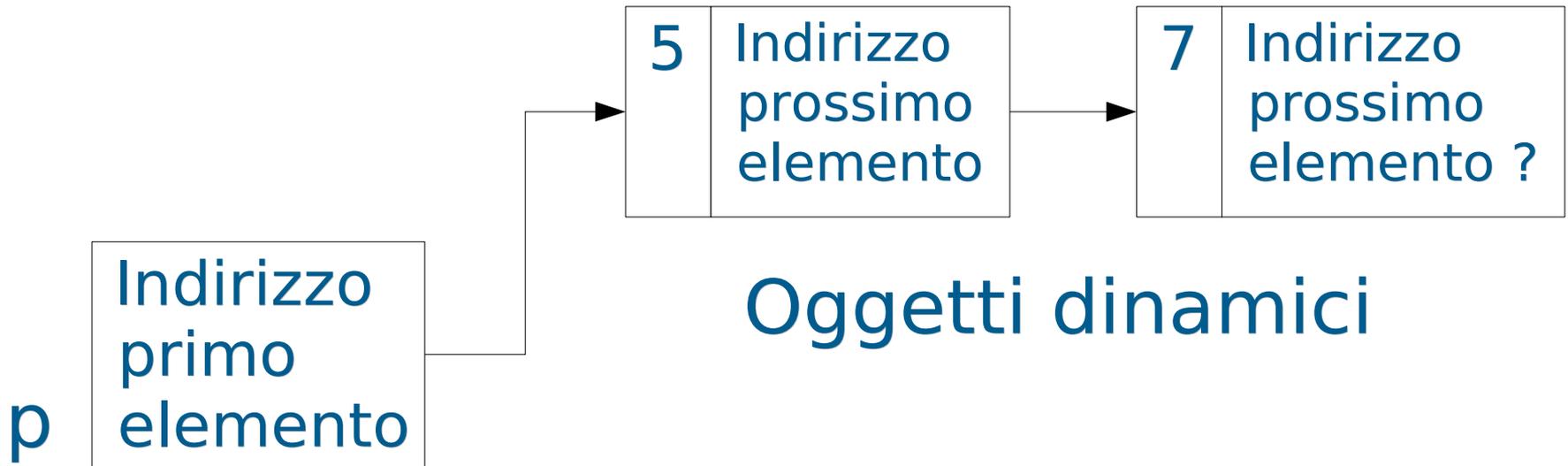
Elementi successivi

- Supponiamo di inserire un altro valore, diciamo 7
- Come facciamo per memorizzare l'indirizzo del secondo elemento, ed in generale l'indirizzo del prossimo elemento ogni volta che ne aggiungiamo uno?

Puntatore al successivo

- Per ciascun valore, potremmo allocare spazio in memoria
 - sia per il valore dell'elemento,
 - che per un puntatore che punti al prossimo elemento
- Così, una volta raggiunto un elemento, abbiamo le informazioni necessarie per accedere al prossimo

Puntatore al successivo



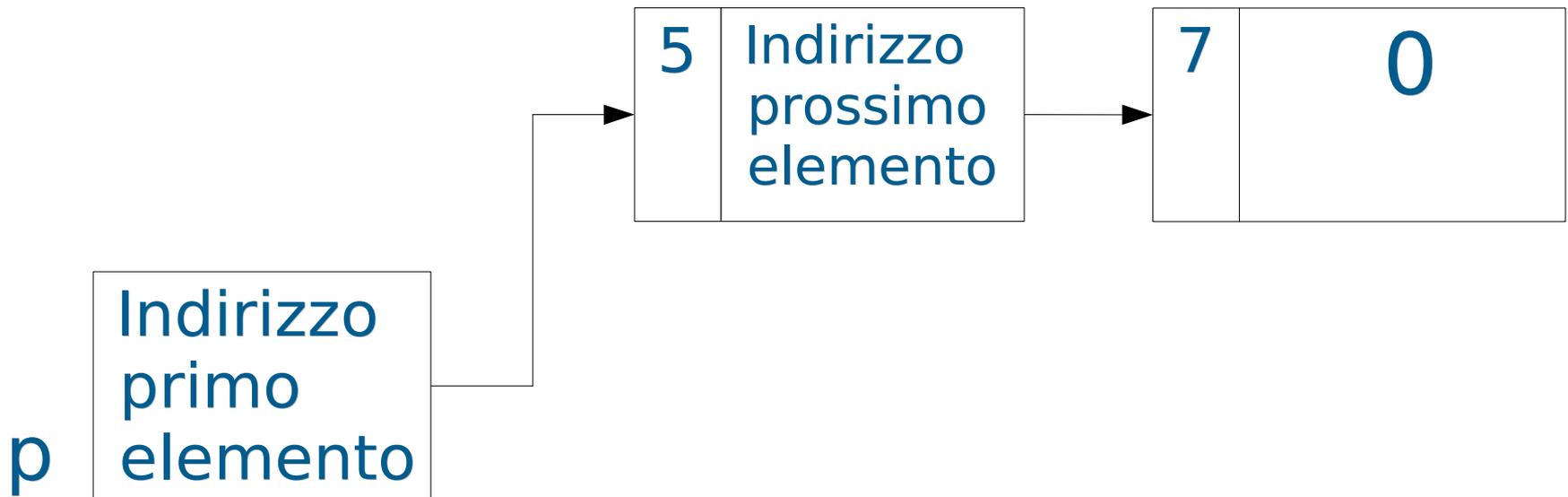
Variabile locale o globale: oggetto automatico o statico

Ultimo elemento 1/2

- L'elemento contenente il valore 7 è attualmente l'ultimo (ce ne sono solo due)
- Cosa conterrà il puntatore all'interno della struttura che lo rappresenta?
- Come facciamo a dire che non ci sono altri elementi dopo di lui?

Ultimo elemento 2/2

- Possiamo assegnargli il valore 0 (NULL)



- Abbiamo costruito un oggetto di tipo **lista concatenata**

Lista concatenata

- Struttura dati i cui oggetti/elementi sono disposti in ordine lineare
- Diversamente dall'array, in cui l'ordine è determinato dagli indici, l'ordine in una lista concatenata è determinato da un puntatore in ogni oggetto

Terminologia 1/2

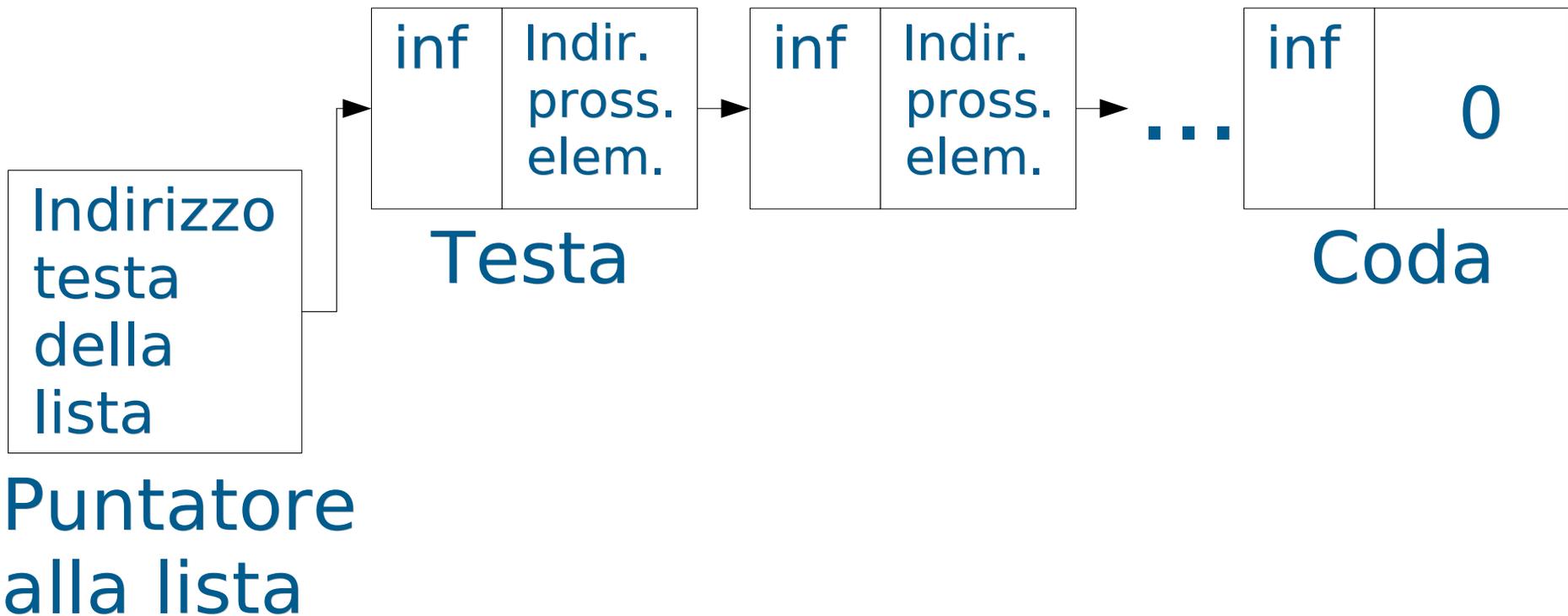
- Diremo che ciascun elemento contiene un **campo informazione** ed un campo puntatore (oppure due, come stiamo per vedere)
- Il primo elemento di una lista è tipicamente chiamato **testa (head)** della lista
- L'ultimo elemento è tipicamente chiamato **coda (tail)** della lista

Terminologia 2/2

- Lista singolarmente concatenata o semplice: ciascun elemento contiene solo un puntatore al prossimo elemento
- Lista doppiamente concatenata o doppia: ciascun elemento contiene sia un puntatore al prossimo elemento che un puntatore all'elemento precedente

Lista semplice 1/2

- Ciascun elemento contiene solo un puntatore al prossimo elemento



Lista semplice 2/2

- Il puntatore al prossimo elemento della coda della lista contiene il valore 0 (NULL)
- Il puntatore alla testa della lista individua la lista stessa
 - E' perciò chiamato anche puntatore alla lista

Tipo di dato lista 1/2

- Esistono varie librerie che forniscono il tipo di dato lista
- Vengono fornite le operazioni di
 - Creazione ed eliminazione
 - Inserimento/estrazione di elementi in testa, in fondo, in una posizione data
 - Tipicamente di costo $O(1)$
 - Restituzione del numero di elementi
 - Attenzione, in alcune implementazioni costa $O(1)$ mentre in altre $O(N)$!

Tipo di dato lista 2/2

- Inserimento in ordine
 - Tipicamente a costo $O(N)$ (per via della ricerca della posizione)
- Riordinamento
 - Tipicamente a costo $O(N \log N)$
- Le funzioni di libreria si occupano dei puntatori, il programmatore di preoccupa solo del campo informazione
- Ad esempio, nella libreria standard del C++ (non in quella del C) c'è il tipo di dato *list*, presentato in `<list>`

Fine del corso

- Con quest'ultima *slide* si chiude il corso
- Spero di essere riuscito a comunicarvi il messaggio forse più importante per un insegnamento di introduzione alla programmazione
 - Applicare il massimo rigore nelle fasi di sviluppo
 - Per valorizzare al massimo uno dei momenti più belli dell'attività di programmazione: la nascita di una nostra **idea nuova**, che ci fa risolvere un problema che prima non sapevamo risolvere

- Vi aspetto all'esame ...

... per il quale vi lascio il mio “in bocca al lupo”