Programmazione e Strutture Dati (PR&SD)

I° ANNO – Informatica

Prof. V. Fuccella

ADT Lista
Operatori Aggiuntivi

ADT: Lista

Sintattica	Semantica
Nome del tipo: List Tipi usati: Item, boolean	Dominio: insieme di sequenze $L=a_1,\ldots,a_n$ di tipo Item L'elemento **nil** rappresenta la lista vuota
newList() → List	newList() → l • Post: l = nil
isEmpty(List) → boolean	isEmpty(l) → b • Post: se l=nil allora b = true altrimenti b = false
addHead(List, Item) → List	addHead(l, e) → l' • Post: l = <a1, a2, … an> AND l' = <e, a1, …, an>
removeHead(List) → List	removeHead(l) → l' • Pre: l = <a1, a2, …, an> n>0 • Post: l' = <a2, …, an>
getFirst(List) → Item	getFirst(l) → e • Pre: l = <a1, a2, …, an> n>0 • Post: e = a1

ADT: Lista – Operatori aggiuntivi

Sintattica	Semantica
Nome del tipo: List Tipi usati: Item, boolean	Dominio: insieme di sequenze $L=a_1,\ldots,a_n$ di tipo Item L'elemento **nil** rappresenta la lista vuota
searchItem(List, Item) → int	searchItem(l, i) → pos • Post: se i in l allora pos = pos. di i in l else pos = -1
removeItem(List, Item) → List	removeItem(l, e) → l' • Pre: l = <a1, a2, …, e, …, an> n>0 • Post: l' = l - <e>
removeItem(List, int) → List	removeItem(l, pos) → l' • Pre: l = <a1, a2, …, a_{pos}, …, an> 1 <= pos <= n • Post: l' = l - <a_{pos}>

searchItem(l, i)
Progettazione

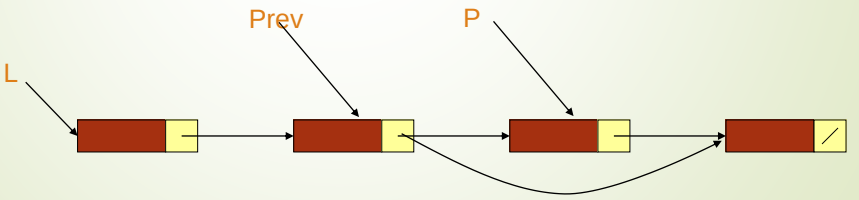
- Dati una lista l e un elemento val, restituisce:
  - La posizione della lista in cui appare la prima occorrenza dell'elemento
  - Oppure -1 se l'elemento non è presente
- Richiede una ***visita finalizzata*** della lista: usciamo da ciclo quando troviamo l'elemento cercato oppure quando raggiungiamo la fine della lista
- Possiamo ottenere sia il riferimento all'Item ricercato, sia la sua posizione, implementando la funzione col seguente prototipo

```
Item searchItem(List list, Item item, int *pos)
```

Eliminazione
Progettazione

Per eliminare un elemento (p.e. il terzo) in una lista concatenata L:

1. Si fanno avanzare due puntatori Prev e P fino a che P punta al nodo da eliminare;
2. Si aggiorna il nodo puntato da Prev, facendolo puntare al successivo di P
3. Si elimina il nodo puntato da P, liberando la memoria



Programmazione e Strutture Dati (PR&SD)

I° ANNO – Informatica

Prof. V. Fuccella

ADT Lista
Operatori Aggiuntivi (2)

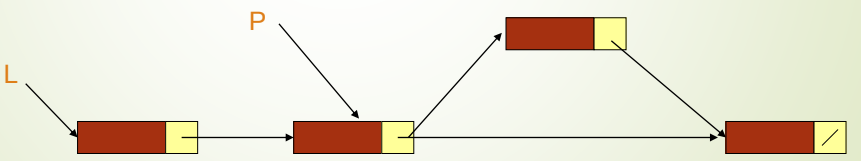
ADT: Lista – Operatori aggiuntivi (2)

Sintattica	Semantica
Nome del tipo: List Tipi usati: Item, boolean	Dominio: insieme di sequenze $L=a_1,\ldots,a_n$ di tipo Item L'elemento **nil** rappresenta la lista vuota
insertItem(List, Item, int) → List	insertItem(l, e, pos) → l' • Pre: l = <a1, …, an> & 1 <= pos <= n+1 • Post: l' = <a1, …, a_{pos}, …, a_{n+1}> & a_{pos}=e
insertTail(List, Item) → List	insertTail(l, e) → l' • Post: l = <a1, …, an> & l' = <a1, …, an, e>
reverseList(List) → List	reverseList(l) → l' • Post: l = <a1, a2, …, an> AND l' = <an, …, a2, a1>
cloneList(List) → List	cloneList(l) → l' • Post: l = <a1, a2, …, an> AND l' = <a1, a2, …, an>

Inserimento
Progettazione

Per inserire un elemento (p.e. in terza posizione) in una lista concatenata L:

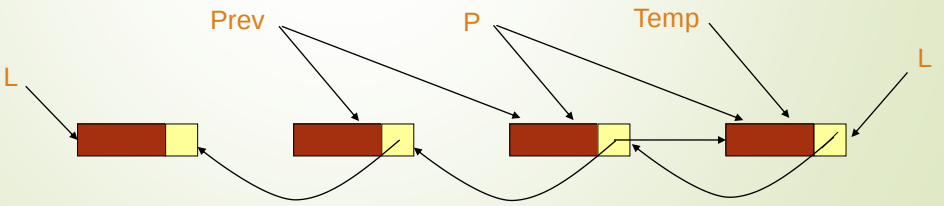
1. Si fa avanzare un puntatore P, fino a che punta al nodo precedente alla posizione dell'inserimento;
2. Si crea il nuovo nodo e lo si fa puntare al successivo di P;
3. Si fa puntare P al nuovo nodo



Reverse
Progettazione

Per invertire una lista concatenata L:

1. **Per ogni nodo** (con i nodi fino a Prev già invertiti) si utilizzano due puntatori Prev e P;
   a) Si salva il successivo di P in un puntatore temporaneo Temp
   b) Si aggiorna il nodo puntato da P, facendolo puntare a Prev
   c) Si fanno avanzare P e Prev
2. Si aggiorna la testa facendola puntare all'ultimo nodo

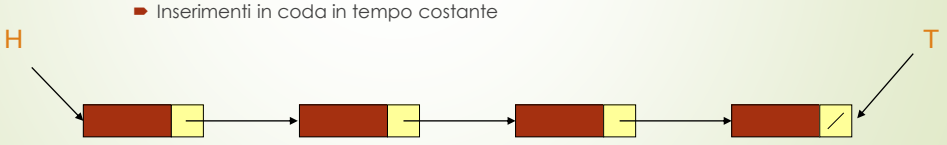


CloneList
Progettazione

- Diverse scelte progettuali possibili:
  - Clonare solo la struct list:
    - I nodi sono gli stessi. Una modifica su una lista viene riflessa nell'altra
  - Clonare i nodi ma non gli item
    - Modifiche alla struttura di una lista non vengono riflesse nell'altra, ma se si modifica un item, risulta modificato in entrambe le liste
  - Clonare i nodi e gli item
    - Le liste, una volta clonate sono totalmente indipendenti

Liste
Implementazioni alternative

- Array: vantaggi e svantaggi
- Liste a collegamento singolo
  - Con puntatore **head**: le operazioni insertTail sono particolarmente inefficienti (bisogna scorrere tutta la lista, numero di operazioni lineare rispetto alla taglia della lista)
  - Con puntatori **head** e **tail**
    - Inserimenti in coda in tempo costante



Liste
Implementazioni alternative

- Liste circolari (ultimo nodo collegato al primo)
  - Permettono l'attraversamento della lista partendo da un nodo qualsiasi
- Liste a collegamento doppio
  - Svantaggio: operazioni di inserimento e cancellazione più complicate a causa della necessità di aggiornare due puntatori
  - Vantaggio: Cancellazione e inserimento in tempo costante quando si ha il puntatore al nodo
- Combinazioni delle soluzioni viste finora: es. liste doppie con tail o circolari