Programmazione e Strutture Dati (PR&SD)

I° ANNO – Informatica
Prof. V. Fuccella

ADT Lista

Il tipo astratto *Lista*

- **Definizione**: Sequenza di elementi di un determinato tipo, in cui è possibile aggiungere o togliere elementi.
  - È possibile specificare la *posizione* relativa nella quale l'elemento va aggiunto o tolto.

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>
getHead(List) → Item	getHead(l) → e • Pre: l = <a1, a2, …, an> n>0 • Post: e = a1

ADT Lista
Progettazione: Liste Concatenate

- Ogni elemento di una lista concatenata è un record con un campo puntatore che serve da collegamento per il record successivo.

- Si accede alla struttura attraverso il puntatore al primo record.
- Il campo puntatore dell'ultimo record contiene il valore NULL

Liste concatenate VS Array

- Array: dimensione fissa, accesso *diretto* ad *ogni* elemento (con indice)
- Lista: *dimensione variabile, accesso* diretto solo al ***primo*** elemento della lista.
  - Per accedere ad un generico elemento, occorre ***scandire sequenzialmente*** gli elementi della lista.

Liste concatenate VS Array

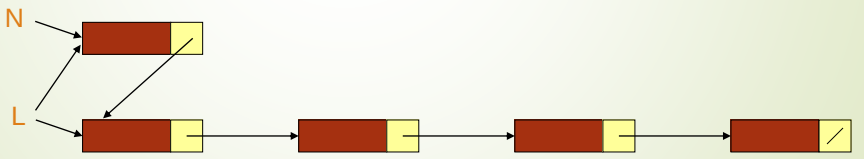
Efficienza computazionale

- Accesso
  - Lista concatenata: per accedere all'*i*-esimo elemento occorre scorrere la lista dal primo all'*i*-esimo elemento (tempo max proporzionale ad *n*)
  - Array: ogni elemento di un array è accessibile direttamente usando il suo indice (tempo costante)
- Inserimento e cancellazione
  - Lista concatenata: dato un elemento, è possibile eliminarlo o aggiungerne uno dopo direttamente (tempo costante)
  - Array: Occorre effettuare delle operazioni di shift (tempo max proporzionale ad *n*)

Lista concatenata
Progettazione: Inserimento in testa

Il modo più semplice per inserire un nuovo elemento in una lista concatenata L è inserire l'elemento in un nuovo nodo da aggiungere in testa alla lista

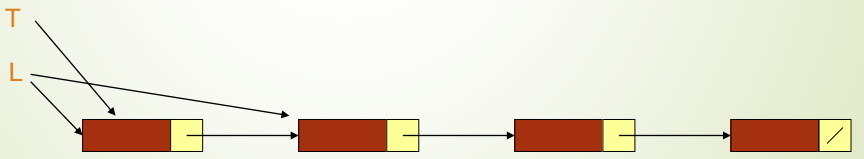
1. si alloca il nuovo nodo N
2. si aggiunge il collegamento con il record iniziale della lista
3. Si aggiorna L facendolo puntare a N



Lista concatenata
Progettazione: Eliminazione in testa

Il modo più semplice per eliminare un elemento in una lista concatenata L è eliminarlo in testa alla lista

1. Si crea un puntatore temporaneo T, copia di L
2. Si aggiorna L facendolo puntare al successivo di L
3. Si elimina il nodo puntato da T, liberando la memoria



Lista concatenata
Progettazione: Visita di una lista

- Alcuni operatori richiedono una ***visita*** parziale o totale della lista.
- Esempi di visita totale:
  - Calcolo della size;
  - Stampa degli elementi;
- Esempi di visita parziale:
  - Inserimento o rimozione in una posizione i;
  - Ricerca di un elemento
- Calcolo della size: scorriamo i nodi aggiungendo incrementando un contatore
  - Ottimizzazione: mantenere un contatore da incrementare ad ogni inserimento e decrementare ad ogni cancellazione

… e adesso un po' di codice C

Implementazione
Tipo Lista

- Per prima cosa occorre dichiarare il tipo lista

```
typedef struct list *List;
struct list{
 int size;
 struct node *head;
};
```

- Per istanziare la lista occorre riservare memoria e inizializzare la lista vuota

```
List list = malloc(sizeof(struct list));
list->size = 0;
list->head = NULL;
```

Implementazione
Tipo nodo

- Per usare una lista concatenata serve una struttura che rappresenti i nodi
- Si usa una struttura auto-referenziale:

```
struct node {
  Item value;            /* data stored in the node  */
  struct node *next;     /* pointer to the next node */
};
```

- La struttura conterrà i dati necessari ed un puntatore al prossimo elemento della lista
- Per iterare sui nodi si può usare un ciclo for:

```
for(p = list->head; p != NULL; p = p->next)
    outputItem(p->value);
```

Implementazione
Istanziare un nodo

- Man mano che costruiamo la lista, creiamo dei nuovi nodi da aggiungere alla lista
- I passi per creare un nodo sono:
  1. Allocare la memoria necessaria
  2. Memorizzare i dati nel nodo
  3. Inserire il nodo nella lista
- Vediamo i primi due passi per adesso

Implementazione
Tipo nodo

- Per creare un nodo ci serve un puntatore temporaneo che punti al nodo:

  ```
  struct node *new_node;
  ```

- Possiamo usare malloc per allocare la memoria necessaria e salvare l'indirizzo in new_node:

  ```
  new_node = malloc(sizeof(struct node));
  ```

- new_node adesso punta ad un blocco di memoria che contiene la struttura di tipo node: