POLITECNICO
MILANO 1863

Lab 4:

Liste

Recap teoria x liste

Per agevolarne la scrittura si usano le typedef, sia per la struttura,

sia per il suo puntatore.

```
typedef struct  EL {
     TipoElemento Info;
     struct EL * Prox;
} ElemLista;
typedef ElemLista * ListNodePtr;
```

Recap teoria x liste

La lista deve essere dichiarata nel MAIN, attraverso un puntatore alla struttura.
La lista all'inizio del programma "NON ESISTE", corrisponde a NULL. Va inizializzata e riempita.

Lo riempimento può essere fatto in testa (dal primo valore) o in coda (un append in fondo).

Recap teoria x liste con metodi ricorsivi

ES. Inserimento in coda:

Se la lista è vuota, inserisco subito il nodo; se non lo è, devo scorrere la lista fino in fondo

```
list tailInsert(list l, int content) {
  list currentTail = l;
  node *n;
  n = (node *) malloc(sizeof(node));
  n->next = NULL;
  n->content = content;
  // Caso di lista inizialmente vuota
  if (l == NULL) {
return n; }

  // Caso di lista non vuota
  while (currentTail->next != NULL) {
    currentTail = currentTail->next;
  }
  currentTail->next = n;
return l;

}
```

Recap teoria x liste con metodo ricorsivo

Con il metodo ricorsivo, è la **ricorsione** che scorre la lista

```
list tailInsert(list l, int content) {
  node *n;
  if (l == NULL) {
n = (node *) malloc(sizeof(node));
    n->next = NULL;
    n->content = content;
    return n;
  } else {
    l->next = tailInsert(l->next, content);
    return l;
} }
```

Esercizi e f. Gia viste a lezione

NOTA

Si faccia uso, per quanto piu possibile, delle f. viste a lezione, riassunte nel file:

deitel_print_list.c

Caricato su weebeep

Esercizi: NOTA

Alcuni esercizi andranno svolti con entrambi i metodi, per alcuni verrà richiesto di implementare **solo** versione ricorsiva

Note:

1) Per alcuni verrà GIA data la lista caricata.

2) Supponiamo di partire dal una struct simile a:

```
typedef int Data;
struct listNode {
    Data data; // each listNode contains a "data"
   struct listNode *nextPtr; // pointer to next node
};

typedef struct listNode ListNode; // synonym for struct listNode
typedef ListNode *ListNodePtr;    // synonym for ListNode*
```

Es warm-up

Si scrivano delle f. Di comune uso:

```
bool isEmpty(ListNodePtr head);

ListNodePtr last(ListNodePtr head);

ListNodePtr tail(ListNodePtr head);
```

(È IL "RESTO", la coda della lista tolta la testa)

Es 0 (SUPER Facile)

Dato un array di numeri, di cui è noto la numerosità, si costruisca una lista L COL SEGUENTE PROTOTIPO:

```
ListNodePtr buildFromArray_iterative(
   ListNodePtr head,
   int arr[],
   int n_elems);
```

Che costruisca uan lista di interi in modo Iterativo

NB: il primo elem dell' array deve essere la testa della lista

Es 0 B ricorsivo

Dato un array di numeri, di cui è noto la numerosità, si costruisca una lista L COL SEGUENTE PROTOTIPO:

```
ListNodePtr buildFromArray(
   ListNodePtr head,
   int arr[],
   int n_elems);
```

Che costruisca uan lista di interi in modo Ricorsivo

NB: il primo elem dell' array deve essere la testa della lista

Es 1

(Facile)

Dato un array di numeri, di cui è noto la numerosità, si costruisca una lista L.

Si scriva una funzione

int **somma(ListNode* head);**

*(Oppure.. int somma(**ListNodePtr** head);*

Che calcoli il totale degli elementi in modo

A) **iterativo**

B) **Ricorsivo**

Es 2

(Facile)

Utilizzando il codice iniziale dell'esercizio uno, si scriva la funzione

int **max(ListNodePtr head);**

Che cerchi il massimo degli elementi in modo

A) **iterativo**

B) **Ricorsivo**

(potete supporre che tutti i numeri siano interi positivi)

Es 3

(Facile)

Utilizzando il codice iniziale dell'esercizio uno, si scriva la funzione

```
bool isLast(ListNodePtr head);
```

Che restituisce true se H è ultimo elemento,

False negli altri casi.

A) iterativo (potrebbero servirvi dei parametri aggiuntivi)

B) Ricorsivo

Es 4

(Facile)

Utilizzando il codice iniziale dell'esercizio uno, si scriva la funzione

```
int stampa(ListNodePtr head, bool inverse);
```

Che stampa la lista dal primo all'ultimo se inverse

é **false**, dall' ultimo al primo se **true**

A) iterativo (potrebbero servirvi dei parametri aggiuntivi)

B) Ricorsivo

Es 5

Utilizzando il codice iniziale dell'esercizio uno, si scriva la funzione

```
ListNodePtr appendTo( ListNodePtr head,
TipoElemento v);
```

Appenda in coda il valore.

A) iterativo (potrebbero servirvi dei parametri aggiuntivi)

B) Ricorsivo

NB: deve funzionare ANCHE se si passa un head NULL

Es 6 ricorsivo

Utilizzando quanto piu codice si sia gia scritto,

Si scriva una f.

```
ListNodePtr onlyEven(ListNodePtr head);
```

Che crea una nuova lista con i soli valori pari presenti nella lista gia esistente.

Es 7 ricorsivo (+ iterativo)

Utilizzando quanto piu codice si sia gia scritto,

Si scriva una f.

```
ListNodePtr appendListTo(
ListNodePtr head,
ListNodePtr altraLista);
```

Che appende alla lista head tutti gli elementi della lista "altraLista"

ATTENZIONE! Nn concatenate la lista "altraLista" alla prima!

Es 8 ricorsivo (+ iterativo)

Utilizzando quanto piu codice si sia gia scritto,

Si scriva una f.

```
ListNodePtr reverseList(ListNodePtr head);
```

Che crea una nuova lista inversa della precedente.