Alma Mater Studiorum—Università di Bologna
Ingegneria elettronica per l'energia e l'informazione, Campus di Cesena
Fondamenti di Informatica A, a.a. 2019/2020

Laboratorio 9: Liste

Sara Montagna

29 Maggio 2020

1 Tipo di dato List e funzioni di base

Completare il programma `list.c` rispettando le specifiche fornite nei commenti del codice stesso e facendo riferimento ai lucidi sull'argomento visti a lezione. In particolare è richiesto di implementare le funzioni eseguenti:

1. `int is_empty(struct list *L)` – restituisce 1 se e solo se L è vuota, 0 altrimenti;
2. `void list_print(struct list *L)` – stampa a video il contenuto della lista L (L può anche essere la lista vuota, e in tal caso la funzione non deve stampare nulla);
3. `struct list *list_search(struct list *L, int k)` – restituisce un puntatore ad un nodo della lista L in cui il campo `val` contenga il valore k; se il valore k non è presente nella lista, la funzione deve restituire `NULL`;
4. `struct list *list_create_n(int n)` – crea una nuova lista i cui nodi contengono i valori $0, 1, \ldots, n-1$, in questo ordine (il primo nodo della lista contiene 0, il secondo contiene 1 e così via). Se $n = 0$ ritorna la lista vuota, cioè `NULL`;
5. `struct list *nth_element(struct list *L, int n)` – restituisce il puntatore all'n-esimo nodo della lista ($n = 0$ è il primo nodo, $n = 1$ è il secondo nodo, ecc.). Se il nodo non esiste, deve restituire `NULL`;
6. `struct list *list_from_array(int v[], int n)` – restituisce una nuova lista contenente gli $n \geq 0$ elementi dell'array v, nell'ordine in cui compaiono in v (il primo nodo della lista contiene $v[0]$, il secondo $v[1]$, e così via);
7. `struct list* list_duplicate(struct list *L)` – crea un duplicato della lista L, e restituisce un puntatore al primo nodo di tale duplicato; in altre parole, crea una

nuova lista avente la stessa lunghezza di L e i cui nodi contengono gli stessi valori presenti in L, nello stesso ordine.

8. `int list_compare(struct list *L1, struct list *L2)` – restituisce 1 se e solo se L_1 e L_2 contengono gli stessi valori nello stesso ordine, 0 altrimenti. Si presti attenzione al fatto che le liste potrebbero avere lunghezza diversa (in tal caso la funzione deve restituire 0).

9. `struct list *list_reverse(struct list *L)` – restituisce un puntatore al primo nodo della lista ottenuta "rovesciando" L: l'ultimo nodo di L diventa il primo della nuova lista; il penultimo diventa il secondo, e così via. Questa funzione **non deve creare nuovi nodi**, ma deve riorganizzare quelli di L. Suggerisco di adottare un approccio iterativo (non ricorsivo), in cui si esegue un ciclo nel cui corpo si rimuove il primo nodo di L per spostarlo all'inizio della lista rovesciata. Il ciclo termina quando L diventa la lista vuota...

In caso di necessità è consentito definire ulteriori funzioni di supporto (questo potrebbe essere necessario per alcune implementazioni ricorsive).

2 Esercizi sulle liste

Esercizio 1 Realizzare poi la procedura `void scambia(struct list* la, struct list* lb)` che prese in ingresso due liste, ne scambia gli ultimi due nodi (deve scambiare i nodi, non semplicemente il loro contenuto!). Per risolvere questo esercizio non si potrà usare la funzione `list_create()`, ossia, sara' necessario riusare le liste esistenti modificando opportunamente i puntatori alle code. Se una delle due liste non contiene almeno due elementi, allora la procedura scambia non dovrà produrre alcun effetto. Come riferimento si consideri il sorgente `list-scambia.c`.

Esercizio 2 Realizzare la funzione `struct list* list_delete_k(struct list *L, int k)` che, data una lista L e un intero k, cancella da L (liberando anche la memoria con la `free()`) tutti i nodi che contengono il valore k, se ce ne sono. Come riferimento si consideri il sorgente `list-delete.c`, all'interno del quale è già stato implementao un test per verificare che la funzione da voi implementata funzioni correttamente.