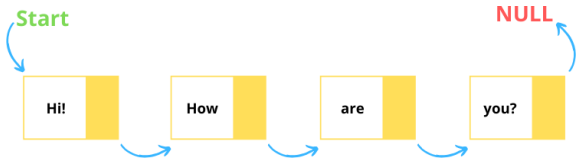
Laboratorio di Programmazione

Linked lists

17 Maggio 2021

La "linked list", o lista legata, è una struttura di dati fondamentale in informatica. Al contrario del array di cui il taglio è predeterminato, la lista ha una lunghezza che cresce o diminuisce a misura che vengono aggiunti o tolti elementi.

Una definizione induttiva di una lista è che una lista è sia la lista vuota, sia un elemento più una lista. In C, una lista viene rappresentata di questo modo: una lista è un puntatore verso il primo elemento della lista. La lista vuota è rappresentata dal puntatore NULL. Una lista non vuota punta su una struttura struct che contiene il primo elemento della lista, e un puntatore verso il resto della lista. Se questo puntatore è NULL, allora l'elemento corrente è l'ultimo elemento della lista.



In questo laboratorio si lavora su delle liste di interi, per cui utilizzerete il tipo seguente:

```
typdedef struct node {
    int val;
    struct node *next;
} list;
```

Per creare un elemento, si riserva una zona in memoria utilizzando la funzione `malloc` (per "memory allocation"), con argomento il taglio di un elemento (calcolato grazie alla funzione `sizeof`):

```
list *head = NULL;
head = (list *) malloc(sizeof(list))
head->val = 1;
head->next = NULL;
```

1. Scrivi una funzione `void print_list(list *l)` che stampa la lista `l`.
2. Scrivi una funzione `void push_tail(list *l, int val)` che aggiunge l'elemento `val` alla fine della lista `l`.
3. Scrivi una funzione `void push_head(list **l, int val)` che aggiunge l'elemento `val` all'inizio della lista `*l`. Questa volta si prende in argomento un puntatore verso l'indirizzo del primo elemento della lista perché questa funzione deve cambiare l'indirizzo del primo elemento. Alla fine dell'esecuzione, `l` deve contenere l'indirizzo del nuovo primo elemento.

4. Scrivi una funzione `int pop_tail(list *l)` che toglie l'ultimo elemento della lista `l` e ritorna il suo valore.

5. Scrivi una funzione `int pop_head(list **l)` che toglie il primo elemento della lista `*l` e ritorna il suo valore.

6. Scrivi una funzione `void remove_val(list **l, int val)` che toglie dalla lista `*l` tutti gli elementi di valore `val`.

7. Scrivi una funzione `list* intersection(list* l1, list* l2)` che ritorna una nuova lista contenente l'intersezione delle liste `l1` e `l2`.

8. Scrivi una funzione `list* union(list* l1, list* l2)` che ritorna una nuova lista contenente l'unione delle liste `l1` e `l2` (senza ripetizioni).