```
/*
 * All'inizio possiamo "dimenticarci" di includere le librerie
standard: in questo modo
 * gli mostriamo come usare il man da linea di comando
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 100

typedef struct _item {
        char name[MAX + 1];
        struct _item *next;
} item;

/* Aggiungono in coda e rimuovono in testa: gli si spiega che la
struttura dati e'
 * concettualmente definita dalla lista in sé più due puntatori a
testa e coda, e che
 * quindi e' vantaggioso definire una struct che la rappresenti.
 */
typedef struct _list {
        item *head;
        item *tail;
} list;

/* Una funzione per ogni operazione! */
item *read_item()
{
        /* Allocazione ed inizializzazione di ogni campo */
        item *to_ret = (item*)malloc(sizeof(*to_ret));
        to_ret->next = NULL;
        /* Accennare qui che non e' necessario usare la & */
        scanf("%s", to_ret->name);
        return to_ret;
}

list add_tail(list l, item *to_add)
{
        if (l.head == NULL) {
                /* Lista vuota: head e tail puntano a to_add */
                l.head = l.tail = to_add;
        } else {
                /* Lista non vuota: solo tail deve essere
aggiornata */
                l.tail->next = to_add;
                l.tail = to_add;
        }
        /* Devo restituire l aggiornata */
        return l;
}

list pop_head(list l) {
```

```
        if (l.head == NULL) {
                /* È vuota: non fai nulla */
                return l;
        } else {
                /* Aggiornare la testa e liberare la memoria */
                item *to_remove = l.head;
                l.head = l.head->next;
                free(to_remove);
                return l;
        }
}

void free_list(list l)
{
        item *now = l.head;
        while (now != NULL) {
                /* Si salva il puntatore al next: non si può
                 * accedere a "now->next" dopo aver liberato now
                 */
                item *next = now->next;
                free(now);
                now = next;
        }
}

/* Debug: si mostra, passo dopo passo, come cambia la lista dopo
ogni query */
void print_list(list l)
{
        printf("[ ");
        item *now = l.head;
        while (now != NULL) {
                printf("%s, ", now->name);
                now = now->next;
        }
        printf("]\n");
}

/* Stesso discorso della lista: un array e' dato da
 * un puntatore al primo elemento più la sua lunghezza
 */
typedef struct _array
{
        char **data;
        int len;
} array;

int list_size(list l)
{
        int size = 0;
        item *now = l.head;
        while (now != NULL) {
                ++size;
                now = now->next;
```

```
        }
        return size;
}

/* Devo "svuotare" la lista in un array */
void dump_list(list l, char **array)
{
        int pos = 0;
        item *now = l.head;
        while (now != NULL) {
                array[pos] = now->name;
                ++pos;
                now = now->next;
        }
}

/* Funzione compare per qsort */
int compare(const void *e1, const void *e2)
{
        char **s1 = (char**)e1;
        char **s2 = (char**)e2;
        return strcmp(*s1, *s2);
}

/* Stampa array finale */
void print_array(array a)
{
        int i;
        for (i = 0; i < a.len; i++) {
                printf("%s\n", a.data[i]);
        }
        printf("$\n");
}

array sort_list(list l)
{
        /* Scopri numero elementi l */
        int size = list_size(l);

        /* Alloca un array per i pazienti rimanenti e copia la
lista nell'array */
        char **arr = (char**)malloc(size * sizeof(*arr));
        dump_list(l, arr);

        /* Ordina arr usando qsort */
        qsort(arr, size, sizeof(*arr), compare);

        /* Restituisci array */
        array to_ret;
        to_ret.data = arr;
        to_ret.len = size;
        return to_ret;
}
```

```
/* Libera array */
void free_array(array a) {
        free(a.data);
}

int main()
{
        int event;
        list l;
        /* Fare attenzione all'inizializzazione */
        l.head = NULL;
        l.tail = NULL;

        /* Leggere un evento alla volta */
        scanf("%d", &event);
        while (event != 0) {
                if (event == 1) {
                        /* Leggo un elemento e lo aggiungo in coda
*/
                        item *to_add = read_item();
                        l = add_tail(l, to_add);
                } else if (event == 2) {
                        l = pop_head(l);
                } else {
                        printf("ERRORE: evento %d letto\n",
event);
                        exit(1);
                }
                /* print_list(l); */
                scanf("%d", &event);
        }

        /* Ordino array */
        array sorted = sort_list(l);
        /* Stampo */
        print_array(sorted);

        /* Libero la memoria per array e lista */
        free_array(sorted);
        free_list(l);

        return 0;
}
```