Programmazione I

26 febbraio 2013 (tempo disponibile: 2 ore)

Esercizio 1 (12 punti)

Si scriva una funzione

```
char *senza_ripetizioni(const char *s)
```

che restituisce una nuova stringa che contiene i caratteri di `s` che non si ripetono dentro `s`. L'allocazione di memoria per questa nuova stringa risultato deve essere fatta per la quantità di memoria realmente necessaria a contenere la stringa, non di più.

Per esempio, l'esecuzione del seguente programma:

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  char *buffer;

  printf("%s\n", buffer = senza_ripetizioni("ciao amico"));
  free(buffer);

  printf("%s\n", buffer = senza_ripetizioni("ciao amico come va?"));
  free(buffer);

  printf("%s\n", buffer = senza_ripetizioni("precipitevolissimevolmente"));
  free(buffer);

  return 0;
}
```

dovrà stampare:

```
 m
ev?
rcn
```

Esercizio 2 (11 punti)

Si implementi una struttura che realizza un *traduttore*, cioè una funzione associativa di una stringa a un'altra stringa, per al massimo 10 stringhe associate a 10 stringhe. Devono essere disponibili le seguenti funzioni:

- `struct traduttore *construct_traduttore()`, che restituisce un nuovo traduttore vuoto;
- `void destruct_traduttore(struct traduttore *this)`, che dealloca il traduttore `this`;
- `void put(struct traduttore *this, const char *k, const char *v)`, che associa `k` a `v` nel traduttore `this`. Questo significa che la traduzione di `k` è `v` (ma non viceversa!)
- `const char *get(struct traduttore *this, const char *k)`, che restituisce la stringa associata a `k` dentro al traduttore `this`. Questo significa che restituisce la traduzione di `k`. Se nulla è associato a `k`, restituisce `NULL`.

Per esempio, l'esecuzione del seguente programma:

```
#include <stdio.h>
#include "traduttore.h"

int main(void) {
  struct traduttore *t = construct_traduttore();
  const char *valore;

  put(t, "ciao", "hello");
  put(t, "amico", "friend");
  put(t, "come", "how");
  put(t, "va", "goes");

  printf("%s\n", (valore = get(t, "ciao")) ? valore : "NULL");
  printf("%s\n", (valore = get(t, "amico")) ? valore : "NULL");
  printf("%s\n", (valore = get(t, "come")) ? valore : "NULL");
  printf("%s\n", (valore = get(t, "va")) ? valore : "NULL");
  printf("%s\n", (valore = get(t, "oggi")) ? valore : "NULL");
  printf("%s\n", (valore = get(t, "how")) ? valore : "NULL");

  destruct_traduttore(t);

  return 0;
}
```

deve stampare:

```
hello
friend
how
goes
NULL
NULL
```

Esercizio 3 (9 punti)

Si considerino le liste di interi come definite a lezione e si implementi una funzione

`struct list *inner(struct list *this)`

che restituisce una lista uguale a `this` ma senza il primo e l'ultimo elemento di `this`. Se `this` ha meno di tre elementi, la funzione deve restituire `NULL`. La lista `this` non deve essere modificata. Se tutto è corretto, l'esecuzione del programma:

```
#include <stdlib.h>
#include <stdio.h>
#include "list.h"

int main() {
  struct list *l = construct(65, construct(77, construct(11, construct(122, construct(200, NULL)))));

  printf("inner(l) = ");
  print_list(inner(l));
  printf("\n");

  printf("l = ");
  print_list(l);
  printf("\n");

  return 0;
}
```

dovrà stampare:

```
inner(l) = [77,11,122]
l = [65,77,11,122,200]
```