Esame di Programmazione I

14 luglio 2014 (tempo disponibile: 2 ore)

Esercizio 1 (11 punti)

Si scriva una funzione `char *camelize(const char *s)` che restituisce una nuova stringa, che è la versione *camel-style* della stringa `s`, in cui cioè gli spazi sono stati eliminati e l'inizio delle parole (tranne la prima) è stato scritto in maiuscolo. Si assuma per semplicità che `s` contenga solo caratteri alfabetici minuscoli e spazi, non abbia mai due spazi di seguito e cominci e termini con un carattere alfabetico minuscolo.

Se tutto è corretto, l'esecuzione del seguente programma:

```
int main(void) {
  char *s;
  printf("%s\n", s = camelize("camels are sweet and clever animals")); free(s);
  return 0;
}
```

dovrà stampare:

```
camelsAreSweetAndCleverAnimals
```

Esercizio 2 (9 punti)

Si considerino le liste di `char` come viste a lezione. Si definisca una funzione **ricorsiva**

```
struct list *camelize_list(struct list *this)
```

che restituisce una lista ottenuta da `this` traducendola in *camel-style*. Si assuma per semplicità che la lista `this` contenga solo caratteri alfabetici minuscoli e spazi, non abbia mai due spazi di seguito e cominci e termini con un carattere alfabetico minuscolo. La lista `this` non deve venire modificata. Per esempio, l'esecuzione del programma

```
int main(void) {
  struct list *l = construct_list
    ('c', construct_list
     ('a', construct_list
      ('m', construct_list
       ('e', construct_list
        ('l', construct_list
         ('s', construct_list
          (' ', construct_list
           ('a', construct_list
            ('r', construct_list
             ('e', construct_list
              (' ', construct_list
               ('s', construct_list
                ('w', construct_list
                 ('e', construct_list
                  ('e', construct_list
                   ('t', NULL)))))))))))))));

  print_list(camelize_list(l)); printf("\n"); return 0;
}
```

dovrà stampare

```
[c, a, m, e, l, s, A, r, e, S, w, e, e, t]
```

Esercizio 3 (12 punti)

Si scrivano i file `date.h` e `date.c` che definiscono e implementano la struttura `date` che rappresenta una data del calendario, con le seguenti funzioni sulle date:

```
struct date *construct_date(int day, int month, int year);
struct date *construct_date_alt(int day, const char *month, int year);
void print_date(struct date *this);  // stampa this, del tipo "13 gennaio 1973"
int equals_date(struct date *this, struct date *that); // vero se e solo se le date sono uguali
int compare_date(struct date *this, struct date *that); // <0 se this viene prima di that,
  // 0 se this e that sono uguali, >0 se this viene dopo that
```

La funzione di costruzione `construct_date_alt` permette di specificare il mese come il nome minuscolo italiano del mese. Entramble le funzioni di costruzione ritornano `NULL` se si cerca di costruire una data inesistente e considerano gli anni bisestili.

Se tutto è corretto, l'esecuzione del seguente programma:

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

static void compare(struct date *d1, struct date *d2) {
  int comp = compare_date(d1, d2);
  print_date(d1);

  if (comp < 0) printf(" < ");
  else if (comp == 0) printf(" = ");
  else printf(" > ");

  print_date(d2);
  printf("\n");
}

int main(void) {
  struct date *d1 = construct_date(14, 7, 1789);
  struct date *d2 = construct_date_alt(15, "maggio", 1789);
  struct date *d3 = construct_date_alt(15, "giugno", 2014);
  struct date *d4 = construct_date_alt(14, "luglio", 1789);
  compare(d1, d2);
  compare(d2, d3);
  compare(d1, d3);
  compare(d1, d4);
  return 0;
}
```

dovrà stampare:

```
14 luglio 1789 > 15 maggio 1789
15 maggio 1789 < 15 giugno 2014
14 luglio 1789 < 15 giugno 2014
14 luglio 1789 = 14 luglio 1789
```