**Esame di Programmazione I, 2 settembre 2013. 2 ore**

**Esercizio 1** [**7 punti**] Si scriva una funzione ricorsiva

```
int valida(const char *s)
```

che determina se la stringa `s` è un'alternanza di cifre e non-cifre (una cifra, una non cifra, una cifra, una non cifra, ecc.). Per esempio, `3e6y2#0i5u` soddisfa tale proprietà. Si noti che tali stringhe hanno sempre lunghezza pari.

**Esercizio 2** [**13 punti**] Si scriva una funzione

```
char *espandi(const char *s)
```

che controlla se `s` è un'alternanza di cifre e non cifre e in caso contrario ritorna `NULL`. Altrimenti ritorna una nuova stringa ottenuta ripetendo le non cifre il numero di volte indicato dalla cifra alla loro sinistra. Per esempio, se `s` è `3e6y2#0i5u`, il risultato sarà la stringa `eeeyyyyyy##uuuuu`

Se tutto è corretto, una esecuzione del seguente programma:

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

int main(void) {
  char buffer[100];
  char *res;

  printf("input: ");
  scanf("%s", buffer);
  res = espandi(buffer);

  if (res) {
    printf("output: %s\n", res);
    free(res);
  }
  else
    printf("formato errato\n");

  return 0;
}
```

potrebbe essere:

```
input: 3e4r2#0&9i
output: eeerrrr##iiiiiiiii
```

**Esercizio 3** [**11 punti**] Si consideri la seguente specifica di un iscritto a un esame con il relativo voto:

```
#ifndef ISCRITTO_H
#define ISCRITTO_H

struct iscritto {
  const char *nome;
  const char *cognome;
  int matricola;
  int voto;
};

struct iscritto *construct_iscritto(const char *nome, const char *cognome, int matricola);
void destroy_iscritto(struct iscritto *this);
void print_iscritto(struct iscritto *this);

#endif
```

con la relativa implementazione:

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

struct iscritto *construct_iscritto(const char *nome, const char *cognome, int matricola) {
  struct iscritto *this = malloc(sizeof(struct iscritto));
  this->nome = nome;
  this->cognome = cognome;
  this->matricola = matricola;
  this->voto = -1;
  return this;
}

void destroy_iscritto(struct iscritto *this) {
  free(this);
}

void print_iscritto(struct iscritto *this) {
  printf("%10s %10s   (VR%i)\t\t", this->nome, this->cognome, this->matricola);

  if (this->voto >= 18)
    printf("%i", this->voto);
  else if (this->voto >= 0)
    printf("insufficiente");
  else
    printf("assente");
}
```

Si scrivano i file `appello.h` e `appello.c` in modo da implementare un appello d'esame. Un appello deve contenere una quantità variabile di iscritti, non nota a priori, che può crescere arbitrariamente. Si implementino le seguenti funzioni:

```
struct appello *construct_appello();
void destroy_appello(struct appello *this);
void iscrivi(struct appello *this, struct iscritto *studente);
void registra_voto(struct appello *this, int matricola, int voto);
void print_appello(struct appello *this);
```

La prima restituisce un nuovo appello senza ancora alcun iscritto. La seconda dealloca l'appello e tutti gli iscritti contenuti al suo interno. Il terzo permette di iscrivere uno studente a un appello. Il quarto registra un voto a uno studente, identificato tramite la sua matricola, e l'ultimo stampa un appello, cioè tutti i suoi iscritti in sequenza.

Se tutto è corretto, l'esecuzione del programma:

```
#include "appello.h"
#include "iscritto.h"

int main(void) {
  struct appello *app = construct_appello();
  iscrivi(app, construct_iscritto("Fausto", "Spoto", 153535));
  iscrivi(app, construct_iscritto("Giovanni", "Senzaterra", 123456));
  iscrivi(app, construct_iscritto("Roberta", "Rossi", 310453));
  registra_voto(app, 123456, 28);
  registra_voto(app, 153535, 17);
  print_appello(app);
  destroy_appello(app);
  return 0;
}
```

dovrà stampare:

```
   Roberta      Rossi   (VR310453) assente
  Giovanni Senzaterra   (VR123456) 28
    Fausto      Spoto   (VR153535) insufficiente
```