Fondamenti di Informatica L-A (A.A. 2006/2007) - Ingegneria Informatica
Prof.ssa Mello & Prof. Bellavista – Prova d'Esame di Giovedì 29 Marzo 2007 – durata 2h

Compito A

ESERCIZIO 1 (9 punti)

Si considerino due vettori **s1** e **s2** (di dimensioni note **dim1** e **dim2**, rispettivamente) contenenti dei caratteri ripetuti più volte. Il candidato realizzi una funzione:

```
char * sect(char *s1, int dim1, char *s2, int dim2, int *newDim)
```

che, ricevuti come parametri tali vettori e le loro dimensioni, restituisca come risultato un nuovo vettore di caratteri allocato dinamicamente (puntatore a **char**). Tale vettore deve contenere gli elementi presenti in entrambi i vettori **s1** e **s2** (cioè l'intersezione tra i due vettori iniziali); inoltre nel nuovo vettore i caratteri che eventualmente sono ripetuti in **s1** e/o in **s2** devono comparire una sola volta. Il candidato abbia cura di allocare dinamicamente la memoria sufficiente (si consideri che, al massimo, il vettore risultante sarà grande come il minore tra **s1** e **s2**). Tramite il parametro **newDim**, la funzione deve anche restituire la dimensione (eventualmente "logica") del vettore risultato.

Ad esempio, se **s1=['a', 'b', 'c', 'd', 'a', 'b', 'i']** e **s2=['a', 'i', 'l', 'i', 'a']**, il risultato deve essere un vettore di dimensione logica 2, con contenuto **['a', 'i']**.

VARIANTE SOLO PER VECCHIA MODALITÀ: In aggiunta a quanto specificato sopra, il candidato realizzi un programma che legga da un file di testo di nome **"parole.txt"** le prime due parole presenti (separate da uno o più spazi) e, utilizzando la funzione **sect(…)**, stampi a video i caratteri presenti in entrambe. Per semplicità, il candidato ipotizzi che le parole siano lunghe al massimo 24 caratteri; si faccia inoltre attenzione a non considerare, nell'invocazione della funzione **sect(…)**, il terminatore come carattere valido appartenente a **s1** e **s2**.

ESERCIZIO 2 (9 punti)

Si supponga di avere a disposizione ADT lista per interi (denominato come al solito **list**, con relative primitive). Si definisca una funzione ricorsiva:

```
list occur(list l1, list l2)
```

che, ricevute in ingresso le liste **l1** (che non contiene elementi ripetuti) e **l2** (che invece può contenere elementi ripetuti), restituisca in uscita una lista contenente il numero di volte che ogni intero in **l1** compare poi nella lista **l2**. Ad esempio, se invocata con parametro **l1 = [1, 2, 3]** e **l2 = [1, 2, 1, 1]**, la funzione deve restituire come risultato la lista **[3, 1, 0]**. A tal scopo si utilizzino solo le operazioni primitive degli ADT **list**.

ESERCIZIO 3 (6 punti)

Il seguente programma C compila correttamente? In caso affermativo, quali sono i valori stampati a tempo di esecuzione (si motivi opportunamente la risposta data)?

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

int p = 0;

void deCrypt(char * s, int pos) {
     char ch;
     ch = s[pos];
     if (ch != '\0') {
          if (ch >='0' && ch <='9') {
               while (ch >='0' && ch <='9')
                    ch = s[ch-'0'];
               printf("%c", ch);
          }
          deCrypt(s, pos+1);
     }
     else printf("\n");
}

int main(void) {
     char s[] = "abc25i089o";
     deCrypt(s, p);
     return (0);
}
```

ESERCIZIO 4 (5 punti)

Si consideri la seguente funzione:

```
int duplicate(double a, char b) {
      a=a-b;
      if ((a+b)*2 >=0) return duplicate(a, b) + duplicate(a+1, b);
      else return a;}
```

Si dica che cosa restituisce se invocata con parametri **`duplicate(2.0, 3)`** e si mostrino i record di attivazione relativi.

ESERCIZIO 5 (3 punti)

Si discutano brevemente i possibili problemi derivanti dall'utilizzo scorretto di operazioni per la deallocazione della memoria dinamica (problema dei "dangling reference") e si mostri un breve esempio di programma in cui tale problema si manifesta.

ESERCIZIO 1

```
char * sect(char * s1, int dim1, char * s2, int dim2, int * newDim) {
      int i, j, trovato;
      char * result;

      *newDim = 0;
      if (dim1>dim2)
            result = (char *) malloc(sizeof(char) * dim1);
      else
            result = (char *) malloc(sizeof(char) * dim2);

      for (i=0; i<dim1; i++) {
            trovato=0;
            for (j=0; (j<*newDim) && (!trovato); j++) {
                  if (result[j]==s1[i]) trovato=1;
            }
            for (j=0; (j<dim2) && (!trovato); j++) {
                  if (s1[i] == s2[j]) {
                        result[*newDim] = s1[i];
                        *newDim = *newDim + 1;
                        trovato = 1;
                  }
            }
      }
      return result;
}

int main(void) {
      int i, dim;
      char s1[25], s2[25], *vv;
      FILE *fp;

      fp = fopen("parole.txt", "r");
      fscanf(fp, "%s%s", s1, s2);
      vv = sect(s1, strlen(s1), s2, strlen(s2), &dim);
      printf("%d\n", dim);
      for (i=0; i<dim; i++) printf("%c ", vv[i]);

      fclose(fp); free(vv);
      return (0); }
```

ESERCIZIO 2

```
list occur(list l1, list l2) {
      int i = 0;
      list temp;
      if (empty(l1)) return emptyList();
      else {
            temp = l2;
            while (! empty(temp)) {
                  if (head(l1) == head(temp)) i++;
                  temp = tail(temp);
            }
            return cons(i, occur(tail(l1), l2) );
      }
}
```

ESERCIZIO 3

Il programma è corretto sintatticamente e la sua esecuzione produce la stampa:

`ciaoo\n`

Nella funzione **`main(…)`** viene creata una stringa ben formata, contenente sia caratteri che numeri, e poi viene invocata la funzione **`deCrypt(…)`**. Tale funzione ricorsiva scorre tutti i caratteri della stringa **`s`** fino al terminatore. Se il carattere incontrato rappresenta una cifra numerica, la funzione va a vedere il carattere presente nella posizione di indice uguale a quello indicato; se questo è ancora una cifra, viene considerato il carattere nella posizione indicata da quest'ultima cifra, e così via finché non viene incontrato un carattere non numerico. A tal punto, tale carattere viene stampato a video, e la funzione si re-invoca ricorsivamente sul carattere in posizione successiva al valore **`pos`**.

ESERCIZIO 4