Fondamenti di Informatica T-1 (A.A. 2010/2011) - Ingegneria Informatica
Prof.ssa Mello
Prova Parziale d'Esame di Mercoledì 2 Febbraio 2011 – durata 1h
Totale 12 punti, sufficienza con 7
Compito B

ESERCIZIO 1 (6 punti)

Date due liste di float `values` (valori non ordinati) e `target` (valori ordinati in modo decrescente), si realizzi una funzione ricorsiva

```
list closestInferior(list values, list target);
```

che restituisca una nuova lista contenente per ciascun elemento di `values` l'elemento della lista `target` a cui più si avvicina senza superarlo; per ipotesi la lista `target` data in ingresso non è vuota. Se tale elemento non esiste (l'elemento di `values` in considerazione è maggiore di tutti gli elementi di `target`) la funzione deve ignorare tale elemento e continuare coi successivi.

Ad esempio se `values = [2.7, 7.6, 3.5, 0.3]` e `target = [9.0, 8.0, 2.8, 2.5, 1.0]`, la funzione `closestInferior()` deve restituire la lista `[2.5, 2.8, 2.8]` poiché 2.5 è inferiore a 2.7, 2.8 è inferiore a 7.6 e 3.5, mentre non esiste un valore inferiore a 0.3.

La funzione `closestInferior()` dovrà essere implementata utilizzando le sole primitive dell'ADT lista; ogni altra funzione dovrà essere opportunamente specificata dal candidato. Si realizzi inoltre una semplice funzione `main()` di prova che invochi correttamente la funzione `closestInferior()` creata.

Nota: l'ordine degli elementi della lista restituita dalla funzione `closestInferior()` è ininfluente.

ESERCIZIO 2 (2 punti)

Un elaboratore rappresenta i numeri interi su 8 bit tramite la notazione in complemento a 2. Indicare come viene svolta la seguente operazione aritmetica calcolandone il risultato secondo la rappresentazione binaria in complemento a 2 (si trasli anche il risultato in decimale per verificare la correttezza dell'operazione):

ESERCIZIO 3 (3 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>
#include <string.h>

#define DIM 10

int divide(int* val, float* res){
      int i=0;
      while( *(val+1) != -1){
            res[i] = *val / (float)*(val+1);
            i++;
            val++;
      }
      return i;
}

int main(){
      int* values, res, i;
      float *results;
      values = (int*)malloc(sizeof(int)*DIM);
      results = (float*)malloc(sizeof(float)*DIM);
      for(i=0; i<4; i++){
            values[i] = i * 1.5 + 1;
      }
      values[i] = -1;
      res = divide(values, results);
      printf("%d\n", res);
      for(i=0; i<res; i++){
            printf("%f ", *results);
            results++;
      }
      return 0;
}
```

ESERCIZIO 4 (1 punti)

Il candidato illustri brevemente cosa è e a cosa serve la funzione `malloc()`, realizzando un breve esempio di codice che ne faccia uso correttamente.

Soluzioni

ESERCIZIO 1

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

list closestInferior(list values, list target){
      list tempList;
      float bestValue;
      int found;
      if( empty(values) ){
            return emptylist();
      }
      else{
            tempList = target;
            found = 0;
            do{
                  if( head(tempList) < head(values) ){
                        bestValue = head(tempList);
                        found = 1;
                  }
                  tempList = tail(tempList);
            }
            while( !found && !empty(tempList) );
            if(found)
                  return cons(bestValue, closestInferior(tail(values),target));
            else
                  return closestInferior(tail(values),target);
      }
}

int main() {
      list v, t, res;
      v = cons(2.7,cons(7.6,cons(3.5,cons(0.3,emptylist()))));
      t = cons(9.0,cons(8.0,cons(2.8,cons(2.5,cons(1.0,emptylist())))));
      res = closestInferior(v, t);
      while( ! empty(res) ){
            printf("%f ", head(res));
            res = tail(res);
      }
      return 0;
}
```

ESERCIZIO 2

+19 - 74 = -55

+19 --> 00010011
+74 --> 01001010
10110101
-74 --> 10110110

+19 00010011
-74= 10110110
-55 11001001 = -128 + 64 + 8 + 1 = -55

ESERCIZIO 3

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

```
3
0.5 0.5 0.8
```

La funzione `main()` alloca dinamicamente spazio sufficiente per 10 int e 10 float. Lo spazio per 10 int viene inizializzato con i seguenti 5 valori: 1, 2, 4, 5, -1; le restanti celle di memoria non vengono inizializzate. Poi la funzione `main()` invoca la funzione `divide()`.

La funzione `divide()` itera finché il valore della seconda cella partendo da `val` è diverso da -1; notare che il ciclo `while` modifica l'indirizzo a cui punta il puntatore `val`. Il ciclo `while` inizializza il valore delle celle puntate da `res` con il risultato della divisione tra i valori puntati da `res` e `res+1` promossi a float, il primo tramite promotion e il secondo tramite cast esplicito. Infine la funzione `divide()` restituisce il numero di elementi correttamente inizializzati nell'area di memoria puntata da `res`.

La funzione `main()` scrive sullo standard output la quantità di elementi inseriti nell'area di memoria puntata da `results` ed il valore di tali elementi.