COMPITO D'ESAME DEL 22/12/1997

Un file di testo `CONTI.TXT` contiene le informazioni relative ai conti correnti dei clienti di una filiale bancaria. Ogni riga del file contiene le informazioni relative a un singolo conto corrente, e precisamente, nell'ordine:

- *codice del conto* (10 caratteri): è una stringa alfanumerica che individua univocamente il conto
- *intestatario del conto* (20 caratteri): è una stringa che contiene cognome e nome dell'intestatario del conto
- *deposito* (20 caratteri): è un numero intero che rappresenta il saldo attualmente depositato nel conto.

Un secondo file di testo `TRANSAZIONI.TXT` contiene una sequenza di movimenti da effettuare sui conti, uno per riga. In particolare, ogni riga del file contiene:

- *codice del conto* (10 caratteri) su cui effettuare la modifica
- *ammontare della transazione* (20 caratteri), cioè un numero che può essere positivo o negativo, a seconda che si tratti di un'operazione di deposito o di prelievo.

Si realizzi un programma C che:

a) legga gli elementi contenuti in `CONTI.TXT` e li inserisca in una lista LC ordinata in base al codice *(punti 6)*

b) a partire dal file `TRANSAZIONI.TXT`, aggiorni la lista LC effettuando i movimenti sui conti, decrementando o incrementando il deposito sul c.c. relativo *(punti 10)*

c) calcoli e stampi a video il totale netto dei movimenti effettuati al punto b) *(punti 6)*

d) a partire da LC, scriva in un file binario `CONTI.DAT` i dati finali dei c.c. che al termine di tutte le transazioni hanno saldo *positivo*, e in un file binario `ESAURITI.DAT` quelli dei c.c. con saldo negativo o nullo *(punti 8)*

Esempio di file CONTI.TXT:

```
(10 caratteri)          (20 caratteri)          (20 caratteri)
----------XXXXXXXXXXXXXXXXXXXX--------------------
```

```
ae45er44reStefanelli Enrico              +100000000
34assd5dfgDenti Cesare                    +65000000
65dsf65sd6Milano Fabrizio                     +5000
6sdfs5d5d5Riguzzi Michela                  -10000000
2sd5fse5sdBarruffi Franco                +1200000000
6sdf56sd6fZambonelli Rosangela              -9900000
6se6r6df6sFaldella Antonio                -444400000
5ffdss8fdsCorradi Eugenio                 +444400000
```

Esempio di file TRANSAZ.TXT:

```
(10 caratteri)          (20 caratteri)
----------XXXXXXXXXXXXXXXXXXXX
```

```
6sdfs5d5d5            +10000000
ae45er44re           -200000000
34assd5dfg            +35000000
65dsf65sd6               +20000
6se6r6df6s           +400000000
5ffdss8fds           -444400000
2sd5fse5sd          -1000000000
6sdf56sd6f            +10000000
65dsf65sd6               -10000
34assd5dfg            -60000000
```


Organizzazione del progetto

- `main.c` il file del programma principale
- `conto.h` dichiara l'ADT `conto` e le funzioni correlate
- `conto.c` definisce le funzioni dell'ADT `conto`
- `element.h` dichiara l'ADT `element` e le funzioni relative
- `list.h` dichiara l'ADT `lista` e le funzioni correlate
- `list.c` definisce le funzioni dell'ADT `lista`

Perché questa organizzazione?

- il centro di tutto è l'ADT *conto* → `conto.h`, `conto.c`
- la domanda b) richiede le liste → `list.h`, `list.c`
- le liste richiedono un *element* → `element.h`

Varianti

Spesso uno stesso compito ha *più varianti* (A,B,C,D), assegnate a persone che in laboratorio si trovano fianco a fianco.

- le varianti sono molto simili e presentano, quindi, le stesse difficoltà
- ma sono "sottilmente diverse" in alcuni punti-chiave → *se uno copia si vede....*

AD ESEMPIO:

- alberi al posto di liste e viceversa
- ordinamenti diversi (in base a campi diversi e/o con criteri diversi)
- selezioni diverse da fare sugli stessi dati
- ....

L'ADT conto (`conto.h`)

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

#ifndef BOOL /* Astrazione boolean */
typedef enum {False,True} boolean;
#endif

typedef struct {
   char codice[11], intestatario[21];
   long int saldo;
} conto;

typedef conto transazione;

boolean isLess(conto, conto);
boolean isEqual(conto, conto);
void showEl(conto);
void showEl1(transazione);
```

/ Inseriamo qui le show() perché conto e transazione saranno, di fatto, il nostro element */*

L'ADT element (`element.h`)

```
#ifndef EL
#define EL

#include "conto.h"
typedef  conto  element;

#endif
```

L'ADT conto (`conto.c`)

```
#include "element.h"

boolean isLess(conto c1, conto c2){
   return (strcmp(c1.codice, c2.codice)<0);
}

boolean isEqual (conto c1, conto c2) {
   return (strcmp(c1.codice, c2.codice)==0);

      /* E' stato garantito che codice è un identificativo univoco*/
}

void showEl(conto c) {
   printf("C.C. n. %s di %s: saldo L.%12ld",
          c.codice, c.intestatario,
          c.saldo);
}

void showEl1(transazione t) {
   printf("C.C. n. %s: transazione L.%12ld",
          t.codice, t.saldo);
}
```

NOTE:

- se non ci fosse stata la garanzia di unicità del codice, la funzione `isEqual()` avrebbe dovuto verificare l'uguaglianza di *tutti* e tre i campi (codice, intestatario e saldo).
- a rigore, si sarebbe dovuto chiamare le ultime due funzioni `showConto()` e `showTransazione()`, definendo poi in `element.c` le due funzioni `showEl()` e `showEl1()` come alias per le esse.

Il main (`main.c`)

```
#include "element.h"
#include "list.h"

void updateList(list, transazione);
void listToFiles(list, FILE*, FILE*);
void showFile(FILE*);

main() {
 FILE *fctxt, *fttxt, *fdat1, *fdat2;
 char riga[52];
 conto c;
 transazione t;
 long int transtot;
 list LC, OLC, NLC;
 char ss[21];

 fctxt = fopen("conti.txt","r");
 LC = emptyList();
 while (fgets(riga,52,fctxt)) {
   strncpy(c.codice,riga,10);
   c.codice[10]='\0';
   strncpy(c.intestatario,riga+10,20);
   c.intestatario[20]='\0';
   strncpy(ss,riga+30,20); ss[20]='\0';
   c.saldo = atol(ss);
   showEl(c); printf("\n");
   LC = insord(c,LC);
 }
 showListIt(LC);
 fclose(fctxt);
```


Il main (`main.c`) (II)

```
 printf("\n");
 fttxt = fopen("transa.txt","r");
 transtot = 0;
 while (fgets(riga,32,fttxt)) {
   strncpy(t.codice,riga,10);
   t.codice[10]='\0';
   strcpy(t.intestatario,"");
   strncpy(ss,riga+10,20); ss[20]='\0';
   t.saldo = atol(ss);
   updateList(LC,t);
   transtot += t.saldo;
 }
 showListIt(LC);
 printf("\nTotale transazioni:\t%15ld\n",
         transtot);
 fclose(fttxt);

 fdat1 = fopen("conti.dat","wb");
 fdat2 = fopen("esauriti.dat","wb");
 listToFiles(LC,fdat1,fdat2);
 fclose(fdat1);
 fclose(fdat2);

 fdat1 = fopen("conti.dat","rb");
 printf("\nConti in nero:\n");
 showFile(fdat1);
 fclose(fdat1);
 fdat2 = fopen("esauriti.dat","rb");
 printf("\nConti in rosso o quasi:\n");
 showFile(fdat2);
 fclose(fdat2);
}
```


Il main (`main.c`) (III – le utility)

```
void updateList(list L, transazione t) {
   if (!empty(L))
      if (isEqual(head(L),t))
         L->value.saldo += t.saldo;
      else updateList(tail(L),t);
}

void listToFiles(list L, FILE* fnero,
                         FILE* frosso) {
   conto c;
   if (!empty(L)) {
      c = head(L);
      if (c.saldo > 0)
        fwrite(&c, sizeof(conto),1,fnero);
      else
        fwrite(&c, sizeof(conto),1,frosso);
   listToFiles(tail(L), fnero, frosso);
   }
}

void showFile(FILE * f) {
   conto c;
   while (fread(&c, sizeof(conto), 1, f)) {
      printf("\n"); showEl(c);
   }
   printf("\n");
}
```

L'uscita a video

```
C.c. n. ae45er44re di Stefanelli Enrico   : saldo L.   100000000
C.c. n. 34assd5dfg di Denti Cesare        : saldo L.    65000000
C.c. n. 65dsf65sd6 di Milano Fabrizio     : saldo L.        5000
C.c. n. 6sdfs5d5d5 di Riguzzi Michela     : saldo L.   -10000000
C.c. n. 2sd5fse5sd di Barruffi Franco     : saldo L.  1200000000
C.c. n. 6sdf56sd6f di Zambonelli Rosangela: saldo L.    -9900000
C.c. n. 6se6r6df6s di Faldella Antonio    : saldo L.  -444400000
C.c. n. 5ffdss8fds di Corradi Eugenio     : saldo L.   444400000

[
  C.c. n. 2sd5fse5sd di Barruffi Franco     : saldo L. 1200000000,
  C.c. n. 34assd5dfg di Denti Cesare        : saldo L.   65000000,
  C.c. n. 5ffdss8fds di Corradi Eugenio     : saldo L.  444400000,
  C.c. n. 65dsf65sd6 di Milano Fabrizio     : saldo L.       5000,
  C.c. n. 6sdf56sd6f di Zambonelli Rosangela: saldo L.   -9900000,
  C.c. n. 6sdfs5d5d5 di Riguzzi Michela     : saldo L.  -10000000,
  C.c. n. 6se6r6df6s di Faldella Antonio    : saldo L. -444400000,
  C.c. n. ae45er44re di Stefanelli Enrico   : saldo L.  100000000
]

[
  C.c. n. 2sd5fse5sd di Barruffi Franco     : saldo L.  200000000,
  C.c. n. 34assd5dfg di Denti Cesare        : saldo L.   40000000,
  C.c. n. 5ffdss8fds di Corradi Eugenio     : saldo L.          0,
  C.c. n. 65dsf65sd6 di Milano Fabrizio     : saldo L.      15000,
  C.c. n. 6sdf56sd6f di Zambonelli Rosangela: saldo L.     100000,
  C.c. n. 6sdfs5d5d5 di Riguzzi Michela     : saldo L.          0,
  C.c. n. 6se6r6df6s di Faldella Antonio    : saldo L.  -44400000,
  C.c. n. ae45er44re di Stefanelli Enrico   : saldo L. -100000000
]

Totale transazioni: -1249390000

Conti in nero:

C.c. n. 2sd5fse5sd di Barruffi Franco     : saldo L.   200000000
C.c. n. 34assd5dfg di Denti Cesare        : saldo L.    40000000
C.c. n. 65dsf65sd6 di Milano Fabrizio     : saldo L.       15000
C.c. n. 6sdf56sd6f di Zambonelli Rosangela: saldo L.      100000

Conti in rosso o quasi:

C.c. n. 5ffdss8fds di Corradi Eugenio     : saldo L.           0
C.c. n. 6sdfs5d5d5 di Riguzzi Michela     : saldo L.           0
C.c. n. 6se6r6df6s di Faldella Antonio    : saldo L.   -44400000
C.c. n. ae45er44re di Stefanelli Enrico   : saldo L.  -100000000
```


COMPITO D'ESAME DEL 23/2/1998

E' dato un insieme di rettangoli disposti su di un piano cartesiano, disposti con i lati paralleli agli assi.

Ogni rettangolo può quindi essere rappresentato da una coppia di *punti*, ove ogni *punto* è una coppia di coordinate, che supporremo avere valore intero.

L'insieme dei rettangoli che interessano è definito in un file di testo `RETTANG.TXT` in cui ogni riga contiene, nell'ordine, le coordinate dei due vertici (per ciascuno, nell'ordine, ascissa e ordinata, separate da spazi).

Si realizzi un programma C che:

a) definisca il tipo `RETTANG` come coppia di punti rappresentanti un rettangolo;

b) contenga una funzione `area()` che, dato un `RETTANG`, ne calcoli l'area;

c) contenga una funzione `rettalb()` che, letti tutti i rettangoli `RETTANG` descritti nel file `RETTANG.TXT`, li inserisca in un *albero binario di ricerca* ordinati secondo l'area, e restituisca infine tale albero;

d) invochi `rettalb()` per costruire l'albero binario di ricerca di rettangoli `BINRETT`, e visualizzi `BINRETT` a schermo visitandolo in ordine;

e) contenga una funzione `selalb()` che, dato un albero binario di ricerca di rettangoli e un valore reale positivo, costruisca la *lista* di tutti i rettangoli dell'albero che hanno area *maggiore o uguale* al valore dato;

f) invochi `selalb()` usando come parametri `BINRETT` e un valore reale positivo introdotto da input, e costruisca la lista di rettangoli `LISTRETT`, visualizzandola quindi a schermo.

Esempio di file RETTANG.TXT:

```
 -2  20  14  10
  2  10   6  18
  8  14  10  -3
 -2   0  -1  14
 14  -2   0  -1
 12   5  -6 -18
 -8  16 -12   4
  0   3  12   9
 13  10  -4  -8
 10  -3  14   8
```

Organizzazione del progetto

- `main.c` il file del programma principale
- `element.h` dichiara l'ADT `element` e le funzioni relative
- `element.c` definisce le funzioni dell'ADT `element`
- `tree.h` dichiara l'ADT `tree` e le funzioni correlate
- `tree.c` definisce le funzioni dell'ADT `tree`
- `list.h` dichiara l'ADT `lista` e le funzioni correlate
- `list.c` definisce le funzioni dell'ADT `lista`
- `boolean.h` dichiara `boolean` e le costanti `true` e `false`

NOTA:
`boolean.h` non compare direttamente nel nostro progetto, ma è usato (incluso) da `tree.h` e `list.h`
(rischio di inclusioni multiple → `#ifndef` ... `#endif`)

L'ADT element (`element.h`)

```
#ifndef EL
#define EL
#include <stdlib.h>
#include <stdio.h>

typedef struct { int x, y; }     coord;

   /* domanda a)*/
typedef struct { coord v1, v2; } RETTANG;

typedef struct {
          RETTANG r;  int area;
} el_type;

typedef el_type element;  /* list & tree */

unsigned area(RETTANG);    /* domanda b) */
int  fscanRett(FILE *, RETTANG *);
void showEl(el_type);
int  isLess(el_type, el_type);
int  isEqual(el_type, el_type);

#endif
```

Basta così?

Sì, perché la lista e gli alberi sono tutte strutture di *element* (lo stesso *element*!), quindi questo ADT è sufficiente.

NOTA:
poiché alberi e liste usano come tipo-base `element`, occorre fare attenzione ai nomi → una `typedef` deve definire `element` come alias di `el_type`

Implementazione di element (`element.c`)

```
#include "element.h"

int fscanRett(FILE *ftxt, RETTANG *r) {
   return fscanf(ftxt, "%d%d%d%d\n",
                 &(r->v1.x), &(r->v1.y),
                 &(r->v2.x), &(r->v2.y));
}

void showEl(el_type e) {
   printf("(%d,%d)\t(%d,%d)\t\t-- area %d",
             e.r.v1.x, e.r.v1.y,
             e.r.v2.x, e.r.v2.y, e.area);
}

/* domanda b) */
unsigned int area(RETTANG r) {
   return abs(   (r.v1.x - r.v1.y) *
                 (r.v2.x - r.v2.y)    );
}

int isLess(el_type e1, el_type e2) {
   return (e1.area < e2.area);
}
int isEqual(el_type e1, el_type e2) {
   return (e1.area == e2.area);
}
```

NOTA:

- `isLess()` determina l'ordinamento adottato nell'albero binario di ricerca.
- `isEqual()` va definita (anche se nel nostro caso non è usata) in quanto chiamata da `list.c` e `tree.c`.

Il main (`main.c`)

```
#include "element.h"
#include "tree.h
#include "list.h"

#define FILENAME "RETTANG.TXT"

tree rettalb(FILE *);          /* domanda c) */
list selalb(tree, float);      /* domanda e) */

main() {
    FILE *ftxt;
    float f;
    tree BINRETT;
    list LISTRETT;
    RETTANG rett;

    ftxt = fopen(FILENAME,"r");

        /* domanda d) */
    inOrder(BINRETT = rettalb(ftxt));

    fclose(ftxt);

    printf("\n");
    printf("\nLimite inferiore per l'area:\t");
    scanf("%f", &f);

        /* domanda f) */
    showListIt(LISTRETT = selalb(BINRETT,f));

    printf("\n");
}
```


Il main (`main.c`) (II)

```
/* domanda c) */

tree rettalb(FILE *ftxt) {
    RETTANG rett;
    el_type e;
    tree t = emptyTree();

    while (fscanRett(ftxt, &rett) != EOF) {
        e.r = rett;
        e.area = area(rett);
        showEl(e); printf("\n");
        t = insOrdTree(e,t);
    }
    return t;
}


/* domanda e) */
list selalb2(list l, tree t, float f);

list selalb(tree t, float f) {
    return selalb2(emptyList(), t, f);
}

list selalb2(list l, tree t, float f) {
    el_type e;
    if (!isEmptyTree(t)) {
        l = selalb2(l, left(t), f);
        e = root(t);
        if (e.area >= f)  l = cons(e,l);
        l = selalb2(l, right(t), f);
    }
    return l;
}
```


L'uscita a video

```
(-2,20) (14,10) -- area 88
(2,10) (6,18)   -- area 96
(8,14) (10,-3)  -- area 78
(-2,0) (-1,14)  -- area 30
(14,-2) (0,-1)  -- area 16
(12,5) (-6,-18) -- area 84
(-8,16) (-12,4) -- area 384
(0,3) (12,9)    -- area 9
(13,10) (-4,-8) -- area 12
(10,-3) (14,8)  -- area 78
(0,3) (12,9)    -- area 9
(13,10) (-4,-8) -- area 12
(14,-2) (0,-1)  -- area 16
(-2,0) (-1,14)  -- area 30
(8,14) (10,-3)  -- area 78
(10,-3) (14,8)  -- area 78
(12,5) (-6,-18) -- area 84
(-2,20) (14,10) -- area 88
(2,10) (6,18)   -- area 96
(-8,16) (-12,4) -- area 384

Limite inferiore per l'area: 80

-- (-8,16) (-12,4)  -- area 384
-- (2,10) (6,18)    -- area 96
-- (-2,20) (14,10)  -- area 88
-- (12,5) (-6,-18)  -- area 84
```