Fondamenti di Informatica L-A (A.A. 2005/2006 - CdS Ingegneria Informatica)
Prof.ssa Mello & Prof. Bellavista – Seconda Prova Intermedia del 07/12/2005 - durata 2.30h

COMPITO B

ESERCIZIO 1 (14 punti)

L'ufficio finanze del comune di Bologna vorrebbe scoprire tutti gli esercizi commerciali evasori della tassa ICI. Per conseguire tale obiettivo, il comune dispone di due file: il primo, binario, di nome "**`catasto.dat`**", contiene i dati di un insieme di negozi forniti dal catasto, salvati come strutture dati **`Lotto`**. Ogni struttura **`Lotto`** contiene il nome del negozio (stringa di 128 caratteri), l'indirizzo (stringa di 128 caratteri) e la dimensione in metri quadri del negozio (float). Il secondo file, di nome "**`tasse.txt`**", è un file di testo fornito dall'ufficio pagamenti, che contiene i dati relativi alle tasse pagate. Per ogni riga il file "**`tasse.txt`**" contiene il nome del negozio (stringa di 128 caratteri), la tassa pagata (float), e la dimensione in metri quadri del negozio per i quali è stata pagata la tassa (float). Nel file "**`tasse.txt`**" sono presenti tutti i negozi elencati nel file "**`catasto.dat`**" (e viceversa), ma eventualmente in ordine differente. I negozi evasori della tassa ICI hanno pagato le tasse, ma hanno dichiarato una dimensione del negozio minore di quella registrata al catasto. Dopo aver definito opportunamente la struttura dati **`Lotto`**, il candidato deve:

1. Realizzare una funzione
   ```
   int trovaEvasori(FILE * fCatasto, FILE * fTasse, Lotto * dest, int dim)
   ```
   che, ricevuti in ingresso opportuni puntatori a file, un array di strutture **`Lotto`** (precedentemente allocato) e la sua dimensione fisica **`dim`**, salvi nell'array **`dest`** i dati relativi ai negozi che sono presenti nel file **`fCatasto`**, e che hanno dichiarato una dimensione minore nel file **`fTasse`** (hanno quindi pagato di meno). Il candidato abbia cura di rispettare la dimensione fisica del vettore **`dim`**: qualora gli evasori siano in numero maggiore di **`dim`**, la funzione si limiti a restituire i primi **`dim`** evasori. Al termine la funzione deve restituire il numero di evasori salvati nell'array. Si ricorda al candidato l'esistenza della funzione di libreria **`rewind(FILE * f)`**, che ha l'effetto di riposizionare la testina di lettura all'inizio del file passato come parametro. **(8 punti)**

2. Realizzare un programma **`main()`**, che chieda all'utente i nomi dei file e la dimensione del file dei lotti catastali, e allochi dinamicamente sufficiente memoria per registrare gli evasori (il caso peggiore è quello in tutti i negozi hanno dichiarato una dimensione minore di quella accatastata, e quindi il numero di evasori è pari al numero di negozi registrati nel file dei lotti). Determinati gli evasori utilizzando la funzione definita al punto 1, il programma deve poi stampare a video tutti i dati relativi agli evasori, chiedere quale è la tassa al metro quadro (in euro), e stampare poi una stima di quante tasse sarebbero state pagate se nessuno avesse evaso (la stima è data dalla somma dei metri quadri dei negozi evasori moltiplicata per la tassa al metro quadro). (**6 punti**)

ESERCIZIO 2 (8 punti)

È dato un file di testo, di nome "**`analisi.txt`**", generato da uno strumento di analisi del colesterolo, contenente il risultato di tali esami. In particolare, nella prima linea del file è contenuto il limite consigliato per la salute, e poi nelle righe successive, in ogni riga, vi è il codice identificativo del campione analizzato (un intero), e il valore di colesterolo trovato (un intero).
Il candidato deve realizzare un programma che estragga dal file i codici dei campioni analizzati: in una lista **`listaLow`** devono essere inseriti i codici dei campioni il cui colesterolo è minore del limite, mentre in una seconda lista **`listaHigh`** devono essere inseriti i codici dei campioni il cui colesterolo supera il limite. Il programma deve poi stampare a video tutti i codici dei campioni che hanno problemi di colesterolo, cioè quelli registrati nella lista **`listaHigh`**, e il numero di campioni presenti in tale lista (in pratica la sua lunghezza).

1. Si realizzi una prima versione del programma supponendo di possedere il tipo di dato astratto **`list`**, con le relative operazioni primitive viste a lezione, che quindi possono anche non essere riportate nella soluzione.
2. Si mostri poi (scrivendo il codice opportuno) come il programma debba essere modificato qualora non siano disponibili le primitive, ma si possa accedere alle liste solo tramite la notazione a puntatori.

ESERCIZIO 3 (5 punti)

Qualora il programma sorgente C seguente compili ed esegua correttamente, se ne indichino i valori stampati a tempo di esecuzione, motivando la risposta data. In caso di errori di compilazione o errori runtime, si descriva invece nel dettaglio la motivazione di tali errori.

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

char terminatore = '\0';

typedef struct node {
    char value;
    struct node * next;
} Node;
typedef Node * myString;

myString cp(char v[]) {
    myString temp, result = NULL;
    while (v[0]!='\0') {
      temp = (myString) malloc(sizeof(Node));
      temp ->value = v[0]; temp ->next = result;
      result = temp;
      v++; };
    temp = (myString) malloc(sizeof(Node));
    temp ->value = terminatore; temp->next = result;
    result = temp;
    return result; }

int main () {
    char temp[] = "Ciao";
    myString st1;
    int i=0;

    st1 = cp(temp);
    while (st1 != NULL) {
      if (st1->value != terminatore) printf("%c", st1->value);
      i++; st1 = st1 ->next;
    }
    printf("\n%d\n", i);
    return 0; }
```

ESERCIZIO 4 (5 punti)

Utilizzando il tipo di dato astratto **`stack`** presentato a lezione (stack che contenga dati di tipo **`char`**), il candidato realizzi una funzione:

`void proc(char * temp);`

che, dato in ingresso una stringa ben formata **`temp`**, ne stampi a video i caratteri in posizione dispari ed in ordine inverso (terminatore escluso). Si ricorda che uno **`stack`** può essere costruito con la funzione **`newStack()`**; le funzioni **`push(char el, stack * st)`** e **`el pop(stack * st)`** sono utilizzate per inserire un elemento nello stack e per togliere il primo elemento, e che **`isEmptyStack(stack st)`** restituisce un valore logico vero se lo stack è vuoto. Ad esempio, data la stringa di ingresso "**`Inform`**", i caratteri in posizione dispari sono {**`n,o,m`**}, e quindi la funzione deve stampare a video "**`mon`**". Per fare questo, la funzione carichi opportunamente i caratteri della stringa in ingresso nella struttura dati di tipo **`stack`**; le funzioni primitive e la definizione di stack possono anche non essere riportate nella soluzione.

Soluzione Compito B

Esercizio 1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define DIM 129

typedef struct lotto {
      char societa[DIM];
      char indirizzo[DIM];
      float mq;
} Lotto;

int trovaEvasori(FILE * fCatasto, FILE * fTasse, Lotto * dest, int dim) {
      Lotto tempLotto;
      int trovato, realDim;
      char nome[DIM];
      float tassa, mq;

      realDim = 0;
      while ((fread(&tempLotto, sizeof(Lotto), 1, fCatasto) > 0) && realDim < dim)
{
            rewind(fTasse);
            trovato = 0;
            while (fscanf(fTasse, "%s %f %f", nome, &tassa, &mq)!= EOF
                        && !trovato) {
                  if (strcmp(tempLotto.societa, nome) == 0) {
                        trovato = 1;
                        if (tempLotto.mq > mq) {
                              dest[realDim] = tempLotto;
                              realDim = realDim + 1;
                        }
                  }
            }
      }
      return realDim;
}

int main()
{
      FILE * fCatasto, * fTasse;
      int i, maxDim, realDim;
      Lotto * vet;
      char nome1[DIM], nome2[DIM];
      float costoICI, totale = 0;

      printf("Ins. il nome del file del catasto e la dimensione massima: ");
      scanf("%s %d", nome1, &maxDim);
      printf("Ins. il nome del file delle tasse:");
      scanf("%s", nome2);

      if ((fCatasto = fopen(nome1, "rb")) == NULL) {
            printf("Errore in apertura in lettura di %s\n", nome1);
            exit(-1);
      }
      if ((fTasse = fopen(nome2, "r")) == NULL) {
            printf("Errore in apertura in lettura di %s\n", nome2);
            exit(-1);
      }
```

```
	vet = (Lotto *) malloc(sizeof(Lotto)* maxDim);

	realDim = trovaEvasori(fCatasto, fTasse, vet, maxDim);
	for (i=0; i<realDim; i++) {
		printf("Evasore: %s, mq.: %f\n", vet[i].societa, vet[i].mq);
		totale = vet[i].mq + totale;
	}
	printf("Inserire costo ICI: ");
	scanf("%f", &costoICI);
	printf("Stima perdita: %f", totale * costoICI);

	fclose(fCatasto);
	fclose(fTasse);
	free(vet);
	return 0;
}
```

Esercizio 2

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

int main() {
	FILE * fp;
	list listaLow, listaHigh;
	int tempId, tempValues;
	int limit, length=0;

	if ((fp = fopen("analisi.txt", "r")) == NULL) {
		printf("Errore in apertura in lettura di %s\n", "analisi.txt");
		exit(-1);
	}

	listaLow = emptylist();
	listaHigh = emptylist();

	fscanf(fp, "%d", &limit);
	while ( fscanf(fp, "%d %d", &tempId, &tempValues) != EOF) {
		if (tempValues <limit)
			listaLow = cons(tempId, listaLow);
		/* in alternativa, se non possiedo le primitive
		{
			temp = (list) malloc(sizeof(item));
			temp -> value = tempId;
			temp -> next = listaLow;
			listaLow = temp;
		}
		*/
		else
			listaHigh = cons(tempId, listaHigh);
		/* in alternativa, se non possiedo le primitive
		{
			temp = (list) malloc(sizeof(item));
			temp -> value = tempId;
			temp -> next = listaHigh;
			listaHigh = temp;
		}
		*/
	}
```

```
	fclose(fp);

	while(!empty(listaHigh)) {
		printf("Codice: %d\n", head(listaHigh));
		listaHigh = tail(listaHigh);
		length++;
	}
	/* in alternativa, se non possiedo le primitive:
	while(listaHigh!=NULL){
		printf("Codice: %d\n", listaHigh->value);
		listaHigh = listaHigh -> next;
		length++;
	}
	*/
	printf("%d campioni hanno problemi di colesterolo\n", length);
	return 0;
}
```

Esercizio 3

Il programma stampa:

oaiC
5

Il programma **main()**, dopo aver dichiarato un array **temp** di caratteri, subito inizializzato come stringa a "**Ciao**", invoca la funzione **cp**, che restituisce tutti i caratteri dell'array (compreso il terminatore) memorizzati però tramite una lista.
La funzione **cp** non fa altro che copiare il contenuto della stringa **v** passata come parametro nella lista **result** di tipo **myString** (lista identica a quelle viste a lezione, con la differenza che invece di un intero, la lista memorizza un carattere). Da notare che, siccome la lista è costruita aggiungendo gli elementi in testa, l'ordine degli elementi viene invertito.
Infine, il ciclo **while** inserito nel programma **main()** stampa a video i caratteri memorizzati nella lista, avendo cura di non stampare il terminatore (che non è visualizzabile a video). Contemporaneamente tiene traccia tramite il contatore **i** di quanti elementi ci sono nella lista (4 per le lettere di "Ciao" e un terminatore), e quindi stampa il valore 5 a video.

Esercizio 4

```
#include "element.h"
#include "stack_st.h"

void proc(char * temp) {
	stack st;
	st = newStack();
	while (*temp != '\0') {
		temp = temp +1;
		if (*temp != '\0') {
			push(*temp, &st);
			temp = temp + 1;
		}
	}
	while (!isEmptyStack(st)) printf("%c", pop(&st));
	return; }

int main() {
	proc("Informa");
	scanf("%*c");
	return 0;}
```