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 A

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 "**`comune.dat`**", contiene i dati di tutti i negozi, salvati come strutture dati **`Recapito`**. Ogni struttura **`Recapito`** contiene il nome del negozio (stringa di 128 caratteri), indirizzo (stringa di 128 caratteri) e 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 alcuni dati relativi ai soli negozi che hanno pagato. 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). Dopo aver definito opportunamente la struttura dati **`Recapito`**, il candidato deve:

1. Realizzare una funzione
   ```
   int trovaEvasori(FILE * fComune, FILE * fTasse, Recapito * dest, int dim)
   ```
   che, ricevuti in ingresso opportuni puntatori a file, un array di strutture **`Recapito`** (precedentemente allocato) e la sua dimensione fisica **`dim`**, salvi nell'array **`dest`** i dati relativi ai negozi che sono presenti nel file **`fComune`**, **ma** non sono presenti nel file **`fTasse`** (cioè non hanno pagato). 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 degli indirizzi, e allochi dinamicamente sufficiente memoria per registrare gli evasori (il caso peggiore è quello in cui il file **`fTasse`** è vuoto, e quindi il numero di evasori è pari al numero di negozi registrati nel file degli indirizzi). Determinati gli evasori utilizzando la funzione definita al punto 1, il programma deve 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 non sono state pagate (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 "**`esami.txt`**", generato da uno strumento di analisi della glicemia, contenente il risultato di tali esami. In particolare, nella prima linea del file sono contenuti il limite inferiore e il limite superiore che delimitano l'intervallo in cui i valori di glicemia sono considerati normali (due valori interi), e poi nelle righe successive, in ogni riga, vi è il codice identificativo del campione analizzato (un intero), e il valore di glicemia trovato (un intero).
Il candidato deve realizzare un programma che estragga dal file i codici dei campioni che non rispettano i limiti dati: in particolare in una lista **`listaLow`** devono essere inseriti i codici dei campioni la cui glicemia è minore del limite inferiore, mentre in una seconda lista **`listaHigh`** devono essere inseriti i codici la cui glicemia supera il limite superiore. Il programma deve poi stampare a video tutti i codici registrati nella lista **`listaLow`**.

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)

Nel caso in cui 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 result = NULL, temp;
    while (*v!='\0') {
      temp = (myString) malloc(sizeof(Node));
      temp ->value = *v; temp->next = result;
      result = temp;
      v = v+1; };
    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);
    for (; st1!= NULL; st1 = st1->next,i++)
      if (st1->value != terminatore) printf("%c", st1->value);
    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 pari ed in ordine inverso (terminatore escluso). Si ricorda che uno **`stack`** può essere costruito tramite 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 pari sono {**`I,f,r`**}, e quindi la funzione deve stampare a video "**`rfI`**". 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 A

Esercizio 1

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

typedef struct recapito {
      char societa[DIM];
      char indirizzo[DIM];
      float mq;
} Recapito;

int trovaEvasori(FILE * fComune, FILE * fTasse, Recapito * dest, int dim) {
      Recapito tempRecapito;
      int trovato, realDim=0;
      char nome[DIM];
      float tassa, mq;

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


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

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

      if ((fComune = 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 = (Recapito *) malloc(sizeof(Recapito)* maxDim);
```

```
	realDim = trovaEvasori(fComune, 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(fComune); fclose(fTasse);
	free(vet);
	return 0;
}
```

Esercizio 2

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

int main()
{
	FILE * fp;
	list listaLow, listaHigh, temp;
	int tempId, tempValues;
	int low, high;

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

	fscanf(fp, "%d %d", &low, &high);
	while ( fscanf(fp, "%d %d", &tempId, &tempValues) != EOF) {
		if (tempValues <low)
			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 if (tempValues > high)
			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(listaLow)) {
		printf("Codice: %d\n", head(listaLow));
		listaLow = tail(listaLow);
	}
	/* in alternativa, se non possiedo le primitive:
	while(listaLow!=NULL){
		printf("Codice: %d\n", listaLow->value);
		listaLow = listaLow -> next;
	}
	*/

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

int main() {
	proc("Inform");
	return 0; }
```