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 D

ESERCIZIO 1 (14 punti)

L'ufficio recupero crediti di una banca deve calcolare le sanzioni economiche da applicare ai clienti titolari di mutuo che hanno pagato una rata in ritardo. A tale scopo, sono disponibili due file: il primo, binario, di nome "**`mutui.dat`**", contiene i dati relativi ai mutui, salvati come strutture dati **`Mutuo`**. Ogni struttura **`Mutuo`** contiene il cognome del cliente (stringa di 128 caratteri), il giorno del mese entro il quale deve essere pagata la rata (intero) e l'importo di tale rata (float). Per semplicità si supponga che non possano esistere clienti mutuatari con lo stesso cognome. Il secondo file, di nome "**`pagamenti.txt`**", è un file di testo fornito dalle filiali della banca che contiene i dati relativi alle rate pagate. Per ogni riga, "**`pagamenti.txt`**" contiene, nell'ordine, il cognome del cliente (stringa di 128 caratteri), il giorno in cui ha effettuato il pagamento (intero), e l'importo pagato (float). Nel file "**`pagamenti.txt`**" sono presenti tutti i clienti elencati nel file "**`mutui.dat`**" (e viceversa), ma eventualmente in ordine differente. I clienti da penalizzare sono coloro che hanno pagato con almeno un giorno di ritardo rispetto al giorno stabilito. Dopo aver definito opportunamente la struttura dati **`Mutuo`**, il candidato deve:

1. realizzare una funzione
   ```
   int estraiMulte(FILE *fMutui, FILE *fPagamenti, Mutuo *dest, int dim)
   ```
   che, ricevuti in ingresso opportuni puntatori a file, un array di strutture **`Mutuo`** (precedentemente allocato) e la sua dimensione fisica **`dim`**, salvi nell'array **`dest`** i dati relativi ai clienti che sono presenti nel file **`fMutui`**, e che hanno pagato in un giorno successivo a quello previsto. Il candidato abbia cura di rispettare la dimensione fisica del vettore **`dim`**: qualora i ritardatari siano in numero maggiore di **`dim`**, la funzione si limiti a restituire i primi **`dim`** ritardatari. Al termine la funzione deve restituire il numero di ritardatari 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 il numero di clienti registrati nel file contenente i mutui, e allochi dinamicamente memoria sufficiente a registrare i ritardatari (il caso peggiore è quello di tutti i clienti con pagamento in ritardo, e quindi il numero di clienti da multare è pari al numero di clienti registrati nel file dei mutui). Determinati i ritardatari utilizzando la funzione definita al punto 1, il programma deve chiedere all'utente un tasso percentuale che rappresenta la penale, e poi stampare a video cognome, importo previsto e penale relativamente ad ogni ritardatario. La penale è data dall'importo della rata moltiplicato per il tasso di penale. (**6 punti**)

ESERCIZIO 2 (8 punti)

È dato un file di testo "**`dna.txt`**", generato da uno strumento di analisi del dna. Nella prima linea del file sono contenuti i valori di soglia **`errorVal`** (float), e **`mutationVal`** (float). Nelle righe seguenti, in ogni riga, vi è il codice identificativo del campione di dna (intero) e il risultato dell'analisi (float). I campioni il cui valore di analisi è inferiore a **`errorVal`** sono difettosi, mentre i campioni il cui valore è compreso tra **`errorVal`** e **`mutationVal`** sono da considerarsi "normali"; infine i campioni il cui risultato di analisi risulta essere superiore a **`mutationVal`** contengono una mutazione.
Il candidato deve realizzare un programma che estragga dal file i codici dei campioni analizzati con successo: in una lista **`listaNormali`** devono essere inseriti i codici identificativi dei campioni normali, mentre in una seconda lista **`listaMutati`** devono essere inseriti i codici identificativi dei campioni che contengono una mutazione. Il programma deve poi stampare a video tutti i codici dei campioni mutati, cioè quelli registrati nella lista **`listaMutati`**, insieme al numero di campioni presenti in tale lista.

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 termina = '\0';

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

myString inv(char v[], int dim) {
    myString temp, result = NULL; int i;
    for (i=0; i<dim; i++) {
      temp = (myString) malloc(sizeof(Node));
      temp->value = v[0]; temp->next = result;
      result = temp;
      v++;
    }
    return result;
}

int main () {
    char temp[] = "Mare"; myString st1; int i=5;
    st1 = inv(temp, i);
    while (st1 != NULL) {
      if (st1->value != termina)
            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 * v);`

che, data in ingresso una stringa ben formata **`v`**, ne stampi a video i soli caratteri in posizione pari e in ordine inverso (terminatore escluso). Si ricorda che uno **`stack`** può essere costruito tramite la funzione **`newStack()`**; le funzioni **`push(…)`** e **`pop(…)`** sono utilizzate per inserire un elemento nello **`stack`** e per togliere il primo elemento, e che **`isEmptyStack(…)`** restituisce 1 se lo **`stack`** è vuoto. Ad esempio, data la stringa di ingresso "**`Federico`**", la funzione deve stampare a video "**`crdF`**". 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 D

Esercizio 1

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

#define DIM 129

typedef struct mutuo {
      char cognome[DIM];
      int giornoDelMese;
      float importoRata;
} Mutuo;

int estraiMulte(FILE * fMutui, FILE * fPagamenti, Mutuo * dest, int dim) {
      Mutuo tempMutuo;
      int trovato, realDim, data;
      char cognome[DIM];
      float tempImporto;

      realDim = 0;
      while ((fread(&tempMutuo, sizeof(Mutuo), 1, fMutui) > 0) && realDim < dim) {
            rewind(fPagamenti);
            trovato = 0;
            while (fscanf(fPagamenti,"%s %d %f", cognome,&data,&tempImporto)!= EOF
                        && !trovato) {
                  if (strcmp(tempMutuo.cognome, cognome) == 0) {
                        trovato = 1;
                        if (data > tempMutuo.giornoDelMese) {
                              dest[realDim] = tempMutuo;
                              realDim = realDim + 1;
                        }
                  }
            }
      }
      return realDim;
}

int main() {
      FILE * fMutui, * fPagamenti;
      int i, maxDim, realDim;
      Mutuo * vet;
      char nome1[DIM], nome2[DIM];
      float tassoPenale, totale = 0;

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

      printf("Ins. il nome del file dei pagamenti:");
      scanf("%s", nome2);

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

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

	realDim = estraiMulte(fMutui, fPagamenti, vet, maxDim);

	fclose(fMutui);
	fclose(fPagamenti);

	printf("Inserire tasso percentuale della penale: ");
	scanf("%f", &tassoPenale);
	for (i=0; i<realDim; i++) {
		printf("Multa: %s, importo rata: %f, penale: %f\n",
		vet[i].cognome, vet[i].importoRata, vet[i].importoRata*tassoPenale);
	}
	free(vet);

	return 0;
}
```

Esercizio 2

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

int main(){
	FILE * fp;
	list listaNormali, listaMutazioni;
	float tempVal, errorVal, mutationVal;
	int tempId, mutazioni=0;

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

	listaNormali = emptylist();
	listaMutazioni = emptylist();

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

```
	fclose(fp);

	while(!empty(listaMutazioni)) {
		printf("Codice: %d\n", head(listaMutazioni));
		listaMutazioni = tail(listaMutazioni);
		mutazioni++;
	}
	/* in alternativa, se non possiedo le primitive:
	while(listaMutazioni!=NULL){
		printf("Codice: %d\n", listaMutazioni->value);
		listaMutazioni = listaMutazioni -> next;
		mutazioni++;
	}
	*/

	printf("Sono state trovate %d mutazioni\n", mutazioni);
	scanf("%*c");
	return 0; }
```

Esercizio 3

Il programma stampa:

eraM
0

Il programma **main()**, dopo aver dichiarato un array **temp** di caratteri, subito inizializzato come stringa a "**Mare**", invoca la funzione **inv**, che restituisce tutti i caratteri dell'array (compreso il terminatore) memorizzati però tramite una lista.
La funzione **inv** 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 decrementa i per ogniciclo, ed alla fine vale 0.

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("Federico");
	return 0;
}
```