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 C

ESERCIZIO 1 (14 punti)

Una società è stata incaricata di sollecitare il pagamento di rate da parte di alcuni titolari di prestito. A tale scopo, la società dispone di due file: il primo, binario, di nome "**`prestiti.dat`**", contiene i dati dei debitori, salvati come strutture dati **`Prestito`**. Ogni struttura **`Prestito`** contiene il nome e il cognome del titolare del prestito (2 stringhe di 128 caratteri l'una), e l'importo ancora da pagare espresso in euro (float). Il secondo file, di nome "**`rate.txt`**", è un file di testo contenente i dati di chi ha pagato regolarmente nell'ultimo mese. Per ogni riga, "**`rate.txt`**" contiene il cognome della persona (stringa di 128 caratteri), il numero della rata pagata (intero), e l'importo di tale rata (float). Dopo aver definito opportunamente la struttura dati **`Prestito`**, il candidato deve:

1. Realizzare una funzione:
   ```
   int trovaRitardatari(FILE *fPrestiti, FILE *fRate, Prestito *dest, int dim)
   ```
   che, ricevuti in ingresso opportuni puntatori a file, un array di strutture **`Prestito`** (precedentemente allocato) e la sua dimensione fisica **`dim`**, salvi nell'array **`dest`** i dati relativi alle persone che sono presenti nel file **`fPrestiti`**, **ma** non sono presenti nel file **`fRate`** (cioè non hanno pagato nell'ultimo mese). Si supponga, per semplicità, che non esistano casi di più persone con lo stesso cognome. 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 due file e il numero dei prestiti contenuti nel file corrispondente, e allochi dinamicamente memoria sufficiente a registrare chi non ha ancora pagato (il caso peggiore è quello in cui il file **`fRate`** è vuoto, e quindi il numero di ritardatari è pari al numero di persone che hanno ottenuto un prestito). Determinati i ritardatari utilizzando la funzione definita al punto 1, il programma deve stampare a video tutti i dati relativi a chi non ha ancora pagato, chiedere all'utente quale è l'importo medio di una tipica rata (in euro), e stampare poi una stima di quante rate devono ancora essere pagate dai ritardatari (la stima è data dalla somma degli importi che ancora devono essere pagati dai ritardatari divisa per l'importo medio di una rata). (**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 (intero), e il valore di glicemia trovato (intero).
Il candidato deve realizzare un programma che estragga dal file i codici dei campioni, suddividendoli in base al fatto se i valori di glicemia sono o non sono compresi nell'intervallo: in una lista **`listaDentro`** devono essere inseriti i codici dei campioni la cui glicemia è nell'intervallo, mentre in una seconda lista **`listaFuori`** devono essere inseriti i codici la cui glicemia è fuori dall'intervallo. Il programma deve poi stampare a video tutti i codici registrati nella lista **`listaFuori`**.

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 copy(char *v, int dim) {
    myString temp, result = NULL;
    int i = dim-1;
    while (i>=0) {
      temp = (myString) malloc(sizeof(Node));
      temp->value = v[i];
      temp->next = result;
      result = temp;
      i = i-1;
    };
    return result; }

int main () {
    char temp[] = "Neve";
    myString st1;
    int i=0;
    st1 = copy(temp, 5);
    for (; st1 ->value != terminatore; st1 = st1->next,i++)
            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 **`int`**), il candidato realizzi una funzione:

`void proc(int *temp);`

che, dato in ingresso un array di interi positivi **`temp`**, terminati dal valore -1, ne stampi a video i valori in posizione dispari ed in ordine inverso (terminatore -1 escluso). Si ipotizzi per semplicità che **`temp`** abbia sempre almeno un elemento, il teminatore -1. 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 un valore logico vero se lo **`stack`** è vuoto. Ad esempio, dato l'array di ingresso {1,2,3,4,5,-1}, la funzione deve stampare a video "42". Per fare questo, la funzione carichi opportunamente gli interi 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 C

Esercizio 1

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

#define DIM 129

typedef struct prestito {
      char nome[DIM];
      char cognome[DIM];
      float importo;
} Prestito;

int trovaRitardatari(FILE * fPrestiti, FILE * fRate, Prestito * dest, int dim) {
      Prestito tempPrestito;
      int trovato, realDim, numRata;
      char cognome[DIM];
      float importo;

      realDim = 0;
      while ((fread(&tempPrestito, sizeof(Prestito), 1, fPrestiti) > 0)
                  && realDim < dim) {
            rewind(fRate);
            trovato = 0;
            while (fscanf(fRate, "%s %d %f", cognome, &numRata, &importo)!= EOF
                        && !trovato) {
                  if (strcmp(tempPrestito.cognome, cognome) == 0)
                        trovato = 1;
            }
            if (!trovato) {
                  dest[realDim] = tempPrestito;
                  realDim = realDim + 1;
            }
      }
      return realDim;}

int main() {
      FILE * fPrestiti, * fRate;
      int i, maxDim, realDim;
      Prestito * vet;
      char nome1[DIM], nome2[DIM];
      float totale = 0, rataMedia;

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

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

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

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

	realDim = trovaRitardatari(fPrestiti, fRate, vet, maxDim);
	fclose(fPrestiti);
	fclose(fRate);

	for (i=0; i<realDim; i++) {
		printf("Ritardatario: %s %s, importo da pagare: %f\n", vet[i].nome,
				vet[i].cognome, vet[i].importo);
		totale = vet[i].importo + totale;
	}
	free(vet);

	printf("Inserire importo rata media: ");
	scanf("%f", &rataMedia);
	printf("Rate da incassare: %f", totale / rataMedia);

	return 0;}
```

Esercizio 2

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

int main() {
	FILE * fp;
	list listaDentro, listaFuori, temp;
	int tempId, tempValues;
	int lower, higher;

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

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

```
		*/
	}
	fclose(fp);

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

Esercizio 3

Il programma stampa:

Neve
4

Il programma **main()**, dopo aver dichiarato un array **temp** di caratteri, subito inizializzato come stringa a "**Neve**", invoca la funzione **copy**, che restituisce tutti i caratteri dell'array (compreso il terminatore) memorizzati però tramite una lista.
La funzione **copy** 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, nonostante la lista sia costruita aggiungendo gli elementi in testa, l'ordine degli elementi non viene invertito poiché l'array **temp** è "visitato" in senso inverso, dalla fine all'inizio.
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 escluso il terminatore (4 per le lettere di "Neve"), e quindi stampa il valore 4 a video.

Esercizio 4

```
#include "element.h"
#include "stack_st.h"
void proc(int * temp) {
	stack st;
	st = newStack();
	while (*temp != -1) {
		temp++;
		if (*temp != -1) {
			push(*temp, &st);
			temp++;
		}
	}
	while (!isEmptyStack(st)) printf("%d", pop(&st));
	return;
}
int main() {
	int v[] = {1,2,3,4,5,-1};
	proc(v);
	return 0;
}
```