Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 3A di Giovedì 9 Febbraio 2012 – tempo a disposizione 2h

Prima di cominciare: si scarichi dal sito **`http://esamix.labx`** il file **`StartKit3A.zip`** contenente i file necessari (*progetto Visual Studio* ed eventuali altri file di esempio).
Avvertenze per la consegna: apporre all'inizio di ogni file sorgente un commento contenente i propri dati (**cognome, nome, numero di matricola**) e il **numero** della prova d'esame. Al termine, **consegnare tutti i file sorgenti** e i file contenuti nello StartKit.
Nota: **il main non è opzionale**; i *test* richiesti vanno implementati.
Consiglio: per verificare l'assenza di *warning*, eseguire di tanto in tanto *"Rebuild All"*.

Uno studio medico tiene traccia dei pazienti e di alcune informazioni sensibili tramite un insieme di file. In particolare, in un file di testo "**`nomi.txt`**", sono salvati su ogni riga il **nome** di un paziente (una stringa di al più 255 caratteri, senza spazi), il **cognome** del paziente (una stringa di al più 255 caratteri, senza spazi), e il **codice identificativo** unico del paziente (un intero); i valori sono separati tra di loro da uno spazio. I pazienti sono memorizzati nel file in ordine casuale, e non è noto a priori quanti pazienti siano presenti nel file.
In un secondo file di testo di nome "**`esami.txt`**" sono registrate invece le informazioni relative agli esami medici sostenuti dai pazienti. In particolare, in ogni riga del file, sono registrate le seguenti informazioni: il **codice identificativo** unico del paziente (un intero), il nome **dell'esame** clinico sostenuto (una stringa di al più 255 caratteri, senza spazi), e infine **l'anno** in cui l'esame è stato sostenuto; i valori sono separati tra di loro da uno spazio. Gli esami sono memorizzati in ordine casuale, e non è noto a priori quanti esami siano memorizzati nel file.
A titolo di esempio, si vedano i file omonimi forniti nello StartKit.

Esercizio 1 – Struttura dati Paziente e funzioni di lettura/stampa (mod. element.h/.c e paz.h/paz.c)

Si definisca un'opportuna struttura dati **`Paziente`**, al fine di rappresentare i dati relativi a ogni paziente. Si definisca la funzione:

```
list leggiPazienti(char* fileName);
```

che, ricevuto in ingresso il nome di un file di testo rappresentante l'elenco dei pazienti, restituisca una lista di strutture dati di tipo **`Paziente`**, contenente tutte le informazioni presenti nel file il cui nome è passato come parametro. La lista restituita deve essere ordinata in senso lessicografico in base al cognome del paziente e, in caso di cognomi identici, in base al nome.
Si definisca la procedura:

```
void stampaLista(list v);
```

che, ricevuta in ingresso una lista di strutture dati di tipo **`Paziente`**, stampi a video l'elenco dei pazienti con tutte le informazioni relative.
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra.

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 3A di Giovedì 9 Febbraio 2012 – tempo a disposizione 2h

Esercizio 2 – Struttura dati Esame, lettura esami e ordinamento (moduli element.h/.c e paz.h/.c)

Dopo aver definito una opportuna struttura dati di nome **`Esame`**, atta a contenere le informazioni relative ad un esame clinico, il candidato definisca una funzione:

```
Esame * leggiEsami(char * fileName, int * dim);
```

che, ricevuto in ingresso il nome di un file contenente gli esami clinici, e un intero passato per riferimento, restituisca un vettore allocato dinamicamente (della dimensione minima necessaria) contenente gli esami clinici memorizzati nel file indicato come parametro. Tramite l'intero passato per riferimento, la funzione deve restituire la dimensione del vettore allocato dinamicamente.
Il candidato definisca poi una procedura:

```
void ordina(Esame * v, int dim);
```

tale che, ricevuti un vettore di strutture dati di tipo **`Esame`** e la dimensione di tale vettore, ordini tale vettore in base al codice identificativo unico del paziente, poi in base al nome dell'esame clinico (in ordine lessicografico), ed infine in base all'anno in cui l'esame è stato effettuato (in senso crescente).
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra.

Esercizio 3 –Analisi degli esami più frequenti in base ai pazienti (modulo paz.h/paz.c)

Si vuole fornire qualche statistica riguardo gli esami fatti dai pazienti, al fine di capire quali sono gli esami più frequenti, e quali sono i pazienti che li fanno. A tal scopo, si definisca una procedura:

```
void ripetuti(list pazienti, Esame * vEsame, int dim);
```

che, ricevuti come parametri la lista dei pazienti, e un vettore contenente gli esami e la dimensione di tale vettore, stampi a video i dati dei pazienti, il nome dell'esame e il numero di volte che tale esame è stato fatto, solo nel caso che il singolo paziente abbia sostenuto almeno 3 volte lo stesso esame. Ad esempio, usando come riferimento i file forniti nello StartKit, risulta che il paziente con codice identificativo 46 ha sostenuto cinque volte l'esame "ECG_sotto_sforzo". In maniera simile, il paziente con codice identificativo 34 l'esame "ECG_sotto_sforzo" tre volte. La funzione deve stampare a video codice identificativo, nome e cognome del paziente, assieme al nome dell'esame che è stato ripetuto almeno 3 volte, e al numero effettivo di quante volte quell'esame è stato ripetuto.

Esercizio 4 – Stampa delle coppie esame/paziente più frequenti, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che stampi a video le coppie di pazienti—esami più frequenti. Si vuole cioè stampare a video l'elenco dei pazienti che hanno sostenuto almeno 3 volte uno stesso esame. A tal fine si utilizzino tutte le funzioni definite ai punti precedenti.

Al termine del programma, il candidato abbia cura di de-allocare tutta la memoria allocata dinamicamente, ivi compresa la memoria allocata per le liste.

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 3A di Giovedì 9 Febbraio 2012 – tempo a disposizione 2h

"element.h":

```
#include <string.h>
#include "list.h"

#ifndef _ELEMENT_H
#define _ELEMENT_H
#define DIM_NOME 256

typedef struct {
	char cognome[DIM_NOME];
	char nome[DIM_NOME];
	int id;
} Paziente;
typedef Paziente element;

typedef struct {
	int id;
	char nomeEsame[DIM_NOME];
	int anno;
} Esame;

int comparePaziente(Paziente p1, Paziente p2);
int compareEsame(Esame e1, Esame e2);

#endif /* _ELEMENT_H */
```

"element.c":

```
#include "element.h"
#include <string.h>

int comparePaziente(Paziente p1, Paziente p2) {
	int result;
	result = strcmp(p1.cognome, p2.cognome);
	if (result!=0)
		return result;
	else
		return  strcmp(p1.nome, p2.nome);
}

int compareEsame(Esame e1, Esame e2) {
	if (e1.id<e2.id) return -1;
	else
		if (e1.id>e2.id) return 1;
		else {
			if (strcmp(e1.nomeEsame, e2.nomeEsame) != 0)
				return strcmp(e1.nomeEsame, e2.nomeEsame);
			else {
				if (e1.anno<e2.anno) return -1;
				else
					if (e1.anno>e2.anno)
						return 1;
					else
						return 0;
			}
		}
}
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 3A di Giovedì 9 Febbraio 2012 – tempo a disposizione 2h

"list.h"

```
#ifndef LIST_H
#define LIST_H

#include "element.h"

typedef struct list_element
{
    element value;
    struct list_element  *next;
} item;

typedef  item* list;

typedef  int boolean;

/* PRIMITIVE  */
list emptylist(void);
boolean empty(list);
list cons(element, list);
element head(list);
list tail(list);

void showlist(list l);
void freelist(list l);
//int member(element el, list l);

list insord_p(element el, list l);

#endif
```

"list.c":

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

#include "list.h"

/* OPERAZIONI PRIMITIVE */
list  emptylist(void)            /* costruttore lista vuota */
{
    return NULL;
}

boolean  empty(list l)     /* verifica se lista vuota */
{
    return (l==NULL);
}

list  cons(element e, list l)
{
  list t;       /* costruttore che aggiunge in testa alla lista */
  t=(list)malloc(sizeof(item));
  t->value=e;
```

```
  t->next=l;
  return(t);
}

element head(list l) /* selettore testa lista */
{
  if (empty(l)) exit(-2);
  else return (l->value);
}

list  tail(list l)         /* selettore coda lista */
{
  if (empty(l)) exit(-1);
  else return (l->next);
}


void showlist(list l) {
       element temp;
       if (!empty(l)) {
              temp = head(l);
              printf("%s %s %d\n", temp.cognome, temp.nome, temp.id);
              return showlist(tail(l));
       }
       else {
              printf("\n\n");
              return;
       }
}

void freelist(list l) {
       if (empty(l))
              return;
       else {
              freelist(tail(l));
              free(l);
       }
       return;
}

list insord_p(element el, list l) {
       list pprec, patt = l, paux;
       int trovato = 0;

       while (patt!=NULL && !trovato) {
              if (comparePaziente(el, patt->value)<0)
              //if (strcmp(el.cognome, patt->value.cognome)<0)
                     trovato = 1;
              else {
                     pprec = patt;
                     patt = patt->next;
              }
       }
       paux = (list) malloc(sizeof(item));
       paux->value = el;
       paux->next = patt;
       if (patt==l)
```

```
            return paux;
      else {
            pprec->next = paux;
            return l;
      }
}
```

"paz.h":

```
#ifndef PAZ
#define PAZ

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


list leggiPazienti(char* fileName);
void stampaLista(list v);

Esame * leggiEsami(char * fileName, int * dim);
void ordina(Esame * v, int n);

void ripetuti(list pazienti, Esame * vEsame, int dim);

#endif PAZ
```

"paz.c":

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


list leggiPazienti(char* fileName) {
      FILE * fp;
      Paziente temp;
      list result;

      result = emptylist();
      fp = fopen(fileName, "rt");
      if (fp == NULL) {
            printf("A problem occurred while opening file %s\n", fileName);
            system("pause");
            exit(-1);
      }
      else {
            while(fscanf(fp, "%s%s%d", temp.nome, temp.cognome, &temp.id) == 3)
                  result = insord_p(temp, result);
            fclose(fp);
      }
      return result;
}
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 3A di Giovedì 9 Febbraio 2012 – tempo a disposizione 2h

```
void stampaLista(list v) {
	showlist(v);
	return;
}


Esame * leggiEsami(char * fileName, int * dim) {
	FILE * fp;
	Esame * result = NULL;
	Esame temp;
	int i;

	*dim = 0;
	fp = fopen(fileName, "rt");
	if (fp == NULL) {
		printf("A problem occurred when opening the file: %s\n", fileName);
		system("pause");
		exit(-2);
	}
	else {
		while (fscanf(fp, "%d%s%d", &temp.id, temp.nomeEsame, &temp.anno) == 3)
			*dim = *dim + 1;
		rewind(fp);
		result = (Esame *) malloc(sizeof(Esame) * *dim);
		i = 0;
		while (fscanf(fp, "%d%s%d", &temp.id, temp.nomeEsame, &temp.anno) == 3) {
			result[i] = temp;
			i++;
		}
		fclose(fp);
	}
	return result;
}


void scambia(Esame *a, Esame *b) {
	Esame tmp = *a;
	*a = *b;
	*b = tmp;
}
// bubble sort
void ordina(Esame * v, int n) {
	int i, ordinato = 0;
	while (n>1 && !ordinato) {
		ordinato = 1;
		for (i=0; i<n-1; i++)
			if (compareEsame(v[i],v[i+1])>0) {
				scambia(&v[i],&v[i+1]);
				ordinato = 0;
			}
		n--;
	}
}


void ripetuti(list pazienti, Esame * vEsame, int dim) {
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 3A di Giovedì 9 Febbraio 2012 – tempo a disposizione 2h

```
	Esame current;
	int currentCount;
	int i;
	list temp;
	Paziente tempPaziente;

	ordina(vEsame, dim);
	if (dim>0) {
		current = vEsame[0];
		currentCount = 1;
		for (i=1; i<dim; i++) {
			if (current.id == vEsame[i].id && strcmp(current.nomeEsame,
vEsame[i].nomeEsame) == 0)
				currentCount++;
			else {
				if (currentCount>=3) {
					temp = pazienti;
					while(!empty(temp)) {
						tempPaziente = head(temp);
						if (tempPaziente.id == current.id)
							printf ("Il paziente %d %s %s ha effettuato
%d esami del tipo: %s.\n",
									tempPaziente.id,
tempPaziente.cognome, tempPaziente.nome,
									currentCount,
current.nomeEsame);
						temp = tail(temp);
					}
				}
				current = vEsame[i];
				currentCount = 1;
			}
		}
		if (currentCount>=3) {
			temp = pazienti;
			while(!empty(temp)) {
				tempPaziente = head(temp);
				if (tempPaziente.id == current.id)
					printf ("Il paziente %d %s %s ha effettuato %d esami del
tipo: %s.\n",
							tempPaziente.id, tempPaziente.cognome,
tempPaziente.nome,
							currentCount, current.nomeEsame);
				temp = tail(temp);
			}
		}
	}
	return;
}
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 3A di Giovedì 9 Febbraio 2012 – tempo a disposizione 2h

"main.c":

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

#include "element.h"
#include "list.h"
#include "paz.h"

int main(void) {

	list listaPaz;
	Esame * vEsami;
	int numEsami;
	int i;

	listaPaz = leggiPazienti("nomi.txt");
	stampaLista(listaPaz);

	vEsami = leggiEsami("esami.txt", &numEsami);
	for (i=0; i<numEsami; i++)
		printf("%d %s %d\n", vEsami[i].id, vEsami[i].nomeEsame, vEsami[i].anno);
	printf("\n\n");
	ordina(vEsami, numEsami);
	for (i=0; i<numEsami; i++)
		printf("%d %s %d\n", vEsami[i].id, vEsami[i].nomeEsame, vEsami[i].anno);

	ripetuti(listaPaz, vEsami, numEsami);

	free(vEsami);
	freelist(listaPaz);
	return 0;
}
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 3A di Giovedì 9 Febbraio 2012 – tempo a disposizione 2h

"nomi.txt":

```
Federico Chesani 23
Carlo Giannelli 46
Paola Mello 42
Stefano Bragaglia 34
```

"esami.txt":

```
42 glucosio 2009
23 glucosio 2008
23 glicemia 2009
46 ECG_sotto_sforzo 2008
23 trigliceridi 2011
42 glucosio 2012
46 ECG_sotto_sforzo 2009
46 ECG_sotto_sforzo 2010
23 radiografia_torace 2012
34 trigliceridi 2011
46 ECG_sotto_sforzo 2011
34 ECG_sotto_sforzo 2010
23 ECG_sotto_sforzo 2010
34 ECG_sotto_sforzo 2011
46 ECG_sotto_sforzo 2012
34 ECG_sotto_sforzo 2012
```