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

Prova d'Esame 1A di Giovedì 22 Dicembre 2011 – tempo a disposizione 2h

Prima di cominciare: si scarichi dal sito **`http://esamix.labx`** il file **`StartKit1A.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"*.

Un software per la gestione di archivi fotografici mantiene l'elenco delle foto in gestione tramite un file di testo, di nome "**`archivio.txt`**". In particolare, sulla prima riga di tale file memorizza quante foto sono presenti in archivio (tramite un intero). A seguire, per ogni foto memorizza i seguenti dati (una foto su ogni riga): id unico della foto (un intero); il nome dell'autore della foto (una stringa di al più 63 caratteri, senza spazi), e a seguire uno o più interi (fino a un massimo di dieci interi) separati da spazi, rappresentanti i codici identificativi dei tag associati alla singola foto. Ogni foto ha almeno sempre un tag, e ogni riga (compresa l'ultima) è sempre terminata da un carattere ';'.
In un secondo file di testo di nome "**`keyword.txt`**" sono memorizzati i tag. In dettaglio, su ogni riga del file viene memorizzato l'identificativo numerico del tag (un intero), e la sua corrispondente etichetta in linguaggio naturale (una stringa di al più 63 caratteri, senza spazi). Non è noto a priori quante righe siano memorizzate in questo file. Al fine di esempio, si vedano i file omonimi forniti nello StartKit.

Esercizio 1 – Strutture dati Foto, Tag e funzioni di lettura (mod. element.h e foto.h/foto.c)

Si definisca un'opportuna struttura dati **`Foto`**, al fine di rappresentare i dati relativi ad ogni fotografia, avendo cura di rappresentare i tag relativi ad ogni foto tramite un array dichiarato staticamente, e tenendo traccia in un apposito campo di tipo intero il numero di tag associati alla singola foto. Si definisca poi una struttura dati **`Tag`**, al fine di rappresentare il codice di ogni tag e la sua etichetta in linguaggio naturale, come descritto sopra.
Si definisca la funzione:

```
Foto * leggiFoto(char* fileName, int *dim);
```

che, ricevuto in ingresso il nome di un file di testo rappresentante l'archivio, restituisca un array di strutture dati di tipo **`Foto`** allocato dinamicamente (della dimensione minima necessaria), contenente tutte le informazioni presenti nel file il cui nome è passato come parametro. Tramite il parametro **`dim`** la funzione deve restituire la dimensione del vettore.
Si definisca la funzione:

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

che, ricevuto in ingresso il nome di un file di testo contenente l'elenco dei tag, restituisca una lista di strutture dati di tipo **`Tag`**.
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra.

Esercizio 2 – Ordinamento e stampa (moduli element.h/.c e foto.h/.c)

Il candidato definisca una procedura:

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

tale che, ricevuto un vettore di strutture dati di tipo Foto, e la dimensione di tale vettore, ordini tale vettore in ordine lessicografico in base all'autore della foto. In caso di più foto dello stesso autore, devono comparire prima le foto con un numero minore di tag.
Il candidato definisca poi una procedura:

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

Prova d'Esame 1A di Giovedì 22 Dicembre 2011 – tempo a disposizione 2h

```
void stampa(Foto * v, int dim, list tagList);
```

che, ricevuti in ingresso un vettore di strutture dati di tipo **Foto** e la dimensione di tale vettore, e una lista di strutture dati di tipo Tag, stampi a video l'elenco delle foto, avendo cura di sostituire ai codici numerici identificativi delle tag le corrispondenti etichette linguistiche, come specificate tramite il parametro tagList. A tal fine, si suggerisce al candidato di realizzare una funzione ausiliaria che, ricevuti in ingresso la lista dei tag ed un codice numerico, restituisca la stringa contenente l'etichetta linguistica associata a tale codice.
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra.

Esercizio 3 –Eliminazione dei duplicati (modulo foto.h/foto.c)

A causa di un problema nel software, a volte nell'archivio vengono duplicate le informazioni relative ad una stessa foto.
Si definisca una funzione:

```
Foto * duplicati(Foto * v, int dimV, int * dimR);
```

che, ricevuta come parametro un vettore di strutture dati di tipo **Foto** e la sua dimensione **dimV**, restituisca un nuovo vettore di strutture dati di tipo **Foto** e la sua dimensione **dimR** (non necessariamente della dimensione minima), dove siano stati inseriti i dati contenuti nel vettore passato in input, ma con le foto ripetute eliminate. Due foto sono ripetute se hanno lo stesso codice identificativo e lo stesso autore. In caso di due foto ripetute, nel vettore risultante dovrà essere messa la foto con il numero maggiore di tag.

Esercizio 4 – Stampa ordinata delle foto senza duplicati, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **main(…)** un programma che, usando le informazioni fornite tramite i file di esempio forniti nello StartKit, e le funzioni definite agli esercizi precedenti, stampi a video l'elenco ordinato (come specificato nell'esercizio 2) delle foto senza ripetizioni.

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 1A di Giovedì 22 Dicembre 2011 – tempo a disposizione 2h

"element.h":

```
#include <string.h>

#ifndef _ELEMENT_H
#define      _ELEMENT_H

#define DIM_AUTORE 64
#define DIM_TAG 64

typedef struct {
       int id;
       char autore[DIM_AUTORE];
       int numTag;
       int tag[10];
} Foto;

typedef struct {
       int tagId;
       char tagName[DIM_TAG];
} Tag;

typedef Tag element;

int compare(Foto f1, Foto f2);

#endif /* _ELEMENT_H */
```

"element.c":

```
#include "element.h"

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


int compare(Foto f1, Foto f2) {
       int temp;

       temp = strcmp(f1.autore, f2.autore);
       if (temp == 0)
              temp = f1.numTag - f2.numTag;
       return temp;
}
```

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

Prova d'Esame 1A di Giovedì 22 Dicembre 2011 – 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 freelist(list l) {
       if (empty(l))
              return;
       else {
              freelist(tail(l));
              free(l);
       }
       return;
}
```

"foto.h":

```
#ifndef FOTO
#define FOTO

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

Foto * leggiFoto(char * fileName, int* dim);
list leggiTag(char * fileName);

void ordina(Foto v[], int n);
void stampa(Foto * v, int dim, list tagList);

Foto * duplicati(Foto * v, int dimV, int * dimR);

#endif
```

"info.c":

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

#include "foto.h"

Foto * leggiFoto(char * fileName, int* dim) {
       Foto * result;
       FILE * fp;
```

```
	int i, cont;

	fp = fopen(fileName, "rt");
	if (fp == NULL) {
		printf("Errore durante l'apertura del file %s\n", fileName);
		system("pause");
		exit(-1);
	}
	else {
		fscanf(fp, "%d", dim);
		result = (Foto *) malloc(sizeof(Foto) * *dim);
		for (i=0; i<*dim; i++) {
			fscanf(fp, "%d%s", &(result[i].id), result[i].autore);
			cont = 0;
			while (fgetc(fp) != ';' && cont<10) {
				fscanf(fp, "%d", &(result[i].tag[cont]));
				cont++;
			}
			result[i].numTag = cont;
		}
		fclose(fp);
	}

	return result;
}



list leggiTag(char * fileName) {
	list result;
	FILE * fp;
	Tag temp;

	result = emptylist();
	fp = fopen(fileName, "r");
	if (fp == NULL) {
		printf("Errore durante l'apertura del file %s\n", fileName);
		system("pause");
		exit(-1);
	}
	else {
		while (fscanf(fp, "%d%s", &(temp.tagId), temp.tagName) == 2)
			result = cons(temp, result);
		fclose(fp);
	}

	return result;
}

char * getLabel(int tagId, list tagList) {

	while (!empty(tagList)) {
		if (tagId == head(tagList).tagId)
			return head(tagList).tagName;
		tagList = tail(tagList);
	}
	return NULL;
```

```
}


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

void stampa(Foto * v, int dim, list tagList) {
	int i, j;

	for (i=0; i<dim; i++) {
		printf("%d %s\t%d Tag: ", v[i].id, v[i].autore, v[i].numTag);
		for (j=0; j<v[i].numTag; j++)
			printf("%s ", getLabel(v[i].tag[j], tagList));
		printf("\n");
	}
}


Foto * duplicati(Foto * v, int dimV, int * dimR) {
	Foto * result;
	int i, j;
	int trovato;

	*dimR = 0;
	result = (Foto *) malloc(sizeof(Foto) * dimV);

	for (i=0; i<dimV; i++) {
		trovato = 0;
		for (j=0; j<*dimR && !trovato; j++) {
			if (v[i].id == result[j].id &&
				strcmp(v[i].autore, result[j].autore) == 0)
				trovato = 1;
		}
		if (!trovato) {
			result[*dimR] = v[i];
			(*dimR)++;
		}
		else {
			if (v[i].numTag>result[j].numTag)
				result[j] = v[i];
		}
```

```
        }
        return result;
}
```

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

Prova d'Esame 1A di Giovedì 22 Dicembre 2011 – tempo a disposizione 2h

"main.c":

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

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

int main() {

	// Test es. 1 e es. 2
	{
		Foto * v;
		int dim;
		list tagList;

		v = leggiFoto("archivio.txt", &dim);
		tagList = leggiTag("keyword.txt");
		stampa(v, dim, tagList);

		ordina(v, dim);
		printf("\n\n");
		stampa(v, dim, tagList);
		printf("\n\n\n");

		free(v);
		freelist(tagList);
	}

	// Test es. 3  e  es. 4
	{
		Foto * v;
		Foto * v_dup;
		int dim;
		int dim_dup;
		list tagList;

		v = leggiFoto("archivio.txt", &dim);
		tagList = leggiTag("keyword.txt");
		ordina(v, dim);
		stampa(v, dim, tagList);

		v_dup = duplicati(v, dim, &dim_dup);
		ordina(v_dup, dim_dup);
		printf("\n\n");
		stampa(v_dup, dim_dup, tagList);

		free(v);
		free(v_dup);
		freelist(tagList);
	}
	system("pause");

	return 0;
}
```

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

Prova d'Esame 1A di Giovedì 22 Dicembre 2011 – tempo a disposizione 2h

"archivio.txt":

```
8
146 Federico 12 45 65;
149 Federico 12 45 66;
151 Carlo 23;
167 Carlo 12 23 78;
168 Carlo 12 23 78;
213 Federico 12 45 65;
214 Federico 12 45 66;
146 Federico 12;
```

"keyword.txt":

```
12 vacanze
45 famiglia
65 francesco
66 elena
23 lavoro
78 calcetto
```