Fondamenti di Informatica T-1, 2012/2013 – Modulo 2

Prova d'Esame 5A di Giovedì 11 Luglio 2013 – tempo a disposizione 2h

Prima di cominciare: si scarichi dal sito **`http://esamix.labx`** il file **`StartKit5A.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 noto sito web di vendita libri online memorizza tutte le transazioni effettuate in un mese in un file di testo. Le informazioni relative ad ogni transazione sono memorizzate su una riga: in particolare, nell'ordine, un **identificativo** unico della transazione (un intero); separato da uno spazio il **titolo** del libro (una stringa di al più 1023 caratteri utili, contenente anche spazi, ma non contenente il carattere speciale '*'); poi, separato dal carattere '*' il nome e il cognome del **cliente** (una stringa unica di al più 1023 caratteri, contenente spazi ma non '*'); infine, separato dal carattere '*' un carattere singolo che identifica lo **stato** della spedizione: 'C' se consegnato correttamente, 'P' se è stato perduto il libro durante la spedizione e non è stato ancora rispedito, 'R' se è stato perduto ed è già stato rispedito al cliente. Nella prima riga del file, a scopo di controllo, è memorizzato un solo intero che rappresenta il numero di transazioni registrate in tale file. A titolo di esempio, si veda il file "**`transazioni.txt`**" fornito nello StartKit.

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

Si definisca un'opportuna struttura dati **`Transazione`**, al fine di rappresentare i dati relativi a una singola transazione: in particolare si dovrà tenere traccia del titolo del libro, del nome del cliente, dell'id della transazione, e dello stato della spedizione.

Si definisca la funzione:

```
list leggiT(char * nomeFile);
```

che ricevuto come parametro il nome di un file contenente le transazioni, legga i dati ivi contenuti e restituisca come risultato una lista di strutture dati **`Transazione`** istanziata con i dati letti. In caso di errore nell'apertura del file, la funzione deve stampare a video un messaggio di errore, e deve restituire una lista vuota. In caso di apertura e lettura dei dati con successo, la funzione deve effettuare comunque un ulteriore controllo: il numero di transazioni lette e memorizzate nella lista deve essere pari al numero intero specificato nella prima riga. In caso contrario, la funzione deve restituire una lista vuota poiché i dati letti non sono affidabili. Si noti che lo StartKit contiene il modulo "**`read.h/.c`**" con la funzione **`readField(…)`** che può essere usata dal candidato qualora necessario/utile.

Si definisca poi la procedura:

```
void showlist(list l1);
```

che, ricevuta in ingresso una lista **`l1`** di strutture dati di tipo **`Transazione`** , stampi a video il contenuto della lista.

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, 2012/2013 – Modulo 2

Prova d'Esame 5A di Giovedì 11 Luglio 2013 – tempo a disposizione 2h

Esercizio 2 – Transazioni relative a un singolo cliente (moduli element.h/.c e dati.h/.c)

Si definisca una funzione:

```
list ordina(list t, char * nome, int * dim);
```

che ricevuta in ingresso una lista di strutture dati di tipo **`Transazione`**, restituisca in uscita una nuova lista di strutture dati di tipo **`Transazione`**, ordinata in ordine lessicografico in base al titolo del libro acquistato. La nuova lista dovrà contenere solo le transazioni relative al cliente specificato tramite il parametro **`nome`**. Inoltre, tramite il parametro **`dim`** passato per riferimento, la funzione dovrà restituire il numero di elementi memorizzati nella lista.

Si definisca una funzione:

```
int * elenca(list t, int dim);
```

che ricevuta in ingresso una lista di strutture dati di tipo **`Transazione`**, e il numero di elementi ivi contenuti tramite il parametro **`dim`**, restituisca un array di interi allocato dinamicamente, che contenga tutti gli id di transazione presenti nella lista, ordinato in senso crescente in base all'identificatore. Per ordinare il vettore, il candidato deve utilizzare l'algoritmo naiveSort visto a lezione.

Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra.

Esercizio 3 –Identificazione dei tentativi di truffe (modulo dati.h/dati.c)

Il candidato definisca una procedura:

```
void truffe(list t);
```

che riceva in ingresso una lista di strutture dati di tipo **`Transazione`**. La procedura deve stampare a video tutti i clienti per i quali esiste il sospetto di una possibile truffa. Un cliente è a sospetto di truffa se esistono a suo nome almeno due transazioni la cui spedizione sia in stato "perduto" ('P') e almeno una transazione con spedizione in stato "rispedito" ('R'). Inoltre, per ognuno dei clienti a sospetto di truffa, la procedura stampi a video anche tutti gli identificatori di transazione, in ordine crescente di id.
Attenzione: la procedura deve stampare ogni cliente a sospetto di truffa una e una sola volta. Inoltre il candidato abbia cura di non generare memory leaking, tramite deallocazione opportuna delle liste e degli array di appoggio eventualmente creati.
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle procedure di cui sopra.

Esercizio 4 – Stampa di tutte le transazioni e delle possibili truffe, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che provveda a leggere dal file fornito nello StartKit l'elenco delle transazioni e le stampi a video. Di seguito poi stampi i clienti e gli id di transazione che sono a sospetto di truffa, come spiegato nell'Es. 3.

Il candidato abbia cura di de-allocare tutta la memoria allocata dinamicamente, ivi compresa la memoria allocata per le liste. Si noti che potrebbe essere necessario deallocare della memoria anche all'interno delle singole funzioni definite negli esercizi precedenti.

Fondamenti di Informatica T-1, 2012/2013 – Modulo 2

Prova d'Esame 5A di Giovedì 11 Luglio 2013 – tempo a disposizione 2h

"element.h":

```
#ifndef _ELEMENT_H
#define _ELEMENT_H

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

#define DIM 1024

typedef struct {
	char titolo[DIM];
	char cliente[DIM];
	int id;
	char stato;
} Transazione;

typedef Transazione element;

int compare(Transazione t1, Transazione t2);

#endif
```

"element.c":

```
#include "element.h"

int compare(Transazione t1, Transazione t2) {
	return strcmp(t1.titolo, t2.titolo);
}
```

Fondamenti di Informatica T-1, 2012/2013 – Modulo 2

Prova d'Esame 5A di Giovedì 11 Luglio 2013 – 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;
}


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

       while (patt!=NULL && !trovato) {
              if (compare(el, patt->value)<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;
       }
}
```

Fondamenti di Informatica T-1, 2012/2013 – Modulo 2

Prova d'Esame 5A di Giovedì 11 Luglio 2013 – tempo a disposizione 2h

"dati.h":

```
#ifndef _DATI
#define _DATI

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

list leggiT(char * nomeFile);
void showlist(list l);

list ordina(list t, char * nome, int * dim);
int * elenca(list t, int dim);

void truffe(list t);

#endif
```

"pagamenti.c":

```
#include "dati.h"

list leggiT(char * nomeFile) {
	FILE * fp;
	list result;
	int num, count, ok;
	Transazione temp;

	result = emptylist();
	count = 0;
	fp = fopen(nomeFile, "rt");
	if (fp == NULL) {
		printf("Errore nell'apertura del file %s\n", nomeFile);
		return result;
	}
	else {
		fscanf(fp, "%d", &num);
		do {
			ok = (fscanf(fp, "%d", &(temp.id))==1);
			if (ok) ok = fgetc(fp); // elimino lo spazio di separazione
			if (ok) ok = readField(temp.titolo, '*', fp);
			if (ok) ok = readField(temp.cliente, '*', fp);
			if (ok) ok = (fscanf(fp, "%c", &(temp.stato))==1);
			if (ok) {
				result = cons(temp, result);
				count++;
			}
		} while (ok);
		fclose(fp);
		if (count != num) {
			freelist(result);
			return emptylist();
		}
		else
			return result;
```

```
	}
}

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

list ordina(list t, char * nome, int * dim) {
	list result;

	result = emptylist();
	*dim = 0;
	while (!empty(t)) {
		if (strcmp(head(t).cliente, nome) == 0) {
			result = insord_p(head(t), result);
			*dim = *dim + 1;
		}
		t = tail(t);
	}
	return result;
}

void scambia(int *a, int *b) {
	int tmp = *a;
	*a = *b;
	*b = tmp;
}
int trovaPosMax(int v[], int n){
	int i, posMax=0;
	for (i=1; i<n; i++)
		if (v[posMax]<v[i])
			posMax=i;
	return posMax;
}

void naiveSort(int v[], int n){
	int p;
	while (n>1) {
		p = trovaPosMax(v,n);
		if (p<n-1)
			scambia(&v[p],&v[n-1]);
		n--;
	}
}

int * elenca(list t, int dim) {
	int * result;
```

```
	int i;
	result = (int*) malloc(sizeof(int) * dim);
	for (i=0; i<dim && !empty(t); i++) {
		result[i] = head(t).id;
		t = tail(t);
	}
	naiveSort(result, dim);
	return result;
}

int member(char * nome, list t) {
	if (empty(t)) return 0;
	else
		if (strcmp(nome, head(t).cliente)==0) return 1;
		else return member(nome, tail(t));
}

void truffe(list t) {
	list temp;
	int countP;
	int countR;
	char * nome;
	list truffa;
	list truffaTemp;
	int dim;
	int * v;
	int i;

	temp = t;
	while (!empty(temp)) {
		nome = head(temp).cliente;
		if (!member(nome, tail(temp))) {
			countP = 0;
			countR = 0;
			truffa = ordina(t, nome, &dim);
			truffaTemp = truffa;
			while(!empty(truffaTemp)) {
				if (head(truffaTemp).stato == 'P')
					countP++;
				if (head(truffaTemp).stato == 'R')
					countR++;
				truffaTemp = tail(truffaTemp);
			}
			if (countP>=2 && countR>=1) {
				printf("il cliente %s e' a sospetto di truffa.\n", nome);
				printf("Transazioni: ");
				v = elenca(truffa, dim);
				for (i=0; i<dim; i++)
					printf("%d ", v[i]);
				printf("\n\n");
				free(v);
			}
			freelist(truffa);
		}
		temp = tail(temp);
	}
}
```

Fondamenti di Informatica T-1, 2012/2013 – Modulo 2

Prova d'Esame 5A di Giovedì 11 Luglio 2013 – tempo a disposizione 2h

"main.c":

```
#include "element.h"
#include "list.h"
#include "dati.h"


int main() {
	{ // TEST Es. 1
		list l1;

		l1 = leggiT("transazioni.txt");
		showlist(l1);
		freelist(l1);
	}

	{ // TEST Es. 2
		list l1, l2;
		int * v;
		int dim, i;

		l1 = leggiT("transazioni.txt");
		l2 = ordina(l1, "Federico Chesani", &dim);
		showlist(l2);
		v = elenca(l2, dim);
		for (i=0; i<dim; i++)
			printf("%d ", v[i]);
		printf("\n\n");
		freelist(l1);
		freelist(l2);
	}

	{ // TEST Es. 3 & 4
		list l1;
		int * v;
		int dim, i;

		l1 = leggiT("transazioni.txt");
		showlist(l1);
		truffe(l1);

		freelist(l1);

	}

	return 0;
}
```

Fondamenti di Informatica T-1, 2012/2013 – Modulo 2

Prova d'Esame 5A di Giovedì 11 Luglio 2013 – tempo a disposizione 2h

"transazioni.txt":

```
7
1450 Harry Potter and the Deathly Hallows;Federico Chesani;C
1452 Song of Ice and Fire: A Game of Thrones;Federico Chesani;P
2356 Peppa Pig: Flying a Kite and other stories;Carlo Giannelli;C
2357 Favole Della Buonanotte;Carlo Gianneli;C
1455 Song of Ice and Fire: A Clash of Kings;Federico Chesani;R
1453 Song of Ice and Fire: A Storm of Swords;Federico Chesani;R
1458 C: How to Program (6th Edition);Federico Chesani;P
```