Fondamenti di Informatica T-1, 2016/2017 – Modulo 2

Prova d'Esame 6A di Giovedì 14 Settembre 2017 – tempo a disposizione 2h

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 famoso produttore di biciclette da corsa permette ai suoi clienti di acquistare le biciclette direttamente tramite una applicazione web-based. Ogni richiesta di acquisto fatta online viene memorizzata su un file di testo, che poi regolarmente viene preso in consegna dal magazzino per spedire le biciclette ordinate.
Ogni singolo ordine viene memorizzato sul file di testo su una riga a sé stante, con le seguenti informazioni: innnazitutto la **data** dell'ordine (nel formato "AAAA/MM/GG", tre interi); a seguire, separato da uno spazio, l'identificatore unico del **cliente** (un intero); separato da uno spazio, il codice unico del **modello** di bicicletta (una stringa di esattamente 6 caratteri utili, senza spazi); separato da uno spazio, il **prezzo** (un float); infine, ancora separato tramite uno spazio dal campo precedente, un **testo** libero inserito dal cliente (una stringa di al più 511 caratteri utili, contenente spazi, ma non caratteri di fine linea). Ogni riga, compresa l'ultima, è sempre terminata da un carattere 'newline'.

Esercizio 1 – Struttura dati Ordine, e funzioni di lett./scritt. (mod. element.h e ordini.h/.c)

Si definiscano le strutture dati **`Data`** e **`Ordine`**, al fin di memorizzare i dati relativi ad un singolo ordine, cioè data, id del cliente, codice della bici, prezzo e testo libero.

Si definisca la funzione:

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

che, ricevuto in ingresso il nome di un file, legga da tale file tutte le informazioni relative agli ordini di biciclette, e le restituisca tramite una lista di strutture dati di tipo **`Ordine`**. Qualora la funzione incontri dei problemi nell'apertura del file, la funzione dovrà stampare un messaggio di errore a video, e restituire una lista vuota. Si assuma per semplicità che il file sia sempre ben formato secondo la descrizione data sopra.

Si definisca la procedura:

```
void stampaOrdini(list v);
```

che, ricevuta in ingressa una lista di strutture dati di tipo **`Ordine`**, stampi a video le informazioni riguardo gli ordini memorizzati nella lista. La funzione dovrà essere implementata in maniera ricorsiva.

Nel sistema online capita spesso che clienti sbadati ripetano più volte uno stesso ordine, quando la loro intenzione era ovviamente fare un ordine solo. A tal scopo, si rende necessario filtrare via gli ordini ripetuti. Il candidato definisca una funzione:

```
int contaOccorrenze(list ordini, Ordine unOrdine)
```

che, ricevuta in ingresso una lista di strutture dati di tipo **`Ordine`**, ed un ordine ben preciso, conti il numero di occorrenze di quell'ordine nella lista. Due ordini si considerano identici (e quindi ripetuti) se sono stati fatti nella stessa data, dallo stesso cliente, per lo stesso modello di bici (quindi a prescindere dal campo testo libero e dal prezzo). Si definisca poi la funzione:

```
list filtra(list ordini);
```

che, ricevuta in ingresso una lista di strutture dati di tipo **`Ordine`**, restituisca in uscita una nuova lista dove gli ordini ripetuti siano stati eliminati. Nessuno degli ordini ripetuti deve comparire nella lista fornita come risultato. In altre parole, verranno inseriti nella lista risultato solo quegli ordini che comapiono una sola volta nella lista specificata in ingresso.
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra, avendo cura di deallocare la memoria, se necessario.

Fondamenti di Informatica T-1, 2016/2017 – Modulo 2

Prova d'Esame 6A di Giovedì 14 Settembre 2017 – tempo a disposizione 2h

Esercizio 2 – Vettori e Ordinamento (moduli element.h/.c e ordine.h/.c)

Si definisca la funzione:

```
Ordine * perCliente(list ordini, int cliente, int * dim);
```

che, ricevuta in ingresso una lista di strutture dati di tipo **`Ordine`**, e l'identificatore di un cliente, restituisca un vettore di strutture dati **`Ordine`** allocato dinamicamente. Il vettore dovrà contenere tutti gli ordini fatti dal cliente specificato come parametro. Tramite il parametro **`dim`** la funzione dovrà restituire la dimensione di tale vettore.

Si definisca poi una procedura:

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

che, ricevuto in ingresso un vettore di strutture dati di tipo **`Ordine`**, e la sua dimensione **`dim`**, ordini il vettore in senso crescente in base al prezzo dell'ordine effettuato. A tal fine, si utilizzi l'algoritmo Bubble Sort presentato a lezione.

Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra, avendo cura di deallocare la memoria, se necessario.

Esercizio 3 – Individuazione del cliente migliore (modulo ordini.h/ordini.c)

L'azienda vuole dare la precedenza agli ordini del suo cliente migliore. Il cliente "migliore" è quello che, sommati tutti gli ordini da lui effettuati, ha l'importo totale maggiore di tutti. A tal scopo, si definisca una funzione:

```
int migliore(list ordini);
```

che ricevuta come parametro una lista di strutture dati di tipo **`Ordine`**, restituisca il codice identificativo del miglior cliente. Tale cliente è quello la cui somma di tutti gli importi di tutti gli ordini da lui effettuati risulti essere maggiore delle somme relative agli altri clienti.

Esercizio 4 Stampa degli ordini effettuati dal cliente migliore, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che legga dai file i dati relativi agli ordini, depuri la lista degli ordini da quelli ripetuti, e poi determini il cliente migliore. Il programma stampi a video, in modo ordinato, gli ordini del cliente migliore.

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, 2016/2017 – Modulo 2

Prova d'Esame 6A di Giovedì 14 Settembre 2017 – tempo a disposizione 2h

"element.h":

```
#ifndef _ELEMENT_H
#define _ELEMENT_H

#include <string.h>

#define DIM 512

typedef struct {
	int anno;
	int mese;
	int giorno;
} Data;

typedef struct {
	Data data;
	int cliente;
	char modello[7];
	float prezzo;
	char testo[DIM];
} Ordine;

typedef Ordine element;

int equals(Ordine o1, Ordine o2);
int compare(Ordine o1, Ordine o2);

#endif
```

"element.c":

```
#include "element.h"

int equals(Ordine o1, Ordine o2) {
	if (
		o1.data.anno == o2.data.anno
		&& o1.data.mese == o2.data.mese
		&& o1.data.giorno == o2.data.giorno
		&& o1.cliente == o2.cliente
		&& strcmp(o1.modello, o2.modello) == 0
		)
		return 1;
	else
		return 0;
}

int compare(Ordine o1, Ordine o2) {
	if (o1.prezzo<o2.prezzo)
		return 1;
	else
		if (o1.prezzo>o2.prezzo)
			return -1;
		else
			return 0;
}
```

Fondamenti di Informatica T-1, 2016/2017 – Modulo 2

Prova d'Esame 6A di Giovedì 14 Settembre 2017 – 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);
}

int member(element el, list l) {
       int result = 0;
       while (!empty(l) && !result) {
              result = (equals(el, head(l))==0);
              if (!result)
                     l = tail(l);
       }
       return result;
}



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

Fondamenti di Informatica T-1, 2016/2017 – Modulo 2

Prova d'Esame 6A di Giovedì 14 Settembre 2017 – tempo a disposizione 2h

"ordini.h":

```
#ifndef _ORDINI_H
#define _ORDINI_H

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

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

// Es. 1
list leggiOrdini(char * fileName);
void stampaOrdini(list v);
list filtra(list ordini);

// Es. 2
Ordine * perCliente(list ordini, int cliente, int * dim);
void ordina(Ordine * v, int dim);

// Es. 3
int migliore(list ordini);

#endif
```

"ordini.c":

```
#include "ordini.h"
// Es. 1
list leggiOrdini(char * fileName) {
	list result;
	FILE * fp;
	Ordine temp;
	int i;
	char ch;

	result = emptylist();
	fp = fopen(fileName, "rt");
	if (fp==NULL) {
		printf("Errore durante l'apertura del file %s\n", fileName);
	}
	else {
		while (fscanf(fp, "%d/%d/%d %d %s %f",
				&(temp.data.anno),
				&(temp.data.mese),
				&(temp.data.giorno),
				&(temp.cliente),
				temp.modello,
				&(temp.prezzo) )
				== 6) {
			fgetc(fp); // butto via il separatore
			i=0;
			do {
				ch =fgetc(fp);
				if (ch!='\n' && i<DIM-1) {
					temp.testo[i] = ch;
					i++;
```

```
                        }
                        temp.testo[i] = '\0';
                } while (ch!='\n' && i<DIM-1);
                result = cons(temp, result);
        }
        fclose(fp);
    }
    return result;
}


void stampaOrdini(list v) {
    Ordine temp;
    if (empty(v)) {
        printf("\n\n");
        return;
    }
    else {
        temp = head(v);
        printf("%d/%d/%d %d %s %.2f %s\n",
                temp.data.anno,
                temp.data.mese,
                temp.data.giorno,
                temp.cliente,
                temp.modello,
                temp.prezzo,
                temp.testo);
        stampaOrdini(tail(v));
        return;
    }
}

int contaOccorrenze(list ordini, Ordine unOrdine) {
    int result = 0;
    while (!empty(ordini)) {
        if (equals(unOrdine, head(ordini)))
            result++;
        ordini = tail(ordini);
    }
    return result;
}

list filtra(list ordini) {
    list result;
    list temp;
    Ordine unOrdine;

    result = emptylist();
    temp = ordini;
    while (!empty(temp)) {
        unOrdine = head(temp);
        if (contaOccorrenze(ordini, unOrdine) == 1) {
            result = cons(unOrdine, result);
        }
        temp = tail(temp);
    }
```

```
	return result;
}


// Es. 2
Ordine * perCliente(list ordini, int cliente, int * dim) {
	Ordine * result;
	list temp;
	Ordine unOrdine;

	*dim = 0;
	temp = ordini;

	while(!empty(temp)) {
		unOrdine = head(temp);
		if (unOrdine.cliente == cliente) {
			*dim = *dim + 1;
		}
		temp = tail(temp);
	}
	result = (Ordine *) malloc(sizeof(Ordine) * *dim);
	*dim = 0;
	while(!empty(ordini)) {
		unOrdine = head(ordini);
		if (unOrdine.cliente == cliente) {
			result[*dim] = unOrdine;
			*dim = *dim + 1;
		}
		ordini = tail(ordini);
	}
	return result;
}

void scambia(Ordine *a, Ordine *b) {
	Ordine tmp = *a;
	*a = *b;
	*b = tmp;
}
void bubbleSort(Ordine 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 ordina(Ordine * v, int dim) {
	bubbleSort(v, dim);
}


// Es. 3
int migliore(list ordini) {
```

```
	int clienteRecord;
	int clienteTemp;
	float totRecord;
	float tempRecord;
	int dim;
	int i;
	Ordine * v;
	list temp;

	temp = ordini;
	if (!empty(temp)) {
		clienteRecord = head(temp).cliente;
		totRecord = 0.0f;
		v = perCliente(ordini, clienteRecord, &dim);
		for (i=0; i<dim; i++)
			totRecord = totRecord + v[i].prezzo;
		free(v);
		temp = tail(temp);

		while (!empty(temp)) {
			clienteTemp = head(temp).cliente;
			tempRecord = 0.0f;
			v = perCliente(ordini, clienteTemp, &dim);
			for (i=0; i<dim; i++)
				tempRecord = tempRecord + v[i].prezzo;
			free(v);
			if (tempRecord > totRecord) {
				clienteRecord = clienteTemp;
				totRecord = tempRecord;
			}
			temp = tail(temp);
		}
		return clienteRecord;
	}
	else {
		printf("Si e' verificato un errore: la lista passat e' vuota!");
		exit(-1);
	}
}
```

Fondamenti di Informatica T-1, 2016/2017 – Modulo 2

Prova d'Esame 6A di Giovedì 14 Settembre 2017 – tempo a disposizione 2h

"main.c":

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

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

int main() {
	// Es. 1
	{
		list ordini;
		list senzaRip;
		ordini = leggiOrdini("ordini.txt");
		stampaOrdini(ordini);

		senzaRip = filtra(ordini);
		stampaOrdini(senzaRip);
		freelist(ordini);
		freelist(senzaRip);
	}

	// Es. 2
	{
		list ordini;
		list senzaRip;
		Ordine * v;
		int dim;
		int i;

		ordini = leggiOrdini("ordini.txt");
		senzaRip = filtra(ordini);
		v = perCliente(senzaRip, 34, &dim);
		for (i=0; i<dim; i++) {
			printf("%d/%d/%d %d %s %.2f %s\n",
				v[i].data.anno,
				v[i].data.mese,
				v[i].data.giorno,
				v[i].cliente,
				v[i].modello,
				v[i].prezzo,
				v[i].testo);
		}
		ordina(v, dim);
		for (i=0; i<dim; i++) {
			printf("%d/%d/%d %d %s %.2f %s\n",
				v[i].data.anno,
				v[i].data.mese,
				v[i].data.giorno,
				v[i].cliente,
				v[i].modello,
				v[i].prezzo,
				v[i].testo);
		}
		freelist(ordini);
```

```
            freelist(senzaRip);
            free(v);
      }

      // Es. 3 && 4
      {
            list ordini;
            list senzaRip;
            Ordine * v;
            int dim;
            int i;
            int ilMigliore;

            ordini = leggiOrdini("ordini.txt");
            senzaRip = filtra(ordini);
            ilMigliore = migliore(senzaRip);
            v = perCliente(senzaRip, ilMigliore, &dim);
            ordina(v, dim);
            printf("Il cliente migliore: \n");
            for (i=0; i<dim; i++) {
                  printf("%d/%d/%d %d %s %.2f %s\n",
                        v[i].data.anno,
                        v[i].data.mese,
                        v[i].data.giorno,
                        v[i].cliente,
                        v[i].modello,
                        v[i].prezzo,
                        v[i].testo);
            }
            freelist(ordini);
            freelist(senzaRip);
            free(v);
      }

      return 0;
}
```

"ordini.txt":

```
2017/09/01 42 E807FA 2100.00 io vorrei la pedivella da 175, perche' sono molto alto. e' possibile?
2017/09/02 12 E807UR 2400.00 vorrei la cassetta 12-30 al posto della 12-28 standard
2017/09/02 16 E807ZH 2350.00 posso avere questo modello nel colore rosso fuoco?
2017/09/02 12 E807UR 2400.00 vorrei la cassetta 12-31 al posto della 12-28 standard
2017/09/03 34 E806I4 10250.00 non sono convinto del freno a disco... voi cosa consigliate?
2017/09/02 12 E807UR 2400.00 scusa la confuzione, ci ho pensato, vorrei la cassetta 11-34 al posto della 12-28 standard
2017/09/02 12 E807UR 2400.00 ma e' partito il mio ordine? non capisco...
2017/09/02 14 E807UR 2400.00 tra quanti giorni e' prevista la consegna?
2017/09/04 34 E806I4 10300.00 questa e' un regalo per il mio bambino
```