Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 3A di Giovedì 13 Febbraio 2015 – 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 supermercato vuole tenere traccia delle abitudini di acquisto dei propri clienti. A tal scopo memorizza i dati di ogni spesa effettuata in un file di testo, così formato: sulla prima riga vi è il numero di spese registrate nel file, sotto forma di un intero. A seguire ogni spesa è registrata su una riga (una sola spesa per ogni riga), nel seguente modo: all'inizio l'identificatore unico del **cliente**, nella forma di una stringa (di al più 2047 caratteri utili, senza spazi); a seguire, separata da uno spazio, la **data** in cui è stata effettuata la spesa, nella forma di gg/mm/yyyy (tre interi); infine ancora separati da spazio, l'elenco degli **oggetti** acquistati, ogni oggetto composto dalla coppia **descrizione** (una stringa di al più 2047 caratteri utili, senza spazi) e **costo** (un float), separati da uno spazio. Non è noto a priori quanti oggetti (cioè quante coppie descrizione/costo) siano memorizzate su ogni riga. E' certo invece che ogni riga (compresa l'ultima) termina con un carattere "new line".

Si veda a titolo di esempio il file di testo fornito nello StartKit.

Esercizio 1 – Strutture dati Oggetto, Spesa, e funzioni di lett./scritt. (mod. element.h e spesa.h/.c)

Si definisca un'opportuna struttura dati **`Oggetto`** nel file "element.h", atta a rappresentare un oggetto acquistato, cioè la sua descrizione e il suo costo. Si definisca poi nel file "spesa.h" una opportuna struttura dati **`Spesa`**, al fine di rappresentare i dati relativi ad una singola spesa, cioè l'id del cliente, la data, e l'elenco degli oggetti acquistati. Per rappresentare l'elenco di oggetti acquistati, il candidato usi (obbligatoriamente) una lista di strutture dati di tipo **`Oggetto`**.

Si definisca la funzione:

```
Spesa * leggiSpese(char * nomeFile, int * dim);
```

che, ricevuto in ingresso il nome di un file contenente l'elenco delle spese (strutturato come sopra), legga da tale file le informazioni e le restituisca tramite un vettore di strutture dati di tipo **`Spesa`** allocato dinamicamente della dimensione minima necessaria. Tramite l'intero **`dim`** passato per riferimento la funzione deve restituire la dimensione del vettore allocato. In caso di errore nell'apertura del file, la funzione deve restituire un puntatore a NULL, e **`dim`** pari al valore "meno tre".

Si definisca la procedura:

```
void stampaSpese(Spesa * v, int dim);
```

che, ricevuto in ingresso un vettore di strutture dati di tipo **`Spesa`**, e la sua dimensione **`dim`**, stampi a video il contenuto di tali strutture dati.

Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra. Il candidato abbia cura anche di deallocare eventuale memoria allocata dinamicamente nei test.

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 3A di Giovedì 13 Febbraio 2015 – tempo a disposizione 2h

Esercizio 2 – Valutazione delle spese, e ordinamento (modulo spesa.h/.c)

Il candidato definisca una funzione:

```
float totale(list elenco);
```

che, ricevuto in ingresso una lista di strutture dati di tipo **`Oggetto`**, calcoli in modo **RICORSIVO** la spesa totale per acquistare gli oggetti indicati nella lista **`elenco`**.

Il candidato definisca poi una funzione:

```
float totalePerId(Spesa * v, int dim, char * id);
```

che, ricevuto in ingresso un vettore di strutture dati di tipo **`Spesa`** e la sua dimensione, e una stringa rappresentante l'identificatore unico di un cliente, calcoli in modo **RICORSIVO** la somma di tutte le spese sostenute dal cliente specificato.

Il candidato definisca poi una procedura:

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

che, ricevuti in ingresso un vettore di strutture dati di tipo **`Spesa`** e la dimensione di tale vettore, ordini il vettore secondo il seguente criterio: in ordine crescente lessicografico in base all'identificatore del cliente; a parità di cliente, in maniera crescente in base all'importo speso in quella specifica spesa. A tal fine, il candidato utilizzi l'algoritmo di ordinamento "QuickSort" 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. Il candidato abbia cura anche di deallocare eventuale memoria allocata dinamicamente nei test.

Esercizio 3 – Riassunto delle spese sostenute da ogni cliente (modulo spesa.h/spesa.c)

Il supermercato vuole avere per ogni cliente un riassunto completo di tutte le spese da lui sostenute.
Si definisca una funzione:

```
Spesa * riassunto(Spesa * v, int dim, int * dimR);
```

che, ricevuto in ingresso un vettore di strutture dati di tipo **`Spesa`** e la sua dimensione, restituisca un vettore allocato dinamicamente ancora di strutture dati di tipo **`Spesa`**, della dimensione minima necessaria. Tramite il parametro **`dimR`** la funzione deve restituire la dimensione del vettore risultato. Il vettore risultato dovrà contenere una e una sola struttura dati di tipo **`Spesa`** per ogni singolo cliente; la data di tale spesa dovrà valere una data qualunque tra le spese effettuate da quel cliente; l'elenco delle spese effettuate dovrà contenere tutte le spese effettuate da quel cliente e presenti nel vettore di ingresso **`v`**.

Esercizio 4 – Stampa dei clienti, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che, usando le informazioni fornite tramite il file di esempio fornito nello StartKit e le funzioni definite agli esercizi precedenti, stampi a video l'elenco dei clienti con l'elenco di tutti gli oggetti acquistati in tutte le spese (tramite la funzione di cui al punto 3), e poi per ognuno di essi stampi il totale speso (tramite le funzioni di cui al punto 2).

Al termine del programma, il candidato abbia cura di de-allocare tutta la memoria allocata dinamicamente, ivi compresa la memoria allocata per le liste, se non in structure sharing (in tal caso, si segnali la situazione con un breve commento nel codice).

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 3A di Giovedì 13 Febbraio 2015 – tempo a disposizione 2h

"element.h":

```
#ifndef _ELEMENT_H
#define _ELEMENT_H

#define DIM 2048

typedef struct {
	char desc[DIM];
	float costo;
} Oggetto;

typedef Oggetto element;


#endif
```

"element.c":

Non c'è in questo compito…

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 3A di Giovedì 13 Febbraio 2015 – 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;
}
```

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 3A di Giovedì 13 Febbraio 2015 – tempo a disposizione 2h

"spesa.h":

```
#ifndef _SPESA_H
#define _SPESA_H

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

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

typedef struct {
	char id[DIM];
	int dd;
	int mm;
	int yyyy;
	list elenco;
} Spesa;

// es. 1
Spesa * leggiSpese(char * nomeFile, int * dim);
void stampaSpese(Spesa * v, int dim);

// es. 2
float totale(list elenco);
float totalePerId(Spesa * v, int dim, char * id);
void ordina(Spesa * v, int dim);

// es. 3
Spesa * riassunto(Spesa * v, int dim, int * dimR);

#endif
```

"spesa.c":

```
#include "spesa.h"

// Es. 1

Spesa * leggiSpese(char * nomeFile, int * dim) {
	FILE * fp;
	Spesa * result;
	int i;
	char ch;
	Oggetto temp;

	fp = fopen(nomeFile, "rt");
	if (fp==NULL) {
		result = NULL;
		*dim = -3;
		return result;
	}
	else {
		fscanf(fp, "%d", dim);
```

```
		result = (Spesa*) malloc(sizeof(Spesa) * *dim);
		for (i=0; i<*dim; i++) {
			result[i].elenco = emptylist();
			fscanf(fp, "%s", result[i].id);
			fscanf(fp, "%d/%d/%d", &(result[i].dd), &(result[i].mm),
&(result[i].yyyy));
			ch = fgetc(fp);
			while (ch != '\n') {
				fscanf(fp, "%s%f", temp.desc, &(temp.costo));
				result[i].elenco = cons(temp, result[i].elenco);
				ch = fgetc(fp);
			}
		}
		fclose(fp);
		return result;
	}
}

void stampaSpese(Spesa * v, int dim) {
	int i;
	list temp;
	for (i=0; i<dim; i++) {
		printf("%s %d/%d/%d\n", v[i].id, v[i].dd, v[i].mm, v[i].yyyy);
		temp = v[i].elenco;
		while (!empty(temp)) {
			printf("\t%s %f\n", head(temp).desc, head(temp).costo);
			temp = tail(temp);
		}
	}
}



// Es. 2
float totale(list elenco) {
	if (empty(elenco))
		return 0;
	else
		return head(elenco).costo + totale(tail(elenco));
}

float totalePerId(Spesa * v, int dim, char * id) {
	if (dim == 0)
		return 0;
	else
		if (strcmp(v[0].id, id)==0)
			return totale(v[0].elenco) + totalePerId(v+1, dim-1, id);
		else
			return totalePerId(v+1, dim-1, id);
}


int compareSpesa(Spesa s1, Spesa s2) {
	int result;
	float t1, t2;
	result = strcmp(s1.id, s2.id);
	if (result == 0) {
```

```
		t1 = totale(s1.elenco);
		t2 = totale(s2.elenco);
		if (t1>t2)
			result = 1;
		else if (t1<t2)
				result = -1;
			else
				result = 0;
	}
	return result;
}

void scambia(Spesa * f1, Spesa * f2) {
	Spesa temp;
	temp = *f1;
	*f1 = *f2;
	*f2 = temp;
}

void quickSortR(Spesa a[], int iniz, int fine) {
	int i, j, iPivot;
	Spesa pivot;
	if (iniz < fine) {
		i = iniz;
		j = fine;
		iPivot = fine;
		pivot = a[iPivot];
		do { /* trova la posizione del pivot */
			while (i < j && compareSpesa(a[i], pivot) <= 0) i++;
			while (j > i && compareSpesa(a[j], pivot) >= 0) j--;
			if (i < j) scambia(&a[i], &a[j]);
		} while (i < j);

		/* determinati i due sottoinsiemi */
		/* posiziona il pivot */
		if (i != iPivot && compareSpesa(a[i], a[iPivot]) != 0) {
			scambia(&a[i], &a[iPivot]);
			iPivot = i;
		}

		/* ricorsione sulle sottoparti, se necessario */
		if (iniz < iPivot - 1)
			quickSortR(a, iniz, iPivot - 1);
		if (iPivot + 1 < fine)
			quickSortR(a, iPivot + 1, fine);
	} /* (iniz < fine) */
} /* quickSortR */

void ordina(Spesa * v, int dim) {
	quickSortR(v,0,dim-1);
}


Spesa * riassunto(Spesa * v, int dim, int * dimR) {
	Spesa * result = NULL;
	int i;
	int j;
```

```
	list temp;

	*dimR = 0;

	if (dim >0) {
		*dimR = 1;
		ordina(v, dim);
		for (i=1; i<dim; i++) {
			if (strcmp(v[i].id, v[i-1].id))
				*dimR = *dimR+1;
		}

		result = (Spesa *) malloc(sizeof(Spesa) * *dimR);

		i = 0;
		j = 0;
		result[j] = v[i];
		for (i=1; i<dim; i++) {
			if (strcmp(v[i].id, v[i-1].id) == 0) {
				temp = v[i].elenco;
				while (!empty(temp)) {
					result[j].elenco = cons(head(temp), result[j].elenco);
					temp = tail(temp);
				}
			}
			else {
				j++;
				result[j] = v[i];
			}
		}
	}
	return result;
}
```

"main.c":

```
#include <stdio.h>

#include "spesa.h"

int main(int argc, char **argv)
{
	{ // Es. 1
		Spesa * v;
		int dim;
		int i;

		v = leggiSpese("spese.txt", &dim);
		stampaSpese(v, dim);
		for (i=0; i<dim; i++) {
			freelist(v[i].elenco);
		}
		free(v);
	}
```

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 3A di Giovedì 13 Febbraio 2015 – tempo a disposizione 2h

```
	{ // Es. 2
		Spesa * v;
		int dim;
		int i;

		v = leggiSpese("spese.txt", &dim);
		printf("%f", totalePerId(v, dim, "75CHSFRC"));
		ordina(v, dim);
		stampaSpese(v,dim);
		for (i=0; i<dim; i++) {
			freelist(v[i].elenco);
		}
		free(v);
	}

	{ // Es. 3 & 4
		Spesa * v;
		Spesa * r;
		int dimV;
		int dimR;
		int i;

		v = leggiSpese("spese.txt", &dimV);
		r = riassunto(v, dimV, &dimR);
		stampaSpese(r, dimR);
		for (i=0; i<dimR; i++)
			printf("Cliente %s: %f\n", r[i].id, totalePerId(r, dimR, r[i].id));
		/*
		 * Qui non dealloco le liste, per via del structure sharing che ho creato
nella funzione riassunto;
		for (i=0; i<dim; i++) {
			freelist(v[i].elenco);
		}
		 */
		free(v);
		free(r);
	}

	getchar();
	getchar();
	return 0;
}
```

Fondamenti di Informatica T-1, 2014/2015 – Modulo 2

Prova d'Esame 3A di Giovedì 13 Febbraio 2015 – tempo a disposizione 2h

"spese.txt":

```
7
75CHSFRC 13/02/2015 pane 4.56 salame 5.66 vino 19.99
77GNNCRL 11/02/2015 pane 2.37 latte 3.45 spinaci 5.78 formaggio 4.13
75CHSFRC 13/02/2015 burro 4.75 panna 3.55 biscotti 2.40 the 4.70 caffe 3.50 sale 0.75
75CHSFRC 14/02/2015 cioccolatini 40.00 10_rose_rosse_e_due_bianche 85.00
77GNNCRL 14/02/2015 ipod 99.00
88MLLPLA 10/02/2015 tartufo 147.00 olio 8.00
88MLLPLA 14/02/2015 macbook_pro 2647.00
```