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

Prova d'Esame 3A di Giovedì 14 Febbraio 2013 – 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"*.

Una ditta utilizza un sistema informatico per gestire i rimborsi dei viaggi di lavoro dei propri dipendenti. A tal scopo ha definito il concetto di **`Spesa`**, rappresentante un singolo esborso di denaro sostenuto dal dipendente, e il concetto di **`Rimborso`**, composto dall'insieme delle singole spese sostenute dal dipendente durante il viaggio di lavoro.
In particolare ogni **`Spesa`** è caratterizzata da un **identificatore** unico del viaggio di lavoro (un intero), dal **tipo** di spesa (una stringa di al più 1023 caratteri, senza spazi) e **dall'importo** (un float). Un **`Rimborso`** invece è caratterizzato dall'**identificatore** unico del viaggio di lavoro, da un array di **singole spese** relative al viaggio in questione e dalla **dimensione** di tale array. Ovviamente non è possibile sapere a priori quante spese sono state sostenute durante un certo viaggio, quindi l'array dovrà essere allocato dinamicamente della opportuna dimensione una volta noto il numero di spese.
Il sistema informatizzato funziona nel seguente modo: ogni dipendente accede al programma, specificando la propria matricola (una stringa di al più 7 caratteri numerici, senza spazi). Di seguito inserisce i dati relativi a tutte le spese da lui sostenute (spesso accade che vengano inserite spese relative a più viaggi differenti). Il programma salva tutte le spese inserite in un file di testo il cui nome è dato dalla matricola del dipendente, a cui viene aggiunto il suffisso "**`.txt`**". In un secondo momento poi il programma leggerà il contenuto di tale file, istanzierà le opportune strutture dati di tipo **`Rimborso`**, e per ognuna di queste ne calcolerà il valore di rimborso totale.

Esercizio 1 – Struttura dati Spesa e funzioni di scrittura/lettura (mod. element.h/.c e rimborsi.h/.c)

Si definisca un'opportuna struttura dati **`Spesa`**, al fine di rappresentare i dati relativi ad una singola spesa: in particolare si dovrà tenere traccia dell'identificatore del viaggio, del tipo di spesa e dell'importo.
Si definisca la procedura:

```
void scriviSpese(char * nomeFile);
```

che legga da standard input le spese sostenute da un lavoratore e le scriva su un file di testo. In particolare, la procedura deve innanzitutto chiedere che venga specificata la matricola del lavoratore, e poi deve aprire in scrittura un file col nome opportuno, come specificato in precedenza (matricola più ".txt"). Quindi la procedura deve chiedere all'utente di inserire i dati relativi ad ogni spesa, e salvarli sul file di testo, ogni **`Spesa`** su una singola riga, i vari campi separati da uno spazio. Non è noto a priori quante spese saranno introdotte. L'utente segnala l'intenzione di terminare la fase di inserimento specificando come identificatore unico del viaggio il valore "-1". Tramite il parametro **`nomeFile`**, la procedura deve salvare/restituire al chiamante il nome del file su cui sono state salvate le informazioni.
Si definisca poi la funzione:

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

che, ricevuto in ingresso il nome di un file di testo contenente le spese di un dipendente, e un intero **`dim`** passato per riferimento, allochi dinamicamente memoria a sufficienza (la minima necessaria) per contenere le strutture dati di tipo **`Spesa`** registrate nel file. La funzione deve restituire la dimensione di tale vettore tramite il parametro **`dim`**. La funzione dovrà leggere il contenuto del file e copiare i dati dentro l'apposito vettore allocato: dovrà infine restituire un puntatore all'area di memoria allocata.

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

Prova d'Esame 3A di Giovedì 14 Febbraio 2013 – tempo a disposizione 2h

Esercizio 2 – Struttura dati Rimborso, ordinamento e eliminazione duplicati (moduli element.h/.c e rimborsi.h/.c)

Si definisca un'opportuna struttura dati **`Rimborso`**, al fine di rappresentare i dati relativi ad una richiesta di rimborso per un viaggio di lavoro: in particolare si dovrà tenere traccia dell'identificatore del viaggio, dell'elenco delle spese sostenute durante tale viaggio, e della dimensione di tale vettore.
Si definisca una procedura:

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

che, ricevuto in ingresso un vettore **`v`** di strutture dati di tipo **`Spesa`** rappresentante le spese effettuate da un impiegato in un viaggio di lavoro, e la dimensione **`dim`** di tale vettore, ordini il vettore secondo il seguente criterio: gli elementi devono essere ordinati in modo crescente in base all'identificatore del viaggio; in caso di stesso identificatore, allora in ordine lessicografico rispetto al tipo di spesa; in caso di spesa uguale, in modo crescente in base all'importo. **Il candidato, per effettuare l'ordinamento, usi l'algoritmo MergeSort visto a lezione. Ovviamente si abbia cura di de-allocare eventuali aree di memoria usate temporaneamente per l'ordinamento.**
Il candidato definisca una funzione:

```
Spesa * eliminaDuplicati(Spesa * v, int dim, int * dimNew);
```

che ricevuti in ingresso un vettore di strutture dati di tipo **`Spesa`** e la dimensione **`dim`** di tale vettore, restituisca un nuovo vettore allocato dinamicamente (non necessariamente della dimensione minima possibile), da cui siano stati eliminati eventuali spese duplicate. Una spesa è un duplicato di un'altra se tutti i campi sono identici. Tramite il parametro dimNew la funzione deve restituire la dimensione logica del nuovo vettore restituito come risultato.
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 rimborsi (modulo rimborsi.h/rimborsi.c)

Il candidato definisca una funzione:

```
list creaRimborsi(Spesa * v, int dim);
```

che ricevuti in ingresso un vettore di strutture dati di tipo **`Spesa`** e la dimensione **`dim`** di tale vettore, restituisca una lista di strutture dati di tipo **`Rimborso`**. In particolare ogni **`Rimborso`** dovrà contenere tutte le spese relative ad uno stesso viaggio. A tal scopo si consiglia di ordinare il vettore, raggruppando così consecutivamente tutte le spese relative ad uno stesso viaggio.
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra.

Esercizio 4 – Calcolo dei rimborsi e del totale complessivo, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che chieda ad un singolo impiegato di fornire i propri dati e l'elenco delle spese sostenute (usando le funzioni di cui all'Es. 1); il programma poi usando le funzioni di cui agli Es. 2 e 3 determini una lista di strutture dati di tipo Rimborso, e infine calcoli e stampi a video l'importo totale relativo ad ogni singolo rimborso, nonché l'importo totale dato dalla somma di tutti i rimborsi.

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, 2012/2013 – Modulo 2

Prova d'Esame 3A di Giovedì 14 Febbraio 2013 – tempo a disposizione 2h

"element.h":

```
#ifndef ELEMENTH
#define ELEMENTH

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

#define DIMTIPO 1024
#define DIMFILE 12
#define DIMMATR 8

typedef struct {
	int viaggio;
	char tipo[DIMTIPO];
	float importo;
} Spesa;

typedef struct {
	int viaggio;
	Spesa * spese;
	int dim;
} Rimborso;

typedef Rimborso element;
int compare(Spesa s1, Spesa s2);

#endif
```

"element.c":

```
#include "element.h"

int compare(Spesa s1, Spesa s2) {
	int temp;
	temp = s1.viaggio - s2.viaggio;
	if (temp == 0)
		temp = strcmp(s1.tipo, s2.tipo);
	if (temp == 0)
		if (s1.importo-s2.importo<0)
			temp = -1;
			else if (s1.importo == s2.importo)
				temp = 0;
				else temp = 1;
	return temp;
}
```

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

Prova d'Esame 3A di Giovedì 14 Febbraio 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;
}
```

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

Prova d'Esame 3A di Giovedì 14 Febbraio 2013 – tempo a disposizione 2h

"rimborsi.h":

```
#ifndef RIMBORSIH
#define RIMBORSIH

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

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


void scriviSpese(char * nomeFile);
Spesa * leggiSpese(char * nomeFile, int * dim);

void ordina(Spesa * v, int dim);
Spesa * eliminaDuplicati(Spesa * v, int dim, int * dimNew);

list creaRimborsi(Spesa * v, int dim);

#endif
```

"rimborsi.c":

```
#include "element.h"
#include "rimborsi.h"


void scriviSpese(char * nomeFile) {
	FILE * fp;
	Spesa temp;

	printf("Inserire la matricola: ");
	scanf("%s", nomeFile);
	if (strlen(nomeFile)>0 && strlen(nomeFile)<=7)
		strcat(nomeFile, ".txt");
	else {
		printf("Errore nel trattare la matricola: %s.\n", nomeFile);
		getchar();
		exit(-1);
	}
	fp = fopen(nomeFile, "wt");
	if (fp == NULL) {
		printf("Errore nell'apertura del file: %s.\n", nomeFile);
		getchar();
		exit(-2);
	}
	printf("Inserire la spesa: ");
	scanf("%d%s%f", &(temp.viaggio), temp.tipo, &(temp.importo));
	while (temp.viaggio != -1) {
		fprintf(fp, "%d %s %f\n", temp.viaggio, temp.tipo, temp.importo);
		printf("Inserire la spesa: ");
		scanf("%d%s%f", &(temp.viaggio), temp.tipo, &(temp.importo));
	}
	fclose(fp);
```

```
	return;
}


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

	fp = fopen(nomeFile, "rt");
	if (fp == NULL) {
		printf("Errore nell'apertura in lettura del file %s.\n", nomeFile);
		getchar();
		exit(-3);
	}
	*dim = 0;
	while (fscanf(fp, "%d%s%f", &(temp.viaggio), temp.tipo, &(temp.importo)) == 3)
		*dim = *dim + 1;
	result = (Spesa*) malloc(sizeof(Spesa)* *dim);
	rewind(fp);
	i = 0;
	while (fscanf(fp, "%d%s%f", &(temp.viaggio), temp.tipo, &(temp.importo)) == 3) {
		result[i] = temp;
		i++;
	}
	fclose(fp);
	return result;
}


void merge(Spesa v[], int i1, int i2, int fine, Spesa vout[]){
	int i=i1, j=i2, k=i1;
	while ( i <= i2-1 && j <= fine ) {
		if (compare(v[i], v[j])<0)
			vout[k] = v[i++];
		else
			vout[k] = v[j++];
		k++;
	}
	while (i<=i2-1) { vout[k] = v[i++]; k++; }
	while (j<=fine) { vout[k] = v[j++]; k++; }
	for (i=i1; i<=fine; i++) v[i] = vout[i];
}

void mergeSort(Spesa v[], int first, int last, Spesa vout[]) {
	int mid;
	if ( first < last ) {
		mid = (last + first) / 2;
		mergeSort(v, first, mid, vout);
		mergeSort(v, mid+1, last, vout);
		merge(v, first, mid+1, last, vout);
	}
}


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

```
	Spesa * temp;
	temp = (Spesa*) malloc(sizeof(Spesa) * dim);
	mergeSort(v, 0, dim-1, temp);
	free(temp);
}

Spesa * eliminaDuplicati(Spesa * v, int dim, int * dimNew) {
	Spesa * result;
	int i, j;
	int trovato;

	result = (Spesa*) malloc(sizeof(Spesa) * dim);
	*dimNew = 0;
	for (i=0; i<dim; i++) {
		trovato = 0;
		for (j=0; j<*dimNew && !trovato; j++)
			if (compare(v[i], result[j])==0)
				trovato = 1;
		if (!trovato) {
			result[*dimNew] = v[i];
			*dimNew = *dimNew + 1;
		}
	}
	return result;
}


list creaRimborsi(Spesa * v, int dim) {
	Rimborso temp;
	list result;
	int start;
	int end;
	int i;

	result = emptylist();
	ordina(v, dim);
	start = 0;
	end = 0;
	while (end<dim) {
		while (end<dim && v[end].viaggio == v[start].viaggio)
			end++;
		if (start != end) {
			temp.viaggio = v[start].viaggio;
			temp.spese = (Spesa*) malloc(sizeof(Spesa) * (end-start));
			for (i=start; i<end; i++)
				temp.spese[i-start] = v[i];
			temp.dim = end-start;
		}
		start = end;
		result = cons(temp, result);
	}
	return result;
}
```

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

Prova d'Esame 3A di Giovedì 14 Febbraio 2013 – tempo a disposizione 2h

"main.c":

```
#include <stdio.h>

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

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

int main(int argc, char **argv)
{
	list elenco, tempElenco;
	char nomeFile[DIMFILE];
	Spesa * spese;
	int dimSpese;
	Spesa * spese2;
	int dimSpese2;
	int i;
	Spesa * v;
	float totaleRimborso;
	float totale;

	// es 1

	//scriviSpese(nomeFile);
	//spese = leggiSpese(nomeFile, &dimSpese);
	spese = leggiSpese("1234567.txt", &dimSpese);
	for (i=0; i<dimSpese; i++)
		printf("%d %s %f\n", spese[i].viaggio, spese[i].tipo, spese[i].importo);
	printf("FINE ES 1\n\n");


	// es 2
	ordina(spese, dimSpese);
	for (i=0; i<dimSpese; i++)
		printf("%d %s %f\n", spese[i].viaggio, spese[i].tipo, spese[i].importo);
	spese2 = eliminaDuplicati(spese, dimSpese, &dimSpese2);
	for (i=0; i<dimSpese2; i++)
		printf("%d %s %f\n", spese2[i].viaggio, spese2[i].tipo, spese2[i].importo);
	printf("FINE ES 2\n\n");

	// es 3
	elenco = creaRimborsi(spese2, dimSpese2);
	tempElenco = elenco;
	while (!empty(tempElenco)) {
		v = head(tempElenco).spese;
		for (i=0; i<head(tempElenco).dim; i++)
			printf("%d %s %f\n", v[i].viaggio, v[i].tipo, v[i].importo);
		printf("\n\n");
		tempElenco = tail (tempElenco);
	}

	// es 4
	totale = 0;
```

```
        tempElenco = elenco;
        while (!empty(tempElenco)) {
               totaleRimborso = 0;
               v = head(tempElenco).spese;
               for (i=0; i<head(tempElenco).dim; i++)
                      totaleRimborso = totaleRimborso + v[i].importo;
               printf("Viaggio %d, rimborso: %6.2f\n", v[0].viaggio, totaleRimborso);
               totale = totale + totaleRimborso;
               tempElenco = tail (tempElenco);
        }
        printf("Totale: %6.2f", totale);

        // DE-ALLOCAZIONE
        free(spese);
        free(spese2);
        tempElenco = elenco;
        while (!empty(tempElenco)) {
               free(head(tempElenco).spese);
               elenco = tempElenco;
               tempElenco = tail(tempElenco);
               free(elenco);
        }

        getchar();
        return 0;
}
```