Fondamenti di Informatica T-1, 2009/2010 – Modulo 2
Prova d'Esame 3a di Giovedì 11 Febbraio 2010 – tempo a disposizione 2h30'

Prima di cominciare: si scarichi dal sito **`http://esamix.labx`** il file **`StartKit3A.zip`** contenente i file necessari (*solution* di VS2008 e progetto compresi).

Avvertenze per la consegna: apporre all'inizio di ogni file sorgente un commento contenente i propri dati (**cognome, nome, numero di matricola**) e **l'identificativo (A/B)** 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*, effettuare di tanto in tanto *"Rebuild All"*.

AutostradePerBologna s.p.a. tiene traccia di tutte le macchine che attraversano i caselli autostradali memorizzando targa del veicolo (sequenza di 7 caratteri alfanumerici) e importo del pedaggio (numero reale) in un file di testo **`pedaggi.txt`**. La prima riga di tale file contiene il numero di pedaggi memorizzati nel file stesso, ciascuna riga successiva contiene targa e importo relativi ad un pedaggio. Ciascuna targa può comparire più volte, in quanto uno stesso veicolo può aver attraversato più volte un casello autostradale.

Ai fini della fatturazione AutostradePerBologna s.p.a. accorpa i pedaggi per gruppi di autovetture, ad esempio allo scopo di mandare un'unica fattura ad un'azienda di autotrasporti che possiede più furgoni. I gruppi di autovetture sono specificati in file contenenti targhe tutte diverse tra loro (si veda il file **`gruppo1.txt`**).

Attenzione: ciascuna funzione realizzata deve essere opportunamente testata invocandola correttamente dal **`main(...)`** e mostrando sullo standard output il risultato dell'elaborazione effettuata. Inoltre tutta la memoria allocata dinamicamente (quindi anche gli elementi delle liste) deve essere opportunamente deallocata alla fine della computazione di ciascun esercizio.

Esercizio 1 – Strutture dati e funzioni di lettura/stampa (moduli element.h e pedaggi.h/pedaggi.c)

a) Si definisca un'opportuna struttura dati **`pedaggio`** al fine di rappresentare un singolo pedaggio (targa dell'autoveicolo e costo del pedaggio).

b) Si realizzi la funzione

```
pedaggio* leggiPedaggi(FILE *fp, int* dim);
```

che, dato un file contenente i pedaggi, legga i dati contenuti e li memorizzi in un vettore di strutture dati di tipo **`pedaggio`** di dimensione opportuna. L'array restituito deve essere di dimensione minima sufficiente a contenere tutti i pedaggi registrati; a tal scopo la prima riga del file passato in ingresso contiene il numero di pedaggi memorizzati nel file stesso.

c) Si realizzi poi la procedura

```
void stampaPedaggi(pedaggio *array, int dim);
```

che, ricevuto in ingresso un vettore di strutture dati di tipo **`pedaggio`** e la sua dimensione **`dim`**, stampi a video tutti i dati contenuti.

Fondamenti di Informatica T-1, 2009/2010 – Modulo 2
Prova d'Esame 3a di Giovedì 11 Febbraio 2010 – tempo a disposizione 2h30'

Esercizio 2 - Fatturazione in base a gruppi (modulo pedaggi.h/pedaggi.c)

Si definisca la funzione

```
pedaggio* gruppoPedaggi(char* fileNameGruppo, pedaggio* tuttiPedaggi,
                                                int dimTutti, int* dimGruppo)
```

che, dati il nome di un file contenente un gruppo di targhe (ad esempio **`gruppo1.txt`**) ed un array di strutture pedaggio **`tuttiPedaggi`** con relativa dimensione **`dimTutti`**, restituisca un nuovo array di strutture pedaggio (e relativa dimensione) contenente un elemento **`pedaggio`** per ciascuna targa presente nel file **`fileNameGruppo`**; ad esempio se il file **`fileNameGruppo`** contiene sei targhe, l'array restituito deve contenere sei elementi.
Il costo del pedaggio di ciascun elemento deve rappresentare la somma totale dei pedaggi relativi a quell'elemento; se un'autovettura non è presente nell'array **`tuttiPedaggi`** allora tale autovettura avrà un costo totale uguale a 0.

Esercizio 3 - Ordinamento in base al pedaggio (pedaggi.h/pedaggi.c/element.h/element.c)

Si definisca la funzione

```
list ordinaGruppo(pedaggio* gruppo, int dimGruppo);
```

che, dato un array di strutture **`pedaggio`** relativo ad un gruppo di autovetture ottenuto utilizzando la funzione **`gruppoPedaggi(...)`**, restituisca una lista (ADT **`list`**) contenente tutti gli elementi di **`gruppo`** ordinati secondo il costo totale, mettendo in testa l'elemento del gruppo con costo totale inferiore ed in coda l'elemento del gruppo con costo totale superiore.

A tal fine si consiglia l'uso della funzione **`insord(...)`** già definita per l'ADT list che si appoggia su un'opportuna funzione

```
int compare(element el1, element el2);
```

da realizzare da parte del candidato.

Esercizio 4 - Calcolo del massimo pedaggio (pedaggi.h/pedaggi.c)

Si definisca la funzione

```
pedaggio* maxPedaggio(pedaggio* tuttiPedaggi, int dimTutti, int* dimMax)
```

che, dati un array **`tuttiPedaggi`** contenente tutti i pedaggi presenti nel file **`pedaggi.txt`** ed il numero di elementi presenti in tale array, restituisca un nuovo array (e relativa dimensione logica) contenente un solo elemento per ciascuna targa presente nell'array **`tuttiPedaggi`**; il costo assegnato ad ogni elemento deve rappresentare l'importo massimo registrato per quella certa targa. Non è necessario che l'array restituito abbia dimensione fisica minima.

Dove specificato realizzare le funzioni nei moduli indicati; laddove non specificato si lascia libertà di scelta al candidato.
NOTA: Si abbia cura di deallocare la memoria allocata dinamicamente dal programma realizzato.

Fondamenti di Informatica T-1, 2009/2010 – Modulo 2
Prova d'Esame 3a di Giovedì 11 Febbraio 2010 – tempo a disposizione 2h30'

"pedaggi.txt"

```
9
aa111aa 6.7
bb222bb 3.1
aa111aa 7.7
cc333cc 4
dd444dd 12.2
aa111aa 6.7
bb222bb 6.2
dd444dd 0.9
aa111aa 6.7
```

"gruppo1.txt"

```
aa111aa
cc333cc
ee555ee
```

"element.h"

```
#ifndef _ELEMENT_H
#define _ELEMENT_H

typedef struct {
      char targa[8];
      float costo;
} pedaggio;

typedef pedaggio element;

int compare(element el1, element el2);

#endif _ELEMENT_H
```

"element.c"

```
#include "element.h"
#include <string.h>

int compare(element el1, element el2){
      if(el1.costo < el2.costo) return +1;
      else if (el1.costo > el2.costo) return -1;
      else return 0;
}
```

"pedaggi.h"

```
#ifndef _PEDAGGI_H
#define _PEDAGGI_H

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

```
pedaggio* leggiPedaggi(FILE* fp, int* dim);
void stampaPedaggi(pedaggio* array, int dim);
pedaggio* gruppoPedaggi(char* fileName, pedaggio* tuttiPedaggi, int dimTutti,
int* dimGruppo);
list ordinaGruppo(pedaggio* gruppo, int dimGruppo);
pedaggio* maxPedaggio(pedaggio* tuttiPedaggi, int dimTutti, int* dimMax);

#endif _PEDAGGI_H
```

"pedaggi.c"

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "element.h"
#include "list.h"

pedaggio* leggiPedaggi(FILE* fp, int* dim){
      pedaggio *res;
      int count;
      fscanf(fp, "%d", &count);
      res = (pedaggio*)malloc(sizeof(pedaggio)*count);
      *dim=0;
      while( fscanf(fp, "%s %f", res[*dim].targa, &(res[*dim].costo)) == 2 ){
            (*dim)++;
      }
      return res;
}

void stampaPedaggi(pedaggio* array, int dim){
      int i;
      for(i=0; i<dim; i++){
            printf("%s %f\n", array[i].targa, array[i].costo);
      }
}

pedaggio* gruppoPedaggi(char* fileName, pedaggio* tuttiPedaggi, int dimTutti,
int* dimGruppo){
      FILE *fp;
      pedaggio *res;
      char targa[8];
      int i;
      if( (fp=fopen(fileName, "rt")) == NULL ){
            printf("errore fopen %s\n", fileName);
            exit(-1);
      }
      *dimGruppo=0;
      while( fscanf(fp,"%s",targa) == 1 ){
            (*dimGruppo)++;
      }
      res=(pedaggio*)malloc(sizeof(pedaggio)*(*dimGruppo));
      rewind(fp);
      *dimGruppo=0;
      while( fscanf(fp,"%s", res[*dimGruppo].targa) == 1 ){
            res[*dimGruppo].costo=0;
            for(i=0; i<dimTutti; i++){
                  if(strcmp(tuttiPedaggi[i].targa,res[*dimGruppo].targa)==0){
                        res[*dimGruppo].costo+=tuttiPedaggi[i].costo;
```

```
                }
            }
            (*dimGruppo)++;
      }
      return res;
}

list ordinaGruppo(pedaggio* gruppo, int dimGruppo){
      int i;
      list res;
      res = emptylist();
      for(i=0; i<dimGruppo; i++){
            res = insord(gruppo[i], res);
      }
      return res;
}

pedaggio* maxPedaggio(pedaggio* tuttiPedaggi, int dimTutti, int* dimMax){
      int i, j, trovato;
      pedaggio *res, temp;
      *dimMax = 0;
      res = (pedaggio*)malloc(sizeof(pedaggio)*dimTutti);
      for(i=0; i<dimTutti; i++){
            trovato=0;
            for(j=0; trovato==0 && j<*dimMax; j++){
                  if( strcmp(tuttiPedaggi[i].targa,res[j].targa) == 0 ){
                        temp = res[*dimMax];
                        trovato = 1;
                  }
            }
            if(trovato){
                  if(res[j].costo < tuttiPedaggi[i].costo){
                        res[j].costo = tuttiPedaggi[i].costo;
                  }
            }
            else{
                  res[j] = tuttiPedaggi[i];
                  (*dimMax)++;
            }
      }
      return res;
}
```

"main.c"

```
#include <stdio.h>
#include <stdlib.h>
#include "element.h"
#include "pedaggi.h"
#include "list.h"

int main(void)
{
      FILE *fp;
      int dimTutti, dimGruppo, dimMax;
      pedaggio *tutti, *gruppo, *max;
      list l1, temp;
```

```
	printf("Esercizio 1\n");
	if( (fp=fopen("pedaggi.txt","rt")) == NULL ){
		printf("errore fopen pedaggi.txt\n");
		exit(-2);
	}
	tutti = leggiPedaggi(fp, &dimTutti);
	stampaPedaggi(tutti, dimTutti);
	free(tutti);

	printf("\nEsercizio 2\n");
	if( (fp=fopen("pedaggi.txt","rt")) == NULL ){
		printf("errore fopen pedaggi.txt\n");
		exit(-2);
	}
	tutti = leggiPedaggi(fp, &dimTutti);
	gruppo = gruppoPedaggi("gruppo1.txt", tutti, dimTutti, &dimGruppo);
	stampaPedaggi(gruppo, dimGruppo);
	free(tutti);
	free(gruppo);


	printf("\nEsercizio 3\n");
	if( (fp=fopen("pedaggi.txt","rt")) == NULL ){
		printf("errore fopen pedaggi.txt\n");
		exit(-2);
	}
	tutti = leggiPedaggi(fp, &dimTutti);
	gruppo = gruppoPedaggi("gruppo1.txt", tutti, dimTutti, &dimGruppo);
	l1 = ordinaGruppo(gruppo, dimGruppo);
	while(!empty(l1)){
		printf("%s %f\n", head(l1).targa, head(l1).costo);
		temp = l1;
		l1 = tail(l1);
		free(temp);
	}
	free(tutti);
	free(gruppo);

	printf("\nEsercizio 4\n");
	if( (fp=fopen("pedaggi.txt","rt")) == NULL ){
		printf("errore fopen pedaggi.txt\n");
		exit(-2);
	}
	tutti = leggiPedaggi(fp, &dimTutti);
	max = maxPedaggio(tutti, dimTutti, &dimMax);
	stampaPedaggi(max, dimMax);
	free(tutti);
	free(max);

	return (0);
}
```