Fondamenti di Informatica T-1, 2018/2019 – Modulo 2

Prova d'Esame 1A di Giovedì 10 Gennaio 2019 – 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** necessari alla compilazione e alla corretta esecuzione del programma.
Nota: **il main non è opzionale**; i *test* richiesti vanno implementati.
Consiglio: per verificare l'assenza di *warning*, eseguire sempre *"Rebuild All"*.

Una grande capitale europea ha rivoluzionato il pagamento della metropolitana, sostituendo i classici biglietti cartacei con una tessera con tecnologia di pagamento a contatto, chiamata OstricaCard. In pratica gli utenti "caricano" un certo importo economico sulla loro OstricaCard, e poi ogni volta che prendono la metro il loro credito viene diminuito del prezzo del percorso compiuto.
Tale meccanismo avviene tramite la registrazione su di un file di testo "**`eventi.txt`**" degli eventi importanti, un evento per riga. Ogni evento (ogni riga) contiene le seguenti informazioni: l'**id** unico della OstricaCard (un intero); a seguire, separata da uno spazio, il nome della stazione di **ingresso** (una stringa di al più 255 caratteri utili, contenene spazi); separata dal carattere '@', il nome della stazione di **uscita** (una stringa di al più 255 caratteri utili, contenente spazi).
In un secondo file di nome "**`tariffe.txt`**" si tiene memoria delle tariffe: per ogni possibile coppia di stazione di ingresso/stazione di uscita si memorizza il costo esatto. Quindi su ogni riga si trova: il nome di una stazione di **ingresso** (una stringa di al più 255 caratteri utili, contenene spazi); separata dal carattere '@', il nome di una stazione di **uscita** (una stringa di al più 255 caratteri utili, contenene spazi); infine, separata da un carattere '@', il **costo** in euro della tratta specificata (un float).

Esercizio 1 – Strutture dati Evento, Tariffa, e funzioni di lett./scritt. (mod. element.h e metro.h/.c)

Si definisca una opportuna struttura dati **`Evento`** per memorizzare un singolo evento, cioè l'id della OstricaCard, stazione di ingresso e stazione di uscita. Si definisca poi una opportuna struttura dati **`Tariffa`**, per la memorizzazione della tariffa di una singola tratta; stazione di ingresso, stazione di uscita, e costo.

Si definisca la funzione:

```
Evento leggiUno(FILE * fp);
```

che, ricevuto in ingresso un puntatore ad una struttura dati di tipo **`FILE`**, legga i dati contenuti su una sola riga e li restituisca tramite una struttura dati di tipo **`Evento`**. Qualora vi siano problemi nella lettura, la funzione deve restituire una struttura dati con id della OstricaCard pari a -1.

Si definisca la funzione:

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

che, ricevuto in ingresso il nome di un file, legga da tale file i dati di tutti gli eventi e li restituisca tramite una lista di strutture dati di tipo **`Evento`**. Qualora la funzione incontri dei problemi nella lettura, o vi siano errori nell'apertura del file, la funzione dovrà stampare un messaggio di errore a video, e restituire una lista vuota.

Si definisca la funzione:

```
Tariffa * leggiTariffe(char * fileName, int * dim);
```

che, ricevuto in ingresso il nome di un file, legga da tale file i dati di tutte le tariffe e li restituisca tramite un array (allocato dinamicamente e della dimensione minima) di strutture dati di tipo **`Tariffa`**. Tramite il parametro dim, la funzione dovrà restituire la dimensione dell'array allocato dinamicamente. Qualora la funzione incontri dei problemi nella lettura, o vi siano errori nell'apertura del file, la funzione dovrà stampare un messaggio di errore a video, restituire un puntatore a NULL e dim pari a zero.

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, 2018/2019 – Modulo 2

Prova d'Esame 1A di Giovedì 10 Gennaio 2019 – tempo a disposizione 2h

Esercizio 2 –Ordinamento e conta delle singole card (moduli element.h/.c e metro.h/.c)

Il candidato definisca una procedura:

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

che, ricevuti in ingresso un vettore di strutture dati di tipo **`Tariffa`** e la dimensione di tale vettore, ordini il vettore secondo il seguente criterio: in ordine lessicografico crescente in base alla stazione di ingresso; a parità di quest'ultima, in ordine lessicografico crescente in base alla stazione di uscita; a parità, in ordine crescente in base al costo. A tal scopo, il candidato utilizzi l'algoritmo di ordinamento "quick sort" visto a lezione.

Si definisca poi una funzione:

```
float ricerca(Tariffa * v, int dim, char * ingresso, char * uscita);
```

che, ricevuto in ingresso un vettore di strutture dati di tipo **`Tariffa`**, e la sua dimensione **`dim`**, e le stazioni di ingresso e di uscita, restituisca il costo del percorso specificato. Qualora non sia possibile trovare il costo, la funzione restituisca il valore zero.

Esercizio 3 – Calcolo totali (modulo metro.h/metro.c)

Si sviluppi una procedura:

```
void totali(Tariffa * tariffe, int dim, list eventi);
```

che, ricevuti in ingresso un array di strutture dati di tipo Tariffa e la dimensione di tale array, e la lista degli eventi registrati relativamente alle OstricaCard, stampi a video (e una volta sola per ogni OstricaCard) quanto è dovuto per i viaggi (gli eventi) effettuati da tali OstricaCard.

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

Il candidato realizzi nella funzione **`main(…)`** un programma che legga dal file tutti gli eventi relativi alle OstricaCard, e stampi a video, per ogni card, quanto è l'importo dovuto.

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, 2018/2019 – Modulo 2

Prova d'Esame 1A di Giovedì 10 Gennaio 2019 – tempo a disposizione 2h

"element.h":

```
#define _USE_CRT_NO_WARNINGS
#ifndef _ELEMENT_H
#define _ELEMENT_H

#include <string.h>

#define DIM 256

typedef struct {
	int id;
	char ingresso[DIM];
	char uscita[DIM];
} Evento;

typedef Evento element;


typedef struct {
	char ingresso[DIM];
	char uscita[DIM];
	float costo;
} Tariffa;

int compare(Tariffa t1, Tariffa t2);
#endif
```

"element.c":

```
#include "element.h"

int compare(Tariffa t1, Tariffa t2) {
	int result;
	result = strcmp(t1.ingresso, t2.ingresso);
	if (result == 0)
		result = strcmp(t1.uscita, t2.uscita);
	if (result == 0) {
		if (t1.costo < t2.costo)
			result = -1;
		else
			if (t1.costo > t2.costo)
				result = 1;
	}

	return result;
}
```

Fondamenti di Informatica T-1, 2018/2019 – Modulo 2

Prova d'Esame 1A di Giovedì 10 Gennaio 2019 – 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);

#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 showlist(list l) {
       element temp;
       if (!empty(l)) {
              temp = head(l);
              printf("%d: %s ---TO--- %s\n",
                     temp.id, temp.ingresso, temp.uscita);
              showlist(tail(l));
              return;
       }
       else {
              printf("\n\n");
              return;
       }
}



int member(element el, list l) {
       int result = 0;
       while (!empty(l) && !result) {
              result = (el.id == head(l).id);
              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, 2018/2019 – Modulo 2

Prova d'Esame 1A di Giovedì 10 Gennaio 2019 – tempo a disposizione 2h

"metro.h":

```
#define _CRT_SECURE_NO_WARNINGS
#ifndef _METRO_H
#define _METRO_H

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

// Es. 1
Evento leggiUno(FILE * fp);
list leggiTutti(char * fileName);
Tariffa * leggiTariffe(char * fileName, int * dim);
// Es. 2
void ordina(Tariffa * v, int dim);
float ricerca(Tariffa * v, int dim, char * ingresso, char * uscita);
// Es. 3
void totali(Tariffa * tariffe, int dim, list eventi);
#endif
```

"metro.c":

```
#include "metro.h"
// Es. 1
Evento leggiUno(FILE * fp) {
	Evento result;
	char ch;
	int i;
	if (fscanf(fp, "%d", &(result.id)) == 1) {
		// leggo lo spazio di separazione
		fgetc(fp);
		// leggo la stringa ingresso
		i = 0;
		ch = fgetc(fp);
		// Nota: il controllo sulla dimensione massima della stringa è discutibile
		// poiché assume come dimensione massima DIM-1, che è una costante...
		// Purtroppo il prototipo della funzione non prevede tale informazione,
		// e quindi l'unico modo è usare la costante simbolica DIM
		while (ch != '@' && ch != '\n' && ch != EOF && i<DIM-1) {
			result.ingresso[i++] = ch;
			ch = fgetc(fp);
		}
		result.ingresso[i] = '\0';
		// leggo la stringa uscita
		i = 0;
		ch = fgetc(fp);
		while (ch != '\n' && ch != EOF && i<DIM-1) {
			result.uscita[i++] = ch;
			ch = fgetc(fp);
		}
		result.uscita[i] = '\0';
	}
	else
		result.id = -1;
```

```
	return result;
}

list leggiTutti(char * fileName) {
	FILE * fp;
	list result;
	Evento temp;

	result = emptylist();
	fp = fopen(fileName, "rt");
	if (fp != NULL) {
		temp = leggiUno(fp);
		while (temp.id != -1) {
			if (temp.id != -1)
				result = cons(temp, result);
			temp = leggiUno(fp);
		}
		fclose(fp);
	}
	else {
		printf("Errore nell'apertura del file: %s\n", fileName);
	}
	return result;
}

int leggiUnaTariffa(FILE * fp, Tariffa * dest) {
	int i;
	char ch;
	int success;

	// leggo la stringa ingresso
	i = 0;
	ch = fgetc(fp);
	while (ch != '@' && ch != '\n' && ch != EOF && i<DIM-1) {
		dest->ingresso[i++] = ch;
		ch = fgetc(fp);
	}
	dest->ingresso[i] = '\0';
	// leggo la stringa uscita
	i = 0;
	ch = fgetc(fp);
	while (ch != '@' && ch != '\n' && ch != EOF && i<DIM-1) {
		dest->uscita[i++] = ch;
		ch = fgetc(fp);
	}
	dest->uscita[i] = '\0';
	success = (fscanf(fp, "%f", &(dest->costo)) == 1);
	fgetc(fp);
	return success;
}

Tariffa * leggiTariffe(char * fileName, int * dim) {
	FILE * fp;
	Tariffa * result = NULL;
	Tariffa temp;
	int count;
```

```
	*dim = 0;
	fp = fopen(fileName, "rt");
	if (fp != NULL) {
		count = 0;
		// prima conto quanti elementi ci sono...
		while (leggiUnaTariffa(fp, &temp))
			count++;

		// alloco memoria e riavvolgo il file
		result = (Tariffa*)malloc(sizeof(Tariffa) * count);
		rewind(fp);

		// leggo e copio in memoria
		while (leggiUnaTariffa(fp, &temp)) {
			result[*dim] = temp;
			*dim = *dim + 1;
		}

		fclose(fp);
	}
	else {
		printf("Errore nell'apertura del file: %s\n", fileName);
	}
	return result;
}

// Es. 2
void scambia(Tariffa *a, Tariffa *b) {
	Tariffa tmp = *a;
	*a = *b;
	*b = tmp;
}
void quickSortR(Tariffa a[], int iniz, int fine) {
	int i, j, iPivot;
	Tariffa pivot;
	if (iniz < fine) {
		i = iniz;
		j = fine;
		iPivot = fine;
		pivot = a[iPivot];
		do {  /* trova la posizione del pivot */
			while (i < j && compare(a[i], pivot)<=0) i++;
			while (j > i && compare(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 && compare(a[i], a[iPivot])) {
			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(Tariffa * v, int dim) {
	quickSortR(v, 0, dim - 1);
}
// una variante della ricerca binaria...
int compareBin(char * ingresso, char * uscita, Tariffa t) {
	int result;
	result = strcmp(ingresso, t.ingresso);
	if (result == 0) {
		result = strcmp(uscita, t.uscita);
	}
	return result;
}
float ricerca(Tariffa * v, int dim, char * ingresso, char * uscita) {
	int first = 0, last = dim - 1, med = (first + last) / 2;
	int found = 0;
	float result = 0.0F;
	ordina(v, dim);
	while ((first <= last) && (found == 0)) {
		if (compareBin(ingresso, uscita, v[med]) == 0) {
			found = 1;
			result = v[med].costo;
		}
		else
			if (compareBin(ingresso, uscita, v[med])<0)
				last = med - 1;
			else
				first = med + 1;
		med = (first + last) / 2;
	}
	return result;
}

// Es. 3
void totali(Tariffa * tariffe, int dim, list eventi) {
	list temp;
	list temp2;
	float total;
	Evento current;
	temp = eventi;
	while (!empty(temp)) {
		current = head(temp);
		if (!member(current, tail(temp))) {
			// calcoliamo il totale per questa OstricaCard
			temp2 = eventi;
			total = 0;
			while (!empty(temp2)) {
				if (head(temp2).id == current.id)
					total = total + ricerca(tariffe, dim,
head(temp2).ingresso, head(temp2).uscita);
				temp2 = tail(temp2);
			}
			printf("Totale della Card %d: %6.2f\n", current.id, total);
		}
		temp = tail(temp);
	}
}
```

Fondamenti di Informatica T-1, 2018/2019 – Modulo 2

Prova d'Esame 1A di Giovedì 10 Gennaio 2019 – tempo a disposizione 2h

"main.c":

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

int main() {
	{	// Es. 1
		list eventi;
		Tariffa * tariffe;
		int dim;
		int i;
		eventi = leggiTutti("eventi.txt");
		showlist(eventi);
		freelist(eventi);
		tariffe = leggiTariffe("tariffe.txt", &dim);
		for (i = 0; i < dim; i++) {
			printf("%s ---TO--- %s  EURO: %6.2f\n", tariffe[i].ingresso,
tariffe[i].uscita, tariffe[i].costo);
		}
		free(tariffe);
	}
	{	// Es. 2
		Tariffa * tariffe;
		int dim;
		int i;
		printf("\n\n");
		tariffe = leggiTariffe("tariffe.txt", &dim);
		ordina(tariffe, dim);
		for (i = 0; i < dim; i++) {
			printf("%s ---TO--- %s  EURO: %6.2f\n", tariffe[i].ingresso,
tariffe[i].uscita, tariffe[i].costo);
		}
		printf("Costo da Gloucester a Boston Manor: %6.2f\n", ricerca(tariffe, dim,
"Gloucester Road", "Boston Manor"));
		printf("Costo da Heathrow a  Gloucester Road: %6.2f\n", ricerca(tariffe,
dim, "Heathrow Terminal 5", "Gloucester Road"));
		free(tariffe);
	}
	{	// Es. 3 && 4
		Tariffa * tariffe;
		int dim;
		list eventi;
		printf("\n\n");
		eventi = leggiTutti("eventi.txt");
		tariffe = leggiTariffe("tariffe.txt", &dim);
		totali(tariffe, dim, eventi);
		freelist(eventi);
		free(tariffe);
	}
	return 0;
}
```

Fondamenti di Informatica T-1, 2018/2019 – Modulo 2

Prova d'Esame 1A di Giovedì 10 Gennaio 2019 – tempo a disposizione 2h

"eventi.txt":

```
59 Heathrow Terminal 5@Gloucester Road
134 Heathrow Terminal 5@Gloucester Road
244 Heathrow Terminal 5@Gloucester Road
59 Gloucester Road@Boston Manor
59 Boston Manor@Heathrow Terminal 5
244 Gloucester Road@North Ealing
```

"tariffe.txt":

```
Heathrow Terminal 5@Gloucester Road@5.10
Gloucester Road@Boston Manor@4.00
Boston Manor@Heathrow Terminal@1.10
Gloucester Road@North Ealing@4.00
```