# Fondamenti di Informatica T-1, 2013/2014 – Modulo 2

## Prova d'Esame 5A di Giovedì 10 Luglio 2014 – 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 framework gestionale per aziende include un programma per la registrazione delle fatture. Il programma è composto di due parti: una prima parte in cui l'impiegato inserisce i dati di una fattura (tramite tastiera) e il programma li memorizza su file; e una seconda parte in cui il programma effettua alcuni controlli sulle fatture inserite, e stampa alcune informazioni.
Una fattura è identificata da un **identificatore** unico della fattura (un intero), da un **importo** (un float), e da una **descrizione** della fattura stessa (una stringa di al più 2047 caratteri utili, contenente spazi).

### *Esercizio 1 – Strutture dati Fattura, e funzioni di lett./scritt. (mod. element.h/c e fatture.h/.c)*

Si definisca un'opportuna struttura dati **`Fattura`**, al fine di rappresentare i dati relativi ad una fattura, come da descrizione precedente.

Si definisca la funzione:

```
Fattura leggiUnaFattura(FILE * fp);
```

che, ricevuto in ingresso un puntatore a FILE (uno stream già aperto), legga i dati relativi ad una sola fattura e li restituisca in una apposita struttura dati. Si assuma che l'utente inserisca i dati in ordine, e termini ogni singolo campo con un carattere invio (quindi per una fattura l'utente inserirà tre volte il carattere invio: uno '**\n**' dopo l'id, uno '**\n**' dopo l'importo, e uno '**\n**' dopo la descrizione).

Il candidato implementi poi una funzione:

```
list leggiDaTastiera();
```

che legga un insieme di fatture dal terminale, e restituisca tali informazioni in una lista di strutture dati di tipo Fattura. In particolare, la funzione deve come prima cosa chiedere all'utente se vuole inserire una fattura. Se la risposta è positiva, allora si deve leggere una fattura usando obbligatoriamente la funzione precedente (si passi come parametro fp la costante predefinita "stdin", che indica appunto un file collegato direttamente alla tastiera dell'utente). La fattura letta dovrà essere memorizzata in una lista. Effettuata la lettura, la funzione dovrà ripetere il tutto richiedendo all'utente se vuole inserire una fattura, etc. . Quando l'utente segnalerà l'intenzione di non inserire altre fatture, la funzione dovrà terminare e restituire la lista con i dati letti.

Si definisca poi la procedura:

```
void salvaFatture(char * fileName, list elenco);
```

che, ricevuta in ingresso il nome di un file, e una lista di strutture dati di tipo **`Fattura`**, scriva tale elenco sul file col nome specificato in ingresso. In particolare sulla prima riga del file si scriva il numero di fatture che verranno salvate (in pratica la lunghezza della lista), e a seguire tutte le fatture memorizzate nella lista, avendo cura di salvare ogni campo di una singola fattura su una riga a sé stante. Quindi ogni fattura occuperà tre righe.
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra.

# Fondamenti di Informatica T-1, 2013/2014 – Modulo 2

## Prova d'Esame 5A di Giovedì 10 Luglio 2014 – tempo a disposizione 2h

### *Esercizio 2 – Lettura delle fatture salvate, ordinamento e verifiche (modulo element.h/.c e fatture.h/.c)*

Si definisca una funzione:

```
Fattura* leggiDaFile(char * fileName, int * dim);
```

che, ricevuta in ingresso il nome di un file contenente le fatture salvate tramite le funzioni di cui all'Esercizio 1, restituisca un vettore allocato dinamicamente, della dimensione minima necessaria, contenente le fatture salvate nel file indicato. A tal scopo, il candidato usi obbligatoriamente la funzione **`leggiUnaFattura`**(…) definita in precedenza. Tramite il parametro dim inoltre la funzione restituisca la dimensione del vettore allocato dinamicamente.

Si definisca una procedura:

```
void ordina(Fattura * elenco, int dim);
```

che, ricevuta in ingresso un vettore di strutture dati di tipo **`Fattura`**, e la dimensione di tale vettore, lo ordini in base al seguente criterio: in ordine lessicografico in base alla descrizione della fattura, e a seguire in ordine crescente in base all'id della fattura, e poi in base all'importo. Per effettuare l'ordinamento, si usi l'algoritmo quick sort visto a lezione.

Si realizzi poi una funzione:

```
Fattura * ripetuti(Fattura * elenco, int dim, int * newDim);
```

che restituisca un nuovo vettore di strutture dati di tipo **`Fattura`**, di dimensione eventualmente non minima, da cui siano state eliminate eventuali fatture ripetute presenti nel vettore di fatture fornito in ingresso. Può capitare infatti che un impiegato in fase di inserimento inserisca più volte la stessa fattura: due fatture sono da considerarsi uguali se hanno tutti e tre i campi uguali. In caso di fatture ripetute, il vettore restituito come risultato dovrà contenerne solo una. Tramite il parametro **`newDim`** la funzione deve restituire la dimensione logica del 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 –Calcolo dei totali (modulo fatture.h/.c)*

Si definisca quindi una funzione:

```
float totali(Fattura * elenco, int dim, char * sub);
```

che, ricevuti in ingresso un vettore di strutture dati di tipo **`Fattura`** e la sua dimensione, e una stringa ben formata **`sub`**, calcoli la somma di tutti gli importi delle fatture che contengono nella loro descrizione la stringa sub. A tal scopo si suggerisce al candidato l'uso della funzione di libreria (<string.h>):

```
char *strstr(const char *str, const char *strSearch);
```

che restituisce un puntatore alla prima occorrenza di **`strSearch`** nella stringa **`str`**, oppure NULL se **`strSearch`** non è presente in **`str`**.
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento della funzione di cui sopra.

### *Esercizio 4 – Lettura e salvataggio fatture, controllo ripetizioni e calcolo totali (main.c)*

Il candidato realizzi nella funzione **`main(…)`** un programma che chieda all'utente di inserire le fatture e le salvi in un file (nome a scelta del candidato); in seguito, il programma legga le fatture dal file, elimini le eventuali ripetizioni, e calcoli (e stampi a video) la somma degli importi delle fatture relative ad una stringa indicata dall'utente, che deve comparire come sottostringa nel campo descrizione delle fatture.
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, 2013/2014 – Modulo 2

## Prova d'Esame 5A di Giovedì 10 Luglio 2014 – tempo a disposizione 2h

"element.h":

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

#ifndef _ELEMENT_H
#define _ELEMENT_H

#define DIM_DESC 2048


typedef struct {
	int id;
	float importo;
	char desc[DIM_DESC];
} Fattura;


typedef Fattura element;

int compareFattura(Fattura f1, Fattura f2);

#endif
```

"element.c":

```
#include "element.h"


int compareFattura(Fattura f1, Fattura f2) {
	int result = strcmp(f1.desc, f2.desc);
	if (result == 0)
		result = f1.id-f2.id;
	if (result == 0)
		if (f1.importo-f2.importo<0)
			result = -1;
		else if (f1.importo-f2.importo>0)
			result = 1;
		else
			result = 0;
	return result;
}
```

# Fondamenti di Informatica T-1, 2013/2014 – Modulo 2

## Prova d'Esame 5A di Giovedì 10 Luglio 2014 – 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;
}
/*
list insord_p(element el, list l) {
       list pprec, patt = l, paux;
       int trovato = 0;

       while (patt!=NULL && !trovato) {
              if (compareTrasloco(el, patt->value)<0)
                     trovato = 1;
              else {
                     pprec = patt;
                     patt = patt->next;
              }
       }
       paux = (list) malloc(sizeof(item));
       paux->value = el;
       paux->next = patt;
       if (patt==l)
              return paux;
       else {
              pprec->next = paux;
              return l;
       }
}
*/
```

# Fondamenti di Informatica T-1, 2013/2014 – Modulo 2

## Prova d'Esame 5A di Giovedì 10 Luglio 2014 – tempo a disposizione 2h

"fatture.h":

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

#ifndef _FATTURE_H
#define _FATTURE_H

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

Fattura leggiUnaFattura(FILE * fp);
list leggiDaTastiera();
void salvaFatture(char * fileName, list elenco);

Fattura* leggiDaFile(char * fileName, int * dim);
void ordina(Fattura * elenco, int dim);
Fattura * ripetuti(Fattura * elenco, int dim, int * newDim);

float totali(Fattura * elenco, int dim, char * sub);

#endif
```

"fatture.c":

```
#include "fatture.h"

Fattura leggiUnaFattura(FILE * fp) {
	Fattura result;
	int i;
	char ch;

	if (fscanf(fp, "%d%f", &(result.id), &(result.importo)) == 2) {
		i=0;
		fgetc(fp); // elimino il new line precedente...
		do {
			ch = fgetc(fp);
			if (ch!='\n' && i<DIM_DESC-1) {
				result.desc[i] = ch;
				i++;
			}
		} while (ch!='\n');
		result.desc[i] = '\0';
	}
	return result;
}

list leggiDaTastiera() {
	char answer;
	Fattura temp;

	list result = emptylist();
	do {
		printf("Vuoi inserire una fattura ? (y/n): ");
		scanf("%c%*c", &answer);
		if (answer == 'y') {
```

```
                temp = leggiUnaFattura(stdin);
                result = cons(temp, result);
            }
        } while (answer=='y');
        return result;
}

int length(list l) {
        int result = 0;
        while (!empty(l)) {
            l = tail(l);
            result++;
        }
        return result;
}

void salvaFatture(char * fileName, list elenco) {
        FILE * fp;
        int dim;
        Fattura temp;

        fp = fopen(fileName, "wt");
        if (fp == NULL) {
            printf("Errore nell'apertura del file: %s\n", fileName);
            exit(1);
        }
        else {
            dim = length(elenco);
            fprintf(fp, "%d\n", dim);
            while (!empty(elenco)) {
                temp = head(elenco);
                fprintf(fp, "%d\n%f\n%s\n", temp.id, temp.importo, temp.desc);
                elenco = tail(elenco);
            }
            fclose(fp);
        }
        return;
}

Fattura* leggiDaFile(char * fileName, int * dim) {
        Fattura * result = NULL;
        FILE * fp;
        int i;

        fp = fopen(fileName, "rt");
        if (fp == NULL) {
            printf("Errore nell'apertura del file: %s\n", fileName);
            exit(1);
        }
        else {
            fscanf(fp, "%d", dim);
            result = (Fattura*) malloc(sizeof(Fattura) * *dim);
            for (i=0; i<*dim; i++) {
                result[i] = leggiUnaFattura(fp);
            }
            fclose(fp);
        }
```

```
	return result;
}

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

void quickSortR(Fattura a[], int iniz, int fine) {
	int i, j, iPivot;
	Fattura pivot;
	if (iniz < fine) {
		i = iniz;
		j = fine;
		iPivot = fine;
		pivot = a[iPivot];
		do { /* trova la posizione del pivot */
			while (i < j && compareFattura(a[i], pivot) <= 0) i++;
			while (j > i && compareFattura(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 && compareFattura(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(Fattura * v, int dim) {
	quickSortR(v, 0, dim-1);
}

int presente(Fattura * vet, int dim, Fattura f1) {
	int trovato = 0;
	int i;

	for (i=0; i<dim && !trovato; i++) {
		if (compareFattura(f1, vet[i])==0)
			trovato = 1;
	}
	return trovato;
}


Fattura * ripetuti(Fattura * elenco, int dim, int * newDim) {
```

```
	Fattura * result;
	int i;

	*newDim = 0;
	result = (Fattura*) malloc(sizeof(Fattura) * dim);
	for (i=0; i<dim; i++) {
		if (!presente(elenco, i, elenco[i])) {
			result[*newDim] = elenco[i];
			*newDim = *newDim + 1;
		}
	}
	return result;
}


float totali(Fattura * elenco, int dim, char * sub) {
	float sum = 0.0f;
	int i;

	for (i=0; i<dim; i++)
		if (strstr(elenco[i].desc, sub) != NULL)
			sum = sum + elenco[i].importo;
	return sum;
}
```

# Fondamenti di Informatica T-1, 2013/2014 – Modulo 2

## Prova d'Esame 5A di Giovedì 10 Luglio 2014 – tempo a disposizione 2h

"main.c":

```
#include "element.h"
#include "fatture.h"



int main() {
	/*
	{	// Test Es. 1
		Fattura temp;
		list elenco;
		elenco = leggiDaTastiera();
		salvaFatture("fatture.txt", elenco);
		freelist(elenco);
	}
	*/
	{	// Test Es. 2
		Fattura * vett;
		Fattura * vett2;
		int dim;
		int dim2;
		int i;
		vett = leggiDaFile("fatture.txt", &dim);
		for (i=0; i<dim; i++)
			printf("%d %f %s\n", vett[i].id, vett[i].importo, vett[i].desc);
		ordina(vett, dim);
		for (i=0; i<dim; i++)
			printf("%d %f %s\n", vett[i].id, vett[i].importo, vett[i].desc);
		vett2 = ripetuti(vett, dim, &dim2);
		for (i=0; i<dim2; i++)
			printf("%d %f %s\n", vett2[i].id, vett2[i].importo, vett2[i].desc);
		free(vett);
		free(vett2);
	}
	{	// Test Es. 3 & 4
		Fattura * vett;
		Fattura * vett2;
		int dim;
		int dim2;
		float sum;
		vett = leggiDaFile("fatture.txt", &dim);
		vett2 = ripetuti(vett, dim, &dim2);
		sum = totali(vett2, dim2, "df");
		printf("Totale per la sottostringa df: %f\n", sum);
		free(vett);
		free(vett2);
	}

	return 0;
}
```