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

Prova d'Esame 4A di Giovedì 13 Giugno 2013 – tempo a disposizione 2h

Prima di cominciare: si scarichi dal sito **`http://esamix.labx`** il file **`StartKit4A.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 banca offre tra i propri servizi la possibilità di effettuare pagamenti vari (bollette, tasse, etc.). A tal scopo ha introdotto il concetto di **`Pagamento`**, un insieme di informazioni che contraddistinguono un determinato pagamento. In particolare, ogni **`Pagamento`** racchiude le seguenti informazioni: **id unico del cliente** (un intero), **id unico del pagamento** (un intero), **importo** del pagamento effettuato (un float), e la **causale** del pagamento (una stringa di al più 255 caratteri, contenente spazi).
I server della banca, al termine di ogni giornata, salvano tutti i pagamenti effettuati in un file di testo, ogni singolo pagamento su una riga, i dati nell'ordine in cui sono descritti sopra, e separati da uno spazio. Si veda a titolo di esempio il file "**`20130613.txt`**" fornito nello StartKit. Purtroppo a causa di qualche bug nel software, alcuni pagamenti vengono registrati più volte nel file, con tutti i campi uguali tranne l'importo che varia casualmente.

Esercizio 1 – Struttura dati Pagamento e funzioni di lettura e confronto (mod. element.h/.c e pagamenti.h/.c)

Si definisca un'opportuna struttura dati **`Pagamento`**, al fine di rappresentare i dati relativi ad un singolo pagamento: in particolare si dovrà tenere traccia del id cliente, del id pagamento, dell'importo e della causale.

Si definisca la funzione:

```
list leggiPagamenti(char * nomeFile);
```

che ricevuto come parametro il nome di un file contenente dei pagamenti, legga i dati ivi contenuti e restituisca come risultato una lista di strutture dati **`Pagamento`** istanziata con i dati letti. In caso di errore nell'apertura del file, la funzione deve stampare a video un messaggio di errore, e deve restituire una lista vuota. La funzione deve inoltre controllare in fase di lettura la dimensione della causale: capita infatti che nel file vengano inserite causali molto più lunghe di quanto previsto per il campo causale. In tal caso, la funzione deve leggere soltanto i primi 255 caratteri, e tralasciare i rimanenti. Si noti inoltre che tutte le linee del file sono sempre terminate da un carattere speciale '**`\n`**', e che l'ultima riga del file è vuota.

Si definisca poi la funzione:

```
int equals(Pagamento p1, Pagamento p2);
```

che, ricevuto in ingresso due pagamenti **`p1`** e **`p2`**, restituisca in uscita un intero. L'intero restituito dovrà essere interpretabile come "vero" qualora **`p1`** e **`p2`** siano uguali, o come "falso" altrimenti. Due pagamenti **`p1`** e **`p2`** sono uguali se hanno tutti i campi uguali eccetto l'importo (che quindi può essere diverso in **`p1`** e **`p2`**).

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, 2012/2013 – Modulo 2

Prova d'Esame 4A di Giovedì 13 Giugno 2013 – tempo a disposizione 2h

Esercizio 2 – Eliminazione ripetuti e spese di un singolo cliente (moduli element.h/.c e pagamenti.h/.c)

Si definisca una funzione:

```
list cancellaDoppi(list pag);
```

che ricevuta in ingresso una lista di strutture dati di tipo **`Pagamento`**, restituisca in uscita una nuova lista di strutture dati di tipo **`Pagamento`**, da cui siano stati eliminati eventuali pagamenti ripetuti. Due pagamenti sono ripetuti se sono uguali, secondo il criterio definito per la funzione **`equals(…)`** di cui all'esercizio precedente. Si noti che in caso di pagamenti ripetuti solo uno dei pagamenti ripetuti (a scelta del candidato) deve essere inserito nella lista restituita come risultato, mentre gli altri pagamenti vanno ignorati.

Si definisca una funzione:

```
Pagamento * elencoPag(list pag, int id_cliente, int * dim);
```

che ricevuta in ingresso una lista di strutture dati di tipo **`Pagamento`**, e un intero rappresentante **`l'id`** di un cliente, restituisca in uscita un vettore di strutture dati di tipo **`Pagamento`** contenente tutti i pagamenti effettuati da quel cliente. Tramite il parametro **`dim`** passato per riferimento, la funzione deve restituire la dimensione del vettore. Tale vettore deve essere allocato dinamicamente (della dimensione strettamente necessaria), e inoltre deve essere ordinato in base al valore decrescente dell'importo; per valori di importo uguali, si ordini in base all'ordine lessicografico rispetto alla causale del pagamento. **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 abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra.

Esercizio 3 –Determinazione dei totali spesi dai clienti (modulo pagamenti.h/pagamenti.c)

Il candidato definisca una procedura:

```
void totali(list pag);
```

che riceva in ingresso una lista di strutture dati di tipo **`Pagamento`**. La procedura deve stampare a video, per ogni cliente, i pagamenti a lui riferibili, correttamente ordinati come specificato nell'esercizio precedente. La procedura deve stampare poi anche la somma totale dei pagamenti presenti in **`pag`** e riferibili a tale cliente.
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 totali complessivi, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che provveda a leggere dal file fornito nello StartKit l'elenco dei pagamenti, e dopo aver eliminato le ripetizioni, stampi a video le spese effettuate da ogni singolo cliente (in ordine, come specificato nell'esercizio 2), nonché i totali per ogni cliente.

Il candidato abbia cura di de-allocare tutta la memoria allocata dinamicamente, ivi compresa la memoria allocata per le liste. Si noti che potrebbe essere necessario deallocare della memoria anche all'interno delle singole funzioni definite negli esercizi precedenti.

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

Prova d'Esame 4A di Giovedì 13 Giugno 2013 – tempo a disposizione 2h

"element.h":

```
#ifndef _ELEMENT_H
#define _ELEMENT_H

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

#define DIM_CAUSALE 256

typedef struct {
	int id_cliente;
	int id_pag;
	float importo;
	char causale[DIM_CAUSALE];
} Pagamento;

typedef Pagamento element;

int equals(Pagamento p1, Pagamento p2);
int compare(Pagamento p1, Pagamento p2);

#endif
```

"element.c":

```
#include "element.h"


int equals(Pagamento p1, Pagamento p2) {
	return
		p1.id_cliente==p2.id_cliente &&
		p1.id_pag==p2.id_pag &&
		strcmp(p1.causale, p2.causale)==0;
}

int compare(Pagamento p1, Pagamento p2) {
	if (p1.importo>p2.importo)
		return -1;
	else
		if (p1.importo<p2.importo)
			return 1;
		else
			return strcmp(p1.causale, p2.causale);
}
```

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

Prova d'Esame 4A di Giovedì 13 Giugno 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;
}

void showlist(list l) {
       element temp;
       if (!empty(l)) {
              temp = head(l);
              printf("%d %d %.2f %s\n", temp.id_cliente, temp.id_pag, temp.importo,
temp.causale);
              return showlist(tail(l));
       }
       else {
              printf("\n\n");
              return;
       }
}
```

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

Prova d'Esame 4A di Giovedì 13 Giugno 2013 – tempo a disposizione 2h

"pagamenti.h":

```
#ifndef _PAGAMENTI
#define _PAGAMENTI

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

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


list leggiPagamenti(char * nomeFile);

list cancellaDoppi(list pag);
Pagamento * elencoPag(list pag, int id_cliente, int * dim);

void totali(list pag);

#endif
```

"pagamenti.c":

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

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

list leggiPagamenti(char * nomeFile) {
	FILE * fp;
	Pagamento temp;
	char ch;
	int i;
	list result = emptylist();

	fp = fopen(nomeFile, "rt");
	if (fp == NULL)
		printf("Errore nell'apertura di %s\n", nomeFile);
	else {
		while (fscanf(fp, "%d%d%f", &temp.id_cliente, &temp.id_pag, &temp.importo)
== 3) {
			i=0;
			do {
				ch =fgetc(fp);
				if (ch!='\n' && i<DIM_CAUSALE-1) {
					temp.causale[i] = ch;
					i++;
				}
			} while (ch!='\n' && i<DIM_CAUSALE-1);
			// terminatore di stringa
```

```
                temp.causale[i] = '\0';
                // consumo eventuali caratteri rimanenti
                while (ch!='\n')
                        ch = fgetc(fp);
                result = cons(temp, result);
        }
        fclose(fp);
    }
    return result;
}

int member(element el, list l) {
    if (empty(l))       return 0;
    else
        if (equals(el, head(l))) return 1;
        else return  member(el, tail(l));
}

list cancellaDoppi(list pag) {
    list result = emptylist();
    Pagamento temp;
    while (!empty(pag)) {
        temp = head(pag);
        if ( !member(temp, result) )
            result = cons(temp, result);
        pag=tail(pag);
    }
    return result;
}

void merge(Pagamento v[], int i1, int i2, int fine, Pagamento 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(Pagamento v[], int first, int last, Pagamento 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);
    }
}

Pagamento * elencoPag(list pag, int id_cliente, int * dim) {
    int i;
    list temp;
```

```
	Pagamento * result = NULL;
	Pagamento * temp2;
	*dim = 0;
	temp = pag;
	while (!empty(temp)) {
		if (head(temp).id_cliente == id_cliente)  (*dim)++;
		temp = tail(temp);
	}
	if (*dim>0) {
		result = (Pagamento*) malloc(sizeof(Pagamento)* *dim);
		temp = pag;
		i=0;
		while (!empty(temp)) {
			if (head(temp).id_cliente == id_cliente) {
				result[i] = head(temp);
				i++;
			}
			temp = tail(temp);
		}
		temp2 = (Pagamento*) malloc(sizeof(Pagamento) * *dim);
		mergeSort(result, 0, *dim - 1, temp2);
		free(temp2);
	}
	return result;
}

int memberCliente(int id_cliente, list pag) {
	if (empty(pag)) return 0;
	else
		if (head(pag).id_cliente == id_cliente) return 1;
		else return memberCliente(id_cliente, tail(pag));
}

void totali(list pag) {
	int id_cliente;
	int i;
	int dim;
	float sum = 0;
	Pagamento * spese;
	list temp;
	temp = pag;
	while (!empty(temp)) {
		id_cliente = head(temp).id_cliente;
		if (!memberCliente(id_cliente, tail(temp))) {
			spese = elencoPag(pag, id_cliente, &dim);
			sum = 0;
			for (i=0; i<dim; i++) {
				printf("%d %d %.2f %s\n", spese[i].id_cliente, spese[i].id_pag,
spese[i].importo, spese[i].causale);
				sum = sum + spese[i].importo;
			}
			printf("-------------------\nTotale: %.2f\n\n", sum);
			free(spese);
		}
		temp = tail(temp);
	}
}
```

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

Prova d'Esame 4A di Giovedì 13 Giugno 2013 – tempo a disposizione 2h

"main.c":

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

int main() {
	{ // Es. 1
		list pag;
		pag = leggiPagamenti("20130613.txt");
		showlist(pag);
		freelist(pag);
	}

	{ // Es. 2
		list pag;
		list noDoppi;
		Pagamento * spese;
		int i;
		int dim;

		pag = leggiPagamenti("20130613.txt");
		noDoppi = cancellaDoppi(pag);
		showlist(noDoppi);

		spese = elencoPag(noDoppi, 23, &dim);
		for (i=0; i<dim; i++) {
			printf("%d %d %.2f %s\n", spese[i].id_cliente, spese[i].id_pag,
spese[i].importo, spese[i].causale);
		}
		printf("\n\n");
		freelist(pag);
		freelist(noDoppi);
		free(spese);
	}

	{  // Es. 3 e 4
		list pag;
		list noDoppi;
		int i;
		int dim;
		pag = leggiPagamenti("20130613.txt");
		noDoppi = cancellaDoppi(pag);

		totali(noDoppi);

		freelist(pag);
		freelist(noDoppi);
	}

	system("pause");
	return 0;
}
```

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

Prova d'Esame 4A di Giovedì 13 Giugno 2013 – tempo a disposizione 2h

"20130613.txt":

```
23 12567 483.00 Saldo IMU Comune di Bologna
23 12567 0.12 Saldo IMU Comune di Bologna
45 12689 61.00 Multa Divieto di sosta
23 13500 350.00 Saldo rata scuola materna
45 13689 243.00 Bonifico riparazione macchina fotografica
```