Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 4A di Martedì 12 Giugno 2012 – 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"*.

Un commercialista memorizza in alcuni file i dati relativi ad alcuni suoi clienti. In particolare, in un primo file di testo di nome "**`elenco.txt`**", il commercialista ha scritto per ogni riga del file il **cognome** di un cliente (una stringa di al più 255 caratteri, contenente spazi e terminata da un carattere ';'), il **nome** del cliente (una stringa di al più 255 caratteri, contenente spazi e terminata da un carattere ';'), il **nome di un file** (una stringa di al più 12 caratteri, non contenente spazi, e terminata da uno spazio), e il reddito complessivo annuale (un intero).
Per ogni cliente registrato nel file "**`elenco.txt`**", il commercialista salva in un secondo file di testo (il cui nome è registrato nel file di elenco) le spese sostenute dal cliente, secondo la seguente convenzione: in ogni riga viene registrato **l'importo** della spesa (un intero, separato dal campo successivo da uno spazio), una stringa rappresentante una descrizione della spesa (al più 255 caratteri, senza spazi, ma terminata da uno spazio), e a seguire la lettera 'D' se tale spesa è detraibile, o la lettera 'N' se la spesa non è detraibile.
A titolo di esempio, si vedano i file forniti nello StartKit.

Esercizio 1 – Struttura dati Cliente e funzioni di lettura/stampa (mod. element.h/.c e cliente.h/cliente.c)

Si definisca un'opportuna struttura dati **`Cliente`**, al fine di rappresentare i dati relativi a ogni cliente. Si definisca la funzione:

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

che, ricevuto in ingresso il nome di un file di testo rappresentante l'elenco dei clienti, restituisca una lista di strutture dati di tipo **`Cliente`**, contenente tutte le informazioni presenti nel file il cui nome è passato come parametro. Poiché i cognomi e i nomi dei clienti possono contenere spazi, per leggere tali campi si suggerisce l'uso della funzione **`readField(…)`** fornita nello StartKit tramite il modulo "read.h/read.c". La funzione **`readField(char buffer[], char sep, FILE *f)`** legge dal file **`f`** tutti i caratteri (spazi compresi) fino al separatore **`sep`**, e li copia nell'array **`buffer`**; la funzione restituisce il numero di caratteri letti.
Si definisca la procedura:

```
void stampaLista(list v);
```

che, ricevuta in ingresso una lista di strutture dati di tipo **`Cliente`**, stampi a video l'elenco dei clienti con tutte le informazioni relative.
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, 2011/2012 – Modulo 2

Prova d'Esame 4A di Martedì 12 Giugno 2012 – tempo a disposizione 2h

Esercizio 2 – Struttura dati Spesa, lettura spese e ordinamento (moduli element.h/.c e cliente.h/.c)

Dopo aver definito una opportuna struttura dati di nome **`Spesa`**, atta a contenere le informazioni relative ad una singola voce di spesa, il candidato definisca una funzione:

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

che, ricevuto in ingresso il nome di un file contenente le voci di spesa effettuate da un cliente, e un intero passato per riferimento, restituisca un vettore allocato dinamicamente (della dimensione minima necessaria) contenente le spese memorizzate nel file indicato come parametro. Tramite l'intero passato per riferimento, la funzione deve restituire la dimensione del vettore allocato dinamicamente.
Il candidato definisca poi una procedura:

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

tale che, ricevuti un vettore di strutture dati di tipo **`Spesa`** e la dimensione di tale vettore, ordini tale vettore ponendo prima le spese "detraibili" (indicate con la lettera 'D') e a seguire le spese non detraibili (indicate con la lettera 'N'). Nell'ambito dello stesso tipo di spese, queste devono essere ordinate in base all'importo, in ordine decrescente (dagli importi maggiori agli importi minori).
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra. A tal fine, utilizzi come test uno dei file forniti nello StartKit.

Esercizio 3 – Eliminazione dei clienti con dati incerti (modulo cliente.h/cliente.c)

Il commercialista ha compiuto alcuni errori nel memorizzare le informazioni. In particolare, in fase di inserimento può aver attribuito lo stesso file delle spese a due clienti diversi. Ovviamente non è possibile sapere a quale dei due clienti il file si riferisca effettivamente, e quindi è necessario non considerare tali clienti. A tal scopo, si definisca una funzione:

```
list eliminaRipetuti(list c);
```

che, ricevuto in ingresso una lista di strutture dati di tipo **`Cliente`**, restituisca una nuova lista dove sono stati eliminati tutti i clienti che avevano come riferimento lo stesso file per le spese. Ad esempio, facendo riferimento al file "**`elenco.txt`**" fornito nello StartKit, il terzo e il quarto cliente registrati in tale file fanno riferimento allo stesso file per le spese, e quindi devono essere filtrati via entrambi.

Esercizio 4 – Stampa delle spese totali sostenute dai clienti, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che stampi a video i dati e la spesa totale sostenuta dai clienti noti. Si abbia cura di ignorare quei clienti che hanno un problema sul file contenente le loro spese, come spiegato al punto precedente.
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, 2011/2012 – Modulo 2

Prova d'Esame 4A di Martedì 12 Giugno 2012 – tempo a disposizione 2h

"element.h":

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

#ifndef _ELEMENT_H
#define	_ELEMENT_H

#define DIM_NOME 256
#define DIM_NOMEFILE 13

typedef struct {
	char cognome[DIM_NOME];
	char nome[DIM_NOME];
	char fileName[DIM_NOMEFILE];
	int reddito;
} Cliente;

typedef Cliente element;

typedef struct {
	int importo;
	char causale[DIM_NOME];
	char detraibile;
} Spesa;

int compareSpesa(Spesa s1, Spesa s2);
int equals(Cliente c1, Cliente c2);

#endif /* _ELEMENT_H */
```

"element.c":

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

int compareSpesa(Spesa s1, Spesa s2) {
	if (s1.detraibile != s2.detraibile)
		return s1.detraibile - s2.detraibile;
	else
		return s2.importo - s1.importo;
}


int equals(Cliente c1, Cliente c2) {
	return strcmp(c1.fileName,c2.fileName) == 0;
}
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 4A di Martedì 12 Giugno 2012 – 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);

#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);
}
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 4A di Martedì 12 Giugno 2012 – tempo a disposizione 2h

```
void showlist(list l) {
       element temp;
       if (!empty(l)) {
              temp = head(l);
              printf("%s %s %d\n", temp.cognome, temp.nome, temp.id);
              return showlist(tail(l));
       }
       else {
              printf("\n\n");
              return;
       }
}

void freelist(list l) {
       if (empty(l))
              return;
       else {
              freelist(tail(l));
              free(l);
       }
       return;
}
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 4A di Martedì 12 Giugno 2012 – tempo a disposizione 2h

"read.h":

```
#ifndef READ_H_
#define READ_H_
#include <stdio.h>

int readField(char buffer[], char sep, FILE *f);

#endif /* READ_H_ */
```

"read.c":

```
#include "read.h"

int readField(char buffer[], char sep, FILE *f)
{
	int i = 0;
	char ch = fgetc(f);

	// mangia tutti i terminatori di linea che trova all'inizio...
	if (ch==10)
		do {
			ch = fgetc(f);
		} while (ch==10);
	// legge tutti i caratteri fino a quando non incontra sep, un fine linea o il file
	// termina
	while (ch != sep && ch!=10 && ch != EOF) {
		buffer[i] = ch;
		i++;
		ch = fgetc(f);
	}
	buffer[i] = '\0';
	return i;
}
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 4A di Martedì 12 Giugno 2012 – tempo a disposizione 2h

"cliente.h":

```
#ifndef CLIENTE_H_
#define CLIENTE_H_

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

list leggiClienti(char* fileName);
void stampaLista(list v);
Spesa * leggiSpese(char * fileName, int * dim);
void ordina(Spesa * v, int n);
list eliminaRipetuti(list c);

#endif /* CLIENTE_H_ */
```

"cliente.c":

```
#include "cliente.h"
#include "read.h"

list leggiClienti(char* fileName) {
	FILE * fp;
	list result;
	int ok;
	Cliente temp;

	result = emptylist();
	fp = fopen(fileName,"rt");
	if (fp == NULL) {
		printf("Problema durante l'apertura del file %s\n.", fileName);
		//getch();
		exit(-1);
	}
	else {
		ok =1;
		while (ok) {
			ok = readField(temp.cognome, ';', fp);
			if (ok) ok=readField(temp.nome, ';', fp);
			if (ok) ok = fscanf(fp, "%s", temp.fileName);
			if (ok==1) ok = fscanf(fp, "%d", &(temp.reddito));
			if (ok==1) result = cons(temp, result);
		}
		fclose(fp);
	}
	return result;
}

void stampaLista(list v) {
	Cliente c;
	while(!empty(v)) {
		c = head(v);
		printf("%s %s %s Euro: %d\n", c.cognome, c.nome, c.fileName, c.reddito);
```

```
			v = tail(v);
		}
}

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

		*dim = 0;
		fp = fopen(fileName, "rt");
		if (fp == NULL) {
			printf("Errore nell'aprire il file %s\n", fileName);
			exit(-1);
		}
		else {
			while (fscanf(fp, "%d %s %c", &(temp.importo), temp.causale,
&(temp.detraibile)) == 3)
					*dim = *dim + 1;
			rewind(fp);
			result = (Spesa *) malloc(sizeof(Spesa) * *dim);
			i=0;
			while (fscanf(fp, "%d %s %c", &(temp.importo), temp.causale,
&(temp.detraibile)) == 3) {
				result[i] = temp;
				i++;
			}
			fclose(fp);
		}
		return result;
}

void scambia(Spesa *a, Spesa *b) {
		Spesa tmp = *a;
		*a = *b;
		*b = tmp;
}
// bubble sort
void ordina(Spesa * v, int n) {
		int i, ordinato = 0;
		while (n>1 && !ordinato) {
			ordinato = 1;
			for (i=0; i<n-1; i++)
				if (compareSpesa(v[i],v[i+1])>0) {
					scambia(&v[i],&v[i+1]);
					ordinato = 0;
				}
			n--;
		}
}

int conta(list lc, Cliente c1) {
		int result = 0;
		while (!empty(lc)) {
			if (equals(c1, head(lc)))
				result++;
```

```
            lc = tail(lc);
      }
      return result;
}

list eliminaRipetuti(list c) {
      list result = emptylist();
      list temp = c;

      while (!empty(temp)) {
            if (conta(c, head(temp)) == 1)
                  result = cons(head(temp), result);
            temp = tail(temp);
      }
      return result;
}
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 4A di Martedì 12 Giugno 2012 – tempo a disposizione 2h

"main.c":

```
#include "cliente.h"

int main() {
	list v, v2;
	list temp;
	Cliente c;
	Spesa * sp;
	int dim;
	int i;
	int sum;

	{
		v = leggiClienti("elenco.txt");
		stampaLista(v);
		freelist(v);
	}
	{
		sp = leggiSpese("11.txt", &dim);
		ordina(sp, dim);
		for (i=0; i<dim; i++)
			printf("%c %d %s\n", sp[i].detraibile, sp[i].importo, sp[i].causale);
		free(sp);
	}
	{
		v = leggiClienti("elenco.txt");
		v2 = eliminaRipetuti(v);
		stampaLista(v2);
		freelist(v);
		freelist(v2);
	}
	{
		sum = 0;
		v = leggiClienti("elenco.txt");
		v2 = eliminaRipetuti(v);
		temp = v2;
		while (!empty(temp)) {
			c = head(temp);
			sp = leggiSpese(c.fileName, &dim);
			for (i=0; i<dim; i++)
				sum = sum + sp[i].importo;
			printf("%s %s ha speso euro %d\n", c.cognome, c.nome, sum);
			temp = tail(temp);
			free(sp);
		}
		freelist(v);
		freelist(v2);
	}

	return 0;
}
```

Fondamenti di Informatica T-1, 2011/2012 – Modulo 2

Prova d'Esame 4A di Martedì 12 Giugno 2012 – tempo a disposizione 2h

"elenco.txt":

```
Chesani;Federico Maria;11.txt 20000
Giannelli Serbelloni;Carlo;21.txt 19000
Mello;Paola;33.txt 30000
Chesani; Paolo;33.txt 23000
```

"11.txt":

```
1500 macchinaFotograficaNuova N
130 esameMedico D
140 esameMedico D
280 visitaDietologo D
460 tagliandoAuto N
50 offertaONG D
50 acquistoOlio N
780 bollettaRiscaldamento N
```

"21.txt":

```
370 affettatrice N
340 iscrizionePalestra D
90 torneoCalcetto N
580 stagionaleSkiPass N
120 esameMedico D
```

"33.txt":

```
120 prosciutto N
440 formaDiGrana D
120 esameMedico D
```