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

Prova d'Esame 3A di Giovedì 14 Febbraio 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"*.

Un negozio di cartoleria e articoli per ufficio rifornisce diverse aziende, vendendo a credito gli articoli di cancelleria. In un file di testo "**`credito.txt`**" la cartoleria segna cosa ha venduto a quale azienda. Nel file di testo, per ogni acquisto effettuato a credito, scrive le seguenti informazioni: innanzitutto il nome dell'azienda (una stringa di al più 15 caratteri utili, senza spazi); a seguire, separato da uno spazio, un elenco di coppie nome prodotto (una stringa di al più 15 caratteri utili, senza spazi) e il costo del prodotto acquistato (un float), ogni elemento separato da spazi, e ogni coppia separata da spazi. L'elenco e il numero di articoli presi in ogni acquisto cambiano da acquisto ad acquisto; inoltre, come se non bastasse, l'impiegato della cancelleria scrive gli acquisti a volte un acquisto per riga, a volte molti acquisti sulla stessa riga, altre volte ancora un acquisto è spezzato su più righe. Due cose sono però certe: ogni elenco di cose acquistate termina sempre con la stringa "**`fine`**", e ogni elenco di cose acquistate comprende al massimo 32 coppie articolo-prezzi. Si veda, a titolo di esempio, il file fornito nello StartKit.

Esercizio 1 – Strutture dati Articolo, Acquisto, e funzioni di lett./scritt. (mod. element.h e crediti.h/.c)

Si definisca una opportuna struttura dati **`Articolo`** per memorizzare una singola coppia data dal nome di un articolo acquistato, e dal suo prezzo; si definisca poi una opportuna struttura dati **`Acquisto`**, per la memorizzazione di un acquisto effettuato da una azienda: quindi una stringa per il nome dell'azienda, un array di dimensione fissata staticamente pari a 32, ed un intero per indicare il numero di coppie memorizzate nell'array (cioè la sua dimensione logica).

Si definisca la funzione:

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

che, ricevuto in ingresso un puntatore ad una struttura dati di tipo FILE (rappresentante quindi un file già aperto), legga da tale file una struttura dati di tipo Acquisto e la restituisca come risultato. Qualora non sia possibile leggere più strutture dati (perché ad esempio si è giunti al termine del file), la funzione restituisca una struttura dati col nome azienda pari alla stringa "@@@".

Si definisca la funzione:

```
Acquisto * leggiCrediti(char * fileName, int * dim);
```

che, ricevuto in ingresso il nome di un file, legga da tale file i dati di tutti gli acquisti e li restituisca tramite un array (allocato dinamicamente e della dimensione minima) di strutture dati di tipo **`Acquisto`**. Tramite il parametro **`dim`**, passato per riferimento, 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.

Si definisca la procedura **RICORSIVA**:

```
void stampaCrediti(Acquisto * crediti, int dim);
```

che, ricevuto in ingresso un puntatore ad un array di strutture dati di tipo **`Acquisto`**, e la sua dimensione **`dim`**, stampi in modo **RICORSIVO** il contenuto dell'array.

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 3A di Giovedì 14 Febbraio 2019 – tempo a disposizione 2h

Esercizio 2 – Liste, e Ordinamento (moduli element.h/.c e crediti.h/.c)

Il negozio di cartoleria condivide il file dei crediti con le aziende stesse, che lo usano anche per monitorare le proprie spese. In particolare, le aziende sono interessate a conoscere la lista completa degli articoli acquistati a credito. A tal scopo, si definisca la funzione:

```
list trovaArticoli(Acquisto * crediti, int dim, char * nomeAzienda);
```

che, ricevuto in ingresso un array di strutture dati di tipo **`Acquisto`**, la dimensione **`dim`** di tale array, e il nome di una azienda, restituisca una lista di strutture dati di tipo **`Articolo`**, contenente tutti gli articoli acquistati dall'azienda specificata (considerando tutti gli acquisti effettuati da quell'azienda).

Ovviamente le aziende vogliono sempre ridurre le spese: a tal scopo si definisca la funzione:

```
list ordinaArticoli(list articoli);
```

che, ricevuta in ingresso una lista di strutture dati di tipo **`Articolo`**, ne restituisca in uscita una nuova, contenente tutti gli articoli dati in ingresso ma ordinati in base al prezzo, dal più costoso al meno costoso.

Anche il negozio di cartoleria è interessato a controllare i propri clienti. Si definisca una procedura:

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

che, ricevuti in ingresso un vettore di strutture dati di tipo **`Acquisto`** e la dimensione di tale vettore, ordini il vettore secondo il seguente criterio: in ordine lessicografico crescente in base al nome dell'azienda che ha effettuato l'acquisto; a parità di nome, in ordine decrescente in base all'importo **totale** di un acquisto (gli acquisti più costosi vengono per primi). L'importo totale di un singolo acquisto è dato dalla somma dei costi degli articoli acquistati in tale singolo acquisto. A tal scopo, il candidato utilizzi l'algoritmo di ordinamento "merge sort" visto a lezione.

Esercizio 3 – Calcolo spese (modulo crediti.h/crediti.c)

Il negozio di cartoleria alla fine di ogni mese deve procedere a richiedere il pagamento di quanto dovuto dalle aziende. Quindi, per ogni azienda, deve sapere a quanto ammonta il totale dovuto, dato dalla somma di tutti gli acquisti. Si sviluppi una procedura:

```
void totali(Acquisto * v, int dim);
```

che, ricevuti in ingresso un array di strutture dati di tipo **`Acquisto`** e la dimensione di tale array, stampi a video la spesa totale attribuibile ad ogni singola azienda, relativamente a tutti gli acquisti memorizzati nell'array. Ovviamente, a video ogni singola azienda dovrà comparire una e una sola volta.

Esercizio 4 Stampa del totale dovuto da ogni azienda, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che, dopo aver letto dal file fornito nello StartKit i dati, stampi a video il totale dovuto da ogni azienda.

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 3A di Giovedì 14 Febbraio 2019 – tempo a disposizione 2h

"element.h":

```
#define _CRT_SECURE_NO_WARNINGS
#ifndef _ELEMENT_H
#define _ELEMENT_H

#define DIM_AZIENDA 16
#define DIM_ARTICOLO 16
#define DIM_ELENCO 32

typedef struct {
	char nome[DIM_ARTICOLO];
	float costo;
} Articolo;

typedef struct {
	char azienda[DIM_AZIENDA];
	Articolo elenco[DIM_ELENCO];
	int size;
} Acquisto;

typedef Articolo element;

int compareArticolo(Articolo a1, Articolo a2);
int compareAcquisto(Acquisto a1, Acquisto a2);
#endif
```

"element.c":

```
#define _CRT_SECURE_NO_WARNINGS
#include "element.h"

int compareArticolo(Articolo a1, Articolo a2) {
	int result;
	if (a1.costo > a2.costo)
		result = -1;
	else
		if (a1.costo < a2.costo)
			result = 1;
		else
			result = 0;
	return result;
}


int compareAcquisto(Acquisto a1, Acquisto a2) {
	int result;
	float tot1, tot2;
	int i;
	result = strcmp(a1.azienda, a2.azienda);
	if (result == 0) {
		tot1 = 0;
		tot2 = 0;
		for (i = 0; i < a1.size; i++)
			tot1 = tot1 + a1.elenco[i].costo;
		for (i = 0; i < a2.size; i++)
			tot2 = tot2 + a2.elenco[i].costo;
		if (tot1 > tot2)
```

```
                result = -1;
            else
                if (tot1 < tot2)
                    result = 1;
                else
                    result = 0;
        }
        return result;
}
```

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

Prova d'Esame 3A di Giovedì 14 Febbraio 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;

/* 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);
}
```

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

Prova d'Esame 3A di Giovedì 14 Febbraio 2019 – tempo a disposizione 2h

```
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("%s %6.2f\n",
                     temp.nome, temp.costo);
              showlist(tail(l));
              return;
       }
       else {
              printf("\n\n");
              return;
       }
}
void freelist(list l) {
       if (empty(l))
              return;
       else {
              freelist(tail(l));
              free(l);
       }
       return;
}

list insord_p(element el, list l) {
       list pprec = NULL, patt = l, paux;
       int trovato = 0;
       while (patt!=NULL && !trovato) {
              if (compareArticolo(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, 2018/2019 – Modulo 2

Prova d'Esame 3A di Giovedì 14 Febbraio 2019 – tempo a disposizione 2h

"credito.h":

```
#define _CRT_SECURE_NO_WARNINGS

#ifndef _CREDITO_H
#define _CREDITO_H


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

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

// Es. 1
Acquisto leggiUno(FILE * fp);
Acquisto * leggiCrediti(char * fileName, int * dim);
void stampaCrediti(Acquisto * crediti, int dim);

// Es. 2
list trovaArticoli(Acquisto * crediti, int dim, char * nomeAzienda);
list ordinaArticoli(list articoli);
void ordina(Acquisto * v, int dim);

// Es. 3
void totali(Acquisto * v, int dim);
#endif
```

"credito.c":

```
#include "credito.h"

// Es. 1
Acquisto leggiUno(FILE * fp) {
	Acquisto result;
	Articolo temp;
	int i;

	result.size = 0;
	i = 0;
	if (fscanf(fp, "%s", result.azienda) == 1) {
		fscanf(fp, "%s", temp.nome);
		while (i < DIM_ELENCO && strcmp(temp.nome, "fine") != 0) {
			fscanf(fp, "%f", &(temp.costo));
			result.elenco[i] = temp;
			i++;
			fscanf(fp, "%s", temp.nome);
		}
		result.size = i;
	}
	else {
		strcpy(result.azienda, "@@@");
	}
	return result;
}
```

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

Prova d'Esame 3A di Giovedì 14 Febbraio 2019 – tempo a disposizione 2h

```
Acquisto * leggiCrediti(char * fileName, int * dim) {
	FILE * fp;
	Acquisto * result;
	Acquisto temp;
	int count;

	result = NULL;
	*dim = 0;
	fp = fopen(fileName, "rt");
	if (fp!=NULL) {
		count = 0;
		temp = leggiUno(fp);
		while (strcmp(temp.azienda, "@@@") != 0) {
			count++;
			temp = leggiUno(fp);
		}
		if (count > 0) {
			rewind(fp);
			result = (Acquisto *)malloc(sizeof(Acquisto) * count);
			temp = leggiUno(fp);
			while (strcmp(temp.azienda, "@@@") != 0) {
				result[*dim] = temp;
				*dim = *dim + 1;
				temp = leggiUno(fp);
			}
		}
		fclose(fp);
	}
	else {
		printf("Errore nell'apertura del file: %s\n", fileName);
	}
	return result;
}

void stampaCrediti(Acquisto * crediti, int dim) {
	int i;
	if (dim <= 0)
		return;
	else {
		printf("%s\n", crediti[0].azienda);
		for (i = 0; i < crediti[0].size; i++)
			printf("\t%s %6.2f\n", crediti[0].elenco[i].nome,
crediti[0].elenco[i].costo);
		printf("\n");
		stampaCrediti(crediti + 1, dim - 1);
		return;
	}
}

// Es. 2
list trovaArticoli(Acquisto * crediti, int dim, char * nomeAzienda) {
	list result;
	int i;
	int j;

	result = emptylist();
```

```
	for (i = 0; i < dim; i++) {
		if (strcmp(crediti[i].azienda, nomeAzienda) == 0) {
			for (j = 0; j < crediti[i].size; j++) {
				result = cons(crediti[i].elenco[j], result);
			}
		}
	}
	return result;
}

list ordinaArticoli(list articoli) {
	list result;

	result = emptylist();
	while (!empty(articoli)) {
		result = insord_p(head(articoli), result);
		articoli = tail(articoli);
	}
	return result;
}

void  merge(Acquisto v[], int i1, int i2, int fine, Acquisto vout[]) {
	int i = i1, j = i2, k = i1;
	while (i <= i2 - 1 && j <= fine) {
		if (compareAcquisto(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(Acquisto v[], int first, int last, Acquisto 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);
	}
}

void ordina(Acquisto * v, int dim) {
	Acquisto * temp;
	temp = (Acquisto *) malloc(sizeof(Acquisto) * dim);
	mergeSort(v, 0, dim - 1, temp);
	free(temp);
}
```

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

Prova d'Esame 3A di Giovedì 14 Febbraio 2019 – tempo a disposizione 2h

```
// Es. 3
void totali(Acquisto * v, int dim) {
	char temp[DIM_AZIENDA];
	int i;
	list elenco;
	list elencoTemp;
	float totale;

	strcpy(temp, "@@@");
	ordina(v, dim);
	for (i = 0; i < dim; i++) {
		if (strcmp(v[i].azienda, temp) != 0) {
			totale = 0.0F;
			strcpy(temp, v[i].azienda);
			elenco = trovaArticoli(v, dim, temp);
			elencoTemp = elenco;
			while (!empty(elenco)) {
				totale = totale + head(elenco).costo;
				elenco = tail(elenco);
			}
			printf("Totale credito fatto all'azienda %s: %6.2f\n", temp, totale);
			freelist(elencoTemp);
		}
	}
	return;
}
```

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

Prova d'Esame 3A di Giovedì 14 Febbraio 2019 – tempo a disposizione 2h

"main.c":

```
#define _CRT_SECURE_NO_WARNINGS


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

#include "element.h"
#include "credito.h"

int main() {

	{ // Es. 1
		Acquisto * crediti;
		int dim;

		crediti = leggiCrediti("credito.txt", &dim);
		stampaCrediti(crediti, dim);
		free(crediti);
	}
	{ // Es. 2
		Acquisto * crediti;
		int dim;
		list elenco;
		list elencoOrd;

		crediti = leggiCrediti("credito.txt", &dim);
		elenco = trovaArticoli(crediti, dim, "SValentino");
		printf("Articoli della ditta %s:\n", "SValentino");
		showlist(elenco);
		elencoOrd = ordinaArticoli(elenco);
		showlist(elencoOrd);

		printf("\n\nAcquisti tutti ordinati:\n");
		ordina(crediti, dim);
		stampaCrediti(crediti, dim);

		freelist(elencoOrd);
		freelist(elenco);
		free(crediti);
	}

	{ // Es. 3
		Acquisto * crediti;
		int dim;

		crediti = leggiCrediti("credito.txt", &dim);
		totali(crediti, dim);
		free(crediti);
	}

	return 0;
}
```

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

Prova d'Esame 3A di Giovedì 14 Febbraio 2019 – tempo a disposizione 2h

"credito.txt":

```
Mapos cartellina 3.48 gomma 5.60 post-it 13.90 fine
SValentino bigliettini 60.50 penne 9.99 fine
TuffiSrl quaderno 2.60 matita 0.30 gomma 0.40
Mapos quaderno 5.20 fine TuffiSrl biglietto 4.50 fine
SValentino clipMetallo 1.10
pinzatrice 5.90
fine
```