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

Prova d'Esame 1A di Giovedì 9 Gennaio 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 famoso ristorante vuole realizzare un catalogo informatizzato dei piatti disponibili. A tal scopo ha cominciato a memorizzare in un unico file di testo alcune ricette, una ricetta diversa per ogni riga.

In particolare all'inizio di ogni riga c'è il nome di una **ricetta** (una stringa di al più 63 caratteri, senza spazi), e a seguire, separati da spazi, l'elenco degli ingredienti in tale formato: **nome** dell'**ingrediente** (stringa di al più 127 caratteri, senza spazi), uno spazio di separazione, **quantità** di tale ingrediente (un intero, quantità espressa in grammi), ulteriore spazio di separazione, altro ingrediente etc. etc.. Ogni riga è sempre terminata da un carattere di "a capo" (anche l'ultima riga). Inoltre ogni ricetta al massimo è composta di 10 ingredienti. A scopo esemplificativo si veda il file "**`ricette.txt`**" fornito nello StartKit.

Il ristorante ha poi realizzato un sito Web tramite cui un cliente può prenotare una cena e indicare eventuali esigenze alimentari legate a diete. Il sistema Web fornisce come output un file di testo, dove per ogni riga sono memorizzate le seguenti informazioni: il nome di un **ingrediente** e, separato da uno spazio, la **quantità** massima che il cliente può mangiare (in base alla sua dieta). Ad esempio, un cliente potrebbe dover evitare completamente la pasta e quindi indicherà "pasta 0": se una ricetta conterrà più di 0 grammi di pasta, allora il cliente non potrà mangiare tale piatto. Oppure un cliente potrebbe mangiare al massimo 1000 grammi di cipolla e quindi indicherà "cipolla 1000". Si veda, a titolo di esempio, il file "**`dieta.txt`**" fornito con lo StartKit.

Esercizio 1 – Strutture dati Ingrediente, Ricetta, e funzioni di lett./scritt. (mod. element.h e ricette.h/.c)

Si definisca un'opportuna struttura dati **`Ingrediente`**, al fine di rappresentare il nome di un ingrediente e la quantità di tale ingrediente. Si definisca poi la struttura dati **`Ricetta`**, al fine di rappresentare i dati relativi ad un ricetta, cioè il nome di tale ricetta, il numero di ingredienti usati ed un vettore di al più 10 strutture dati di tipo **`Ingrediente`**.
Si definisca la funzione:

```
Ingrediente leggiIngrediente(FILE * fp);
```

che, ricevuto in ingresso un puntatore ad un file, legga da tale file il nome e la quantità di un ingrediente, e restituisca tali valori tramite una struttura dati di tipo **`Ingrediente`**. Qualora vi siano problemi nella lettura, la funzione deve restituire un ingrediente dal nome "NULL" e quantità pari a zero.
Si definisca la funzione:

```
Ricetta * leggiRicette(char* fileName, int *dim);
```

che, ricevuto in ingresso il nome di un file di testo contenente le ricette, restituisca un array di strutture dati di tipo **`Ricetta`** allocato dinamicamente (della dimensione minima necessaria), contenente tutte le informazioni presenti nel file il cui nome è passato come parametro. Tramite il parametro **`dim`** la funzione deve restituire la dimensione del vettore. Qualora si verifichino dei problemi nell'apertura o nell'accesso al contenuto del file, la funzione deve restituire un puntatore a NULL e restituire il valore zero tramite il parametro **`dim`**.
Si definisca la procedura:

```
void stampaRicette(Ricetta* leRicette, int dim);
```

che, ricevuto in ingresso un array di strutture dati di tipo **`Ricetta`** e la dimensione **`dim`** di tale vettore, stampi a video il contenuto di tali strutture dati.
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 1A di Giovedì 9 Gennaio 2014 – tempo a disposizione 2h

Esercizio 2 – Lettura diete e ordinamento (moduli element.h/.c e ricette.h/.c)

Il candidato definisca una funzione:

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

che, ricevuto in ingresso il nome di un file contenente la dieta di un cliente, restituisca una lista di strutture dati di tipo **`Ingrediente`** opportunamente inizializzate con i dati presenti nel file. In caso di errore nell'apertura del file, il programma deve terminare.
Il candidato definisca poi una procedura:

```
void stampaDieta(list laDieta);
```

che, ricevuta in ingresso una lista di strutture dati di tipo **`Ingrediente`**, ne stampi a video il contenuto.

Il candidato definisca poi una procedura:

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

che, ricevuti in ingresso un vettore di strutture dati di tipo **`Ricetta`** e la dimensione di tale vettore, ordini il vettore secondo il seguente criterio: in maniera crescente in base al numero di ingredienti presenti in ogni ricetta; a parità di ingredienti, in ordine lessicografico in base al nome della ricetta. A tal fine, il candidato utilizzi l'algoritmo naïve sort visto a lezione.
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra.

Esercizio 3 –Identificazione delle ricette possibili (modulo ricette.h/ricette.c)

Una volta note l'elenco delle ricette e le restrizioni dovute alla dieta, è necessario stabilire quali ricette siano compatibili con i vincoli della dieta. A tal scopo, il candidato definisca una funzione:

```
Ricetta * filtra(Ricetta * leRicette, int dim, list laDieta, int * dimR);
```

che, ricevuti come parametri un vettore di strutture dati di tipo **`Ricetta`** e la sua dimensione **`dim`** e una lista contenente i vincoli della dieta, restituisca un nuovo vettore di strutture dati di tipo **`Ricetta`** e la sua dimensione **`dimR`** (non necessariamente della dimensione minima), dove siano state inserite tutte le ricette compatibili. Una ricetta è compatibile con una dieta se tutti gli ingredienti contenuti nella ricetta o (a) non sono soggetti a restrizioni della dieta, o (b) la ricetta prevede per un certo ingrediente una quantità minore o uguale al limite imposto dalla dieta. A tal scopo si suggerisce di implementare le seguenti funzioni ausiliarie:

```
int ingredienteCompatibile(Ingrediente ing, list laDieta);
```

che valuta se un certo ingrediente è compatibile con una dieta specificata;
e la funzione:

```
int ricettaCompatibile(Ricetta r, list laDieta);
```

che valuta se una ricetta è compatibile con una dieta.

Esercizio 4 – Stampa dei progetti finanziabili, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **`main(…)`** un programma che, usando le informazioni fornite tramite i file di esempio forniti nello StartKit, e le funzioni definite agli esercizi precedenti, stampi a video nell'ordine specificato all'esercizio 2 le ricette compatibili con i vincoli dati dalla dieta.

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 1A di Giovedì 9 Gennaio 2014 – tempo a disposizione 2h

"element.h":

```
#ifndef _ELEMENT_H
#define _ELEMENT_H

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

#define DIM_NOME_INGREDIENTE 128
#define DIM_NOME_RICETTA 64

typedef struct {
	char nome[DIM_NOME_INGREDIENTE];
	int quantita;
} Ingrediente;

typedef struct {
	char nome[DIM_NOME_RICETTA];
	int numIngredienti;
	Ingrediente elenco[10];
} Ricetta;


typedef Ingrediente element;

int compare(Ricetta r1, Ricetta r2);

#endif
```

"element.c":

```
#include "element.h"


int compare(Ricetta r1, Ricetta r2) {
	if (r1.numIngredienti != r2.numIngredienti)
		return r1.numIngredienti - r2.numIngredienti;
	else
		return strcmp(r1.nome, r2.nome);
}
```

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

Prova d'Esame 1A di Giovedì 9 Gennaio 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;
}
```

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

Prova d'Esame 1A di Giovedì 9 Gennaio 2014 – tempo a disposizione 2h

"ricette.h":

```
#ifndef _RICETTE_H
#define _RICETTE_H

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

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

// es. 1
Ingrediente leggiIngrediente(FILE * fp);
Ricetta * leggiRicette(char* fileName, int *dim);
void stampaRicette(Ricetta* leRicette, int dim);

// es. 2
list leggiDieta(char * fileName);
void stampaDieta(list laDieta);
void ordina(Ricetta * v, int dim);

// es. 3
Ricetta * filtra(Ricetta * leRicette, int dim, list laDieta, int * dimR);


#endif
```

"ricette.c":

```
#include "ricette.h"

Ingrediente leggiIngrediente(FILE * fp) {
	Ingrediente result;
	if (fscanf(fp, "%s%d", result.nome, &(result.quantita)) != 2) {
		strcpy(result.nome, "NULL");
		result.quantita = 0;
	}
	return result;
}

Ricetta * leggiRicette(char* fileName, int *dim) {
	Ricetta *  result = NULL;
	Ricetta temp;
	FILE * fp;
	int i;
	char ch;

	*dim = 0;
	fp = fopen(fileName, "rt");
	if (fp != NULL) {
		while (fscanf(fp, "%s", temp.nome) == 1) {
			ch = fgetc(fp);
			while (ch!='\n' && strcmp(leggiIngrediente(fp).nome, "NULL")!=0)
				ch = fgetc(fp);
```

```
                        *dim = *dim + 1;
                }
                result = (Ricetta*) malloc(sizeof(Ricetta) * *dim);
                rewind(fp);
                for (i=0; i<*dim; i++) {
                                fscanf(fp, "%s", result[i].nome);
                                result[i].numIngredienti = 0;
                                ch = fgetc(fp);
                                while (ch!='\n') {
                                        result[i].elenco[result[i].numIngredienti] =
leggiIngrediente(fp);
                                        result[i].numIngredienti += 1;
                                        ch = fgetc(fp);
                                }
                }
                fclose(fp);
        }
        return result;
}

void stampaRicette(Ricetta* leRicette, int dim) {
        int i, j;
        for (i=0; i<dim; i++) {
                printf("%s ", leRicette[i].nome);
                for (j=0; j<leRicette[i].numIngredienti; j++)
                        printf("%s gr. %d, ", leRicette[i].elenco[j].nome,
leRicette[i].elenco[j].quantita);
                printf("\n");
        }
        return;
}


list leggiDieta(char * fileName) {
        list result = emptylist();
        FILE * fp;
        Ingrediente temp;
        fp = fopen(fileName, "rt");
        if (fp == NULL) {
                printf("Errore nell'aprire il file %s\n", fileName);
                exit(1);
        }
        else {
                temp=leggiIngrediente(fp);
                while (strcmp(temp.nome, "NULL") != 0) {
                        result = cons(temp, result);
                        temp=leggiIngrediente(fp);
                }
                fclose(fp);
        }
        return result;
}

void stampaDieta(list laDieta) {
        Ingrediente temp;
        while (!empty(laDieta)) {
                temp = head(laDieta);
```

```
            printf("%s gr. %d\n", temp.nome, temp.quantita);
            laDieta =  tail(laDieta);
      }
      return;
}

void scambia(Ricetta *a, Ricetta *b) {
      Ricetta tmp = *a;
      *a = *b;
      *b = tmp;
}

int trovaPosMax(Ricetta v[], int n){
      int i, posMax=0;
      for (i=1; i<n; i++)
            if (compare(v[posMax],v[i])<0)
                  posMax=i;
      return posMax;
}

void naiveSort(Ricetta v[], int n){
      int p;
      while (n>1) {
            p = trovaPosMax(v,n);
            if (p<n-1)
                  scambia(&v[p],&v[n-1]);
            n--;
      }
}

void ordina(Ricetta * v, int dim) {
      naiveSort(v, dim);
}

int ingredienteCompatibile(Ingrediente ing, list laDieta) {
      int result = 1;
      int trovato = 0;
      Ingrediente temp;
      while (!empty(laDieta) && !trovato) {
            temp = head(laDieta);
            if (strcmp(temp.nome, ing.nome)==0)
                  trovato = 1;
            if (trovato && ing.quantita>temp.quantita)
                  result = 0;
            laDieta = tail(laDieta);
      }
      return result;
}

int ricettaCompatibile(Ricetta r, list laDieta) {
      int result = 1;
      int i;

      for (i=0; i<r.numIngredienti && result; i++) {
            if (!ingredienteCompatibile(r.elenco[i], laDieta)) {
                  result = 0;
            }
```

```
        }

        return result;
}


Ricetta * filtra(Ricetta * leRicette, int dim, list laDieta, int * dimR) {
        Ricetta * result;
        int i;

        *dimR = 0;
        result = (Ricetta *) malloc(sizeof(Ricetta) * dim);

        for (i=0; i<dim; i++) {
                if (ricettaCompatibile(leRicette[i], laDieta)) {
                        result[*dimR] = leRicette[i];
                        *dimR += 1;
                }
        }
        return result;
}
```

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

Prova d'Esame 1A di Giovedì 9 Gennaio 2014 – tempo a disposizione 2h

"main.c":

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

#include "element.h"
#include "ricette.h"



int main() {
	// Es. 1
	{
		Ricetta * leRicette;
		int dim;
		printf("\n\n Es. 1:\n");
		leRicette = leggiRicette("ricette.txt", &dim);
		stampaRicette(leRicette, dim);
		free(leRicette);
	}
	// Es. 2
	{
		Ricetta * leRicette;
		int dim;
		list laDieta;
		printf("\n\n Es. 2:\n");
		laDieta = leggiDieta("dieta.txt");
		stampaDieta(laDieta);
		leRicette = leggiRicette("ricette.txt", &dim);
		ordina(leRicette, dim);
		stampaRicette(leRicette, dim);
		free(leRicette);
		freelist(laDieta);
	}
	// Es. 3 & 4
	{
		Ricetta * leRicette;
		int dim;
		Ricetta * possibili;
		int dimP;
		list laDieta;
		printf("\n\n Es. 3 & 4:\n");
		laDieta = leggiDieta("dieta.txt");
		leRicette = leggiRicette("ricette.txt", &dim);
		possibili = filtra(leRicette, dim, laDieta, &dimP);
		ordina(possibili, dimP);
		stampaRicette(possibili, dimP);
		free(leRicette);
		free(possibili);
		freelist(laDieta);
	}
	system("pause");
	return 0;
}
```

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

Prova d'Esame 1A di Giovedì 9 Gennaio 2014 – tempo a disposizione 2h

"ricette.txt":

```
pastaConCipolla pasta 100 cipolla 1000
patateConSalsicca salsiccia 500 patate 1000 cipolla 200 burro 50 olio 50
tomino formaggio 100 speck 50
insalata insalata 200
```

"dieta.txt":

```
olio 40
burro 30
pasta 80
formaggio 200
```