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

Prova d'Esame 3A di Giovedì 20 Febbraio 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"*.

La pubblica amministrazione acquista i beni tramite un nuovo sistema informatico che garantisce buoni prezzi ma che vincola all'acquisto di un numero minimo di pezzi. In particolare in un file di testo denominato "**`forniture.txt`**" sono elencati i beni acquistabili: in ogni riga sono elencate le informazioni relative ad un bene, cioè un **identificatore unico** del bene acquistabile (una stringa di al più 15 caratteri utili, senza spazi), uno spazio di separazione e poi il **prezzo** per singolo pezzo (un float), uno spazio di separazione e il numero minimo di pezzi da acquistare per poter usufruire del prezzo specificato (un intero), e infine dopo un spazio una **descrizione** del bene (una stringa di al più 2047 caratteri utili, contenente spazi). Tutte le righe del file sono sempre terminate da un 'new line', compresa l'ultima riga.

Esercizio 1 – Strutture dati Bene, e funzioni di lett./scritt. (mod. element.h/c e beni.h/.c)

Si definisca un'opportuna struttura dati **`Bene`**, al fine di rappresentare un bene disponibile per l'acquisto.
Si definisca la funzione:

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

che, ricevuto in ingresso il nome di un file contenente i beni disponibili, legga tali informazioni e restituisca l'elenco dei beni tramite una lista di strutture dati di tipo **`Bene`**. In caso di errore nell'apertura del file la funzione deve stampare un messaggio di errore, e poi deve chiedere esplicitamente all'utente se si vuole riprovare ad aprire il file: l'utente deve specificare una risposta tramite un solo carattere ('s' o 'n', seguito da invio), in base al quale il programma deve riprovare ad aprire il file oppure terminare immediatamente. Il programma deve continuare a provare ad aprire il file fino a che non ha successo e fino a che l'utente continua a specificare la risposta 's'.
Si definisca poi una funzione:

```
list ordinaBeni(list elenco);
```

che, ricevuto in ingresso una lista di strutture dati di tipo **`Bene`**, restituisca una nuova lista contenente le stesse strutture dati fornite in ingresso, ma ordinate in base alla descrizione del bene (ordine lessicografico), e a parità di descrizione in base al prezzo per unità, in senso crescente. Per generare la nuova lista, si utilizzi una delle funzioni di inserimento ordinato per le liste viste a lezione, e non si usino gli algoritmi di ordinamento per array.
Si definisca infine la procedura:

```
void stampaBeni(list elenco);
```

che, ricevuta in ingresso una lista di strutture dati di tipo **`Bene`**, stampi a video tale elenco.
Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra.

Esercizio 2 – Ricerca dei beni e ordinamento (modulo element.h e beni.h/.c)

Gli impiegati della pubblica amministrazione effettuano ricerche sui beni da acquistare tramite delle "parole-chiave": partono cioè da una parola (una stringa senza spazi), e cercano tutti i beni la cui descrizione menziona tale parola. A tal scopo il candidato definisca una funzione:

```
Bene * estrai(list elenco, char * keyword, int * dim)
```

che, ricevuto in ingresso una lista di strutture dati di tipo **`Bene`** e una stringa rappresentante la parola chiave, restituisca un array di strutture dati di tipo **`Bene`**, allocato dinamicamente e della dimensione minima necessaria, contenente tutti i beni nella cui descrizione è presente la parola chiave specificata. Tramite il parametro **`dim`** passato per riferimento la funzione deve restituire la dimensione del vettore risultato. Qualora non vi siano beni la cui descrizione contiene la parola-chiave cercata, la funzione deve restituire un

puntatore a NULL e dimensione pari a zero. 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 definisca poi una procedura:

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

che, ricevuti in ingresso un vettore di strutture dati di tipo **Bene** e la dimensione di tale vettore, ordini il vettore secondo il seguente criterio: in ordine lessicografico in base alla descrizione del bene, e a parità di descrizione in base all'importo in senso crescente. A tal fine, il candidato utilizzi l'algoritmo "Merge Sort" visto a lezione.

Il candidato definisca poi una procedura:

```
void stampaVettBeni(Bene * iBeni, int dim);
```

che, ricevuta in ingresso un array di strutture dati di tipo **Bene**, e la sua dimensione **dim**, stampi a video 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.

Esercizio 3 –Identificazione dei beni effettivamente acquistabili (modulo beni.h/beni.c)

Una volta determinati i beni acquistabili tramite la ricerca con parola-chiave, gli impiegati della pubblica amministrazione devono compiere un ulteriore controllo: infatti un tipico acquisto è basato sul numero di beni da acquistare, ed il budget massimo che si può spendere a riguardo (cioè considerando per il singolo ordine la funzione [prezzo singolo pezzo * numero di pezzi]); inoltre deve sempre essere selezionata l'offerta che risulti essere più economica. Al fine di supportare questo ulteriore controllo il candidato implementi una funzione:

```
Bene effettivo(list elenco, char * keyword, int num, float budget)
```

che, ricevuti come parametri una lista di strutture dati di tipo **Bene**, una parola-chiave da cercare, il numero di pezzi da acquistare e il budget disponibile, restituisca l'offerta più vantaggiosa per il bene da acquistare. A tal scopo il candidato utilizzi la funzione di cui al punto precedente per trovare i beni la cui descrizione contiene la parola-chiave specificata. Il candidato poi abbia cura di verificare nella funzione che il numero di pezzi da acquistare sia maggiore o uguale al numero minimo specificato nella struttura dati di tipo **Bene**, e che il costo totale sia inferiore o uguale al budget specificato. Infine il candidato si assicuri che la struttura dati restituita sia effettivamente la più economica possibile.

Qualora non vi sia alcun bene acquistabile secondo i criteri specificati, la funzione restituisca una struttura dati il cui identificatore unico sia "NULL".

Esercizio 4 – Ricerca dei beni acquistabili, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **main(…)** un programma che chieda all'utente di specificare una parola chiave, il numero di pezzi da acquistare ed il budget disponibile, e stampi a video le informazioni relative al bene più economico da acquistare, o un messaggio di errore qualora non sia possibile soddisfare le richieste. Ad esempio, utilizzando il file di esempio fornito nello StartKit, e indicando come parola-chiave la stringa "carta", come numero di pezzi da acquistare 6, e come budget euro 17.00, il programma deve indicare come bene più economico quello con identificatore "ab147".

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 3A di Giovedì 20 Febbraio 2014 – tempo a disposizione 2h

"element.h":

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

#ifndef _ELEMENT_H
#define _ELEMENT_H

#define DIM_DESCRIZIONE 2048
#define DIM_ID_BENE 16


typedef struct {
	char id[DIM_ID_BENE];
	float prezzo;
	int dim_minima;
	char testo[DIM_DESCRIZIONE];
} Bene;


typedef Bene element;

int compareBene(Bene b1, Bene b2);

#endif
```

"element.c":

```
#include "element.h"


int compareBene(Bene b1, Bene b2) {
	int result;
	float diff;
	result = strcmp(b1.testo, b2.testo);
	if (result == 0) {
		diff = b1.prezzo - b2.prezzo;
		if (diff<0)
			result = -1;
		else if (diff>0)
			result = 1;
	}
	return result;
}
```

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

Prova d'Esame 3A di Giovedì 20 Febbraio 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 (compareOggetto(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 3A di Giovedì 20 Febbraio 2014 – tempo a disposizione 2h

"beni.h":

```
#ifndef _BENI
#define _BENI

#include "stdio.h"
#include "stdlib.h"

#include "list.h"


list leggiBeni(char* fileName);
list ordinaBeni(list elenco);
void stampaBeni(list elenco);

Bene * estrai(list elenco, char * keyword, int * dim);
void ordina(Bene * v, int dim);
void stampaVettBeni(Bene * iBeni, int dim);

Bene effettivo(list elenco, char * keyword, int num, float budget);


#endif
```

"beni.c":

```
#include "beni.h"



list leggiBeni(char* fileName) {
	list result;
	FILE * fp;
	char answer;
	int aperto;
	Bene temp;
	char ch;
	int i;
	int read;

	answer = 'n';
	aperto = 0;
	do {
		fp = fopen(fileName, "rt");
		if (fp == NULL) {
			printf("Errore durante l'apertura del file %s. Vuoi riprovare [s/n]
?");
			scanf("%c%*c", &answer);
			if (answer=='n') {
				system("pause");
				exit(-1);
			}
		}
		else
			aperto=1;
	} while(aperto==0 && answer == 's');
```

```
	result = emptylist();

	do {
		read = fscanf(fp, "%s%f%d", temp.id, &(temp.prezzo), &(temp.dim_minima));
		if (read == 3) {
			i=0;
			do {
				ch = fgetc(fp);
				if (ch!='\n')
					temp.testo[i++] = ch;
			} while (ch!='\n');
			temp.testo[i] = '\0';
			result = cons(temp, result);
		}
	} while (read == 3);

	fclose(fp);
	return result;
}


list ordinaBeni(list elenco) {
	list result;

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

	return result;
}

void stampaBeni(list elenco) {
	Bene temp;
	while (!empty(elenco)) {
		temp = head(elenco);
		printf("%s %f %d %s\n", temp.id, temp.prezzo, temp.dim_minima, temp.testo);
		elenco = tail(elenco);
	}
	printf("\n\n");
	return;
}


Bene * estrai(list elenco, char * keyword, int * dim) {
	list temp;
	Bene * result = NULL;
	Bene tempBene;
	int count;

	*dim = 0;
	count = 0;
	temp = elenco;
	while (!empty(temp)) {
		tempBene = head(temp);
		if (strstr(tempBene.testo, keyword)!=NULL)
```

```
                count++;
            temp = tail(temp);
        }
        if (count > 0) {
            result = (Bene*) malloc(sizeof(Bene) * count);
            temp = elenco;
            while (!empty(temp)) {
                tempBene = head(temp);
                if (strstr(tempBene.testo, keyword)!=NULL) {
                    result[(*dim)++] = tempBene;
                }
                temp = tail(temp);
            }
        }
        return result;
}


void merge(Bene v[], int i1, int i2, int fine, Bene vout[]){
        int i=i1, j=i2, k=i1;
        while ( i <= i2-1 && j <= fine ) {
            if (compareBene(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(Bene v[], int first, int last, Bene 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(Bene * v, int dim) {
        Bene * temp;
        temp = (Bene*) malloc(sizeof(Bene) * dim);
        mergeSort(v, 0, dim-1, temp);
        free(temp);
        return;
}

void stampaVettBeni(Bene * iBeni, int dim) {
        int i;
        for (i=0; i<dim; i++) {
            printf("%s %f %d %s\n", iBeni[i].id, iBeni[i].prezzo, iBeni[i].dim_minima,
iBeni[i].testo);
        }
```

```
        printf("\n\n");
        return;
}



Bene effettivo(list elenco, char * keyword, int num, float budget) {
        Bene * temp;
        Bene result;
        int dimTemp;
        int i;
        float prezzo;
        int trovato;

        strcpy(result.id, "NULL");
        trovato = 0;
        temp = estrai(elenco, keyword, &dimTemp);
        if (dimTemp > 0) {
                for (i=0; i<dimTemp; i++) {
                        if (num >= temp[i].dim_minima && num*temp[i].prezzo<=budget) {
                                if (trovato) {
                                        if (temp[i].prezzo < result.prezzo)
                                                result = temp[i];
                                }
                                else {
                                        result = temp[i];
                                        trovato = 1;
                                }
                        }
                }
                free(temp);
        }
        return result;
}
```

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

Prova d'Esame 3A di Giovedì 20 Febbraio 2014 – tempo a disposizione 2h

"main.c":

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

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



int main() {

	{ // es. 1
		list elenco;
		list elencoOrd;

		elenco = leggiBeni("forniture.txt");
		stampaBeni(elenco);
		elencoOrd = ordinaBeni(elenco);
		stampaBeni(elencoOrd);
		freelist(elenco);
		freelist(elencoOrd);
	}

	{ // es. 2
		list elenco;
		list elencoOrd;
		Bene * v;
		int dim;

		elenco = leggiBeni("forniture.txt");
		elencoOrd = ordinaBeni(elenco);
		v = estrai(elencoOrd, "carta", &dim);
		ordina(v, dim);
		stampaVettBeni(v, dim);
		free(v);
		freelist(elenco);
		freelist(elencoOrd);
	}

	{ // es. 3 && 4
		list elenco;
		Bene result;
		int dim;
		char keyword[2048];
		int num;
		float budget;

		elenco = leggiBeni("forniture.txt");
		printf("Parola chiave? ");
		scanf("%s", keyword);
		printf("numero di pezzi da acquistare? ");
		scanf("%d", &num);
		printf("Budget? ");
		scanf("%f", &budget);
		result = effettivo(elenco, keyword, num, budget);
```

```
            if (strcmp(result.id, "NULL")) {
                  printf("Bene piu' economico da acquistare:\n");
                  printf("%s %f %d %s\n", result.id, result.prezzo, result.dim_minima,
result.testo);
            }
            else {
                  printf("Non esiste un bene acquistabile compatibile con i vincoli
introdotti.\n");
            }

            freelist(elenco);
      }

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

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

Prova d'Esame 3A di Giovedì 20 Febbraio 2014 – tempo a disposizione 2h

"forniture.txt":

```
ab146 1.09 20 carta stampante laser getto inchiostro appunti carta comune appunti
ab147 2.12 5 carta stampante laser getto inchiostro appunti carta comune appunti
ab148 2.60 1 carta stampante laser getto inchiostro appunti carta comune appunti
cf689 245.00 10 toner stampante laser colore black
cf700 189.00 10 toner stampante laser ufficio colore cyan
cf701 189.00 10 toner stampante laser ufficio colore magenta
cf700 189.00 10 toner stampante laser ufficio colore yellow
```