Fondamenti di Informatica T-1, 2009/2010 – Modulo 2
Prova d'Esame 6 di Giovedì 9 Settembre 2010 – tempo a disposizione 2h30'

Prima di cominciare: si scarichi dal sito **`http://esamix.labx`** il file **`StartKit6.zip`** contenente i file necessari (*solution* di VS2008 e progetto compresi).
Avvertenze per la consegna: apporre all'inizio di ogni file sorgente un commento contenente i propri dati (**cognome, nome, numero di matricola**) e **l'identificativo (A/B)** 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*, effettuare di tanto in tanto *"Rebuild All"*.

Una ditta è specializzata nella produzione di componenti meccanici complessi, caratterizzati dal fatto che ogni componente è a sua volta composto di altri elementi meccanici. La ditta memorizza tramite un unico sistema informatico la disponibilità in magazzino sia dei componenti complessi, sia dei componenti più semplici che li compongono. In un file di testo, in ogni riga, vengono salvate le seguenti informazioni relativamente ad ogni componente (in ogni riga informazioni relative ad un solo componente):

- il **`codice`** identificativo unico del componente (un intero);
- una **`descrizione`** del componente (stringa di al più 63 caratteri senza spazi);
- la **`disponibilità`** in magazzino del componente in questione (un intero);
- il **`numero`** di altri componenti di cui l'oggetto in questione è composto (un intero); i componenti semplici sono composti da zero altri elementi; si assuma inoltre per semplicità che un componente complesso è composto al massimo da altri 16 elementi;
- a seguire, i **`codici identificativi`** degli elementi di cui è composto il componente (interi, separati da spazi); vi sono tanti codici identificativi quanti specificati nell'intero di cui alla voce precedente (come già detto prima, al massimo 16).

Attenzione: ciascuna funzione realizzata deve essere opportunamente testata invocandola correttamente dal **`main(...)`** e mostrando sullo standard output il risultato dell'elaborazione effettuata. Inoltre tutta la memoria allocata dinamicamente (quindi anche gli elementi delle liste) deve essere opportunamente deallocata alla fine della computazione di ciascun esercizio.

Esercizio 1 – Strutture dati e funzioni di scrittura/lettura/stampa (moduli element.h e componenti.h/componenti.c)

a) Si definisca un'opportuna struttura dati **`Componente`** al fine di rappresentare i dati relativi ad un componente, come descritto in precedenza;
b) Si realizzi la funzione

```
Componente leggiUnComponente(FILE * fp);
```

che, ricevuto un puntatore **`fp`** ad un file di testo già aperto, legga i dati relativi ad un solo componente e restituisca tali dati come risultato, memorizzati in una struttura dati di tipo **`Componente`**. Qualora la lettura non vada a buon fine (perchè, ad esempio, si è raggiunta la fine del file), la funzone deve restituire una struttura dati di tipo **`Componente`** il cui campo descrizione contenga la stringa "**`ERRORE`**";
c) Si realizzi la funzione

```
Componente* leggi(char * fileName, int* dim);
```

che, ricevuto il nome di un file di testo contenente un elenco di informazioni relative a dei componenti (come specificato all'inizio del testo), legga i dati ivi contenuti e li memorizzi in un vettore di strutture dati di tipo **`Componente`** di dimensione opportuna. L'array restituito deve essere della dimensione esatta sufficiente a contenere tutti gli elementi memorizzati nel file. Tale dimensione deve essere restituita tramite il parametro **`dim`**. Come esempio, ed al fine di testare opportunamente il codice realizzato, il candidato può usare il file "**`disp.txt`**" fornito nello StartKit.
d) Si realizzi poi la procedura

```
void stampaComponenti(Componente *vector, int n);
```

che, ricevuto in ingresso un vettore di strutture dati di tipo **Componente**, stampi i primi **n** elementi contenuti nel vettore.

Esercizio 2 – Ordinamento in base alla descrizione (element.h/componenti.h/componenti.c)

Al fine di facilitare l'accesso all'elenco dei componenti da parte di operatori umani, il sistema effettua un ordinamento dei componenti in base alla loro disponibilità (in ordine crescente), e poi in base alla loro descrizione (secondo il criterio lessicografico). Si definisca la funzione:

```
Componente * ordina(Componente * elenco, int dim);
```

che, ricevuto un vettore **elenco** di strutture dati di tipo **Componente** e la sua dimensione **dim**, restituisce il vettore ordinato in base alla descrizione, secondo il criterio lesicografico. Suggerimento: si definisca in "element.h/element.c" una funzione **compare(...)** ausiliaria per confrontare due strutture dati di tipo **Componente**.

Esercizio 3 – Lista dei componenti esauriti (componenti.h/componenti.c)

Ai fini della programmazione del lavoro, è necessario avere una lista dei componenti complessi (cioè composti da un numero di sottoparti maggiore di zero) la cui disponibilità corrente è pari a zero. A tal fine si definisca una funzione:

```
list esauriti(Componente * elenco, int dim);
```

che, ricevuta in ingresso un vettore di strutture dati di tipo **Componente**, e la sua dimensione **dim**, restituisca in uscita una lista di strutture dati **Componente** attualmente con disponibilità pari a zero, e che siano caratterizzati dall'essere composti di più parti. La lista restituita in uscita deve essere ordinata con criterio lessicografico.

Esercizio 4 – Verifica della fattibilità (modulo componenti.h/componenti.c)

Una volta noti quali sono i componenti complessi non disponibili (e quindi da rimettere in produzione al più presto), si rende necessario valutare se è possibile produrre tali componenti, in base alle sottoparti di cui sono composti: al fine di produrre un componente complesso, tutte le sue sottoparti devono poter essere disponibili in magazzino (campo disponiblità maggiore di zero). Può succedere che a sua volta un sottocomponente sia di tipo "complesso", e che non sia disponibile: in tal caso bisogna valutare se le ulteriori sottoparti siano effettivamente disponibili. Si definisca una funzione ricorsiva:

```
int costruibile(Componente c, Componente * elenco, int dim);
```

che, ricevuta in ingresso una struttura dati di tipo **Componente**, valuti se è possibile costruire il componente in base alla disponibilità delle sottoparti. La funzione deve restituire un intero che possa essere valutato con significato booleano (true se è possibile costruire il componente, false altrimenti). Il componente è "costruibile" se per tutte le sue sottoparti vale: a) la sottoparte è una sottoparte semplice ed è disponibile, o b) la sottoparte è complessa e può essere costruita a sua volta. Si assuma, per definizione, che un componente semplice è "costruibile" se ha disponibilità maggiore di zero, "non costruibile" altrimenti.

Dove specificato realizzare le funzioni nei moduli indicati; laddove non specificato si lascia libertà di scelta al candidato.

NOTA: Si abbia cura di deallocare la memoria allocata dinamicamente dal programma realizzato.

Fondamenti di Informatica T-1, 2009/2010 – Modulo 2
Prova d'Esame 6 di Giovedì 9 Settembre 2010 – tempo a disposizione 2h30'

"element.h"

```
#ifndef _ELEMENT_H
#define     _ELEMENT_H

#define DIM_DESC 64
#define MAX_SOTTOPARTI 16

typedef struct
{
      int codice;
      char desc[DIM_DESC];
      int disp;
      int numSottoparti;
      int sottoparti[MAX_SOTTOPARTI];
} Componente;

typedef Componente element;

int compare(element el1, element el2);

#endif      /* _ELEMENT_H */
```

"element.c"

```
#include "element.h"

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


int compare(element el1, element el2) {
      if (el1.disp>el2.disp)
            return 1;
      else
            if (el1.disp<el2.disp)
                  return -1;
            else
                  return strcmp(el1.desc, el2.desc);
}
```

"componenti.h"

```
#ifndef COMPONENTI
#define COMPONENTI

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

Componente leggiUnComponente(FILE * fp);
Componente* leggi(char * fileName, int* dim);
void stampaComponenti(Componente *vector, int n);
Componente * ordina(Componente * elenco, int dim);
list esauriti(Componente * elenco, int dim);
int costruibile(Componente c, Componente * elenco, int dim);

#endif
```

Fondamenti di Informatica T-1, 2009/2010 – Modulo 2
Prova d'Esame 6 di Giovedì 9 Settembre 2010 – tempo a disposizione 2h30'

"componenti.c"

```
#include "componenti.h"

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

Componente leggiUnComponente(FILE * fp) {
	Componente result;
	int i, readOK = 1;
	if (
		fscanf(fp, "%d %s %d %d", &result.codice, result.desc, &result.disp,
&result.numSottoparti) == 4 )
	{
			for (i=0; i<result.numSottoparti && readOK; i++)
				if (fscanf(fp, "%d", &(result.sottoparti[i])) != 1)
					readOK = 0;
	}
	else
		readOK = 0;
	if (!readOK)
		strcpy(result.desc, "ERRORE");
	return result;
}


Componente* leggi(char * fileName, int* dim) {
	FILE * fp;
	int count = 0;
	int i;
	Componente * result;
	Componente temp;

	if ((fp=fopen(fileName, "r")) == NULL) {
		printf("Errore di apertura del file %s\n", fileName);
		exit(-1);
	}

	// conta degli elementi memorizzati nel file
	do {
		temp = leggiUnComponente(fp);
		if (strcmp(temp.desc,"ERRORE") != 0)
			count++;
	}
	while (strcmp(temp.desc, "ERRORE") != 0);

	result = (Componente *) malloc(sizeof(Componente) * count);
	rewind(fp);

	// lettura dei componenti
	for (i=0; i< count; i++) {
		result[i] = leggiUnComponente(fp);
	}

	fclose(fp);
	*dim = count;
	return result;
}
```

Fondamenti di Informatica T-1, 2009/2010 – Modulo 2
Prova d'Esame 6 di Giovedì 9 Settembre 2010 – tempo a disposizione 2h30'

```
void stampaComponenti(Componente *vector, int n) {
      int i;
      int j;
      for (i=0; i<n; i++) {
            printf("%d %s %d %d ", vector[i].codice, vector[i].desc,
vector[i].disp, vector[i].numSottoparti);
            for (j=0; j<vector[i].numSottoparti; j++)
                  printf("%d ", vector[i].sottoparti[j]);
            printf("\n");
      }
}


void insOrd(Componente v[], int pos) {
      int i = pos-1;
      Componente x = v[pos];

      while (i>=0 && compare(x,v[i])<0) {
            v[i+1]= v[i];
            i--;
      }
      v[i+1]=x; /* inserisce l'elemento */
}

void insertSort(Componente v[], int n){
      int k;
      for (k=1; k<n; k++)
            insOrd(v,k);
}



Componente * ordina(Componente * elenco, int dim) {
      insertSort(elenco, dim);
      return elenco;
}


list esauriti(Componente * elenco, int dim) {
      list result;
      int i;

      result = emptylist();
      for (i=0; i<dim; i++) {
            if (elenco[i].disp==0 && elenco[i].numSottoparti>0)
                  result = insord_p(elenco[i], result);
      }
      return result;
}

/*
 Questa funzione valuta se un componente e' costruibile:
 - se il componente e' semplice ed ha disponibilita' pari a zero,
   non e' costruibile
 - se il componente e' semplice ed ha disponibilita' maggiore di zero,
   allora e' costruibile
 - se il componente e' complesso, viene valutata la sua costruibilita'
```

```
   indipendentemente dalla sua disponibilita'
   */
int costruibile(Componente c, Componente * elenco, int dim) {
      int result = 1;
      int i;
      int j;
      int trovato;
      Componente temp;

      if (c.numSottoparti == 0)
            if (c.disp>0) result = 1;
            else result=0;
      else
            for (i=0; i<c.numSottoparti && result; i++) {
                  trovato = 0;
                  for (j=0; j<dim && !trovato; j++)
                        if (c.sottoparti[i] == elenco[j].codice) {
                              temp = elenco[j];
                              trovato = 1;
                        }
                  result = result * costruibile(temp, elenco, dim);
            }
      return result;
}
```

Fondamenti di Informatica T-1, 2009/2010 – Modulo 2
Prova d'Esame 6 di Giovedì 9 Settembre 2010 – tempo a disposizione 2h30'

"main.c"

```
#include "element.h"
#include "componenti.h"

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


int main() {
	// TEST Es. 1
	{
		Componente * elenco;
		int dim;
		elenco = leggi("disp.txt", &dim);
		stampaComponenti(elenco, dim);
		free(elenco);
	}

	// TEST Es. 2
	{
		Componente * elenco;
		int dim;
		printf("\n\n");
		elenco = leggi("disp.txt", &dim);
		stampaComponenti(elenco, dim);
		elenco = ordina(elenco, dim);
		stampaComponenti(elenco, dim);
		free(elenco);
	}

	// TEST Es. 3
	{
		Componente * elenco;
		int dim;
		list listaElenco;
		printf("\n\n");
		elenco = leggi("disp.txt", &dim);
		stampaComponenti(elenco, dim);
		listaElenco = esauriti(elenco, dim);
		printf("\n\n");
		showlist(listaElenco);
		free(elenco);
		freelist(listaElenco);
	}

	// TEST Es. 4
	{
		Componente * elenco;
		int dim;
		list listaElenco;
		list listaTemp;
		printf("\n\n");
		elenco = leggi("disp.txt", &dim);
		listaElenco = esauriti(elenco, dim);
		listaTemp = listaElenco;
		while (!empty(listaTemp)) {
			if (!costruibile(head(listaTemp), elenco, dim))
				printf("Il componente \"%d %s\" non e' costruibile!\n",
head(listaTemp).codice, head(listaTemp).desc);
			listaTemp = tail(listaTemp);
		}
		printf("\n\n");
		free(elenco);
		freelist(listaElenco);
	}

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