Fondamenti di Informatica T-1
Modulo 2

Obiettivo di questa esercitazione

- Allocazione dinamica della memoria ed esempi di problemi tipici d'esame

Esercizio 1

Una ditta utilizza un sistema informatico per gestire i rimborsi dei viaggi di lavoro dei propri dipendenti. A tal scopo ha definito il concetto di **Spesa**, rappresentante un singolo esborso di denaro sostenuto dal dipendente, e il concetto di **Rimborso**, composto dall'insieme delle singole spese sostenute dal dipendente durante il viaggio di lavoro.

In particolare ogni **Spesa** è caratterizzata da:

- un identificatore unico del viaggio di lavoro (un intero);
- dal tipo di spesa (una stringa di al più 1023 caratteri, senza spazi);
- e dall'importo (un float).

Un **Rimborso** invece è caratterizzato da:

- l'identificatore unico del viaggio di lavoro;
- da un array di singole spese relative al viaggio in questione;
- dalla dimensione di tale array.

Ovviamente non è possibile sapere a priori quante spese sono state sostenute durante un certo viaggio, quindi l'array dovrà essere allocato dinamicamente della opportuna dimensione una volta noto il numero di spese.

Esercizio 1

Il sistema informatizzato funziona nel seguente modo: ogni dipendente accede al programma, specificando la propria matricola (una stringa di al più 7 caratteri numerici, senza spazi). Di seguito inserisce i dati relativi a tutte le spese da lui sostenute (spesso accade che vengano inserite spese relative a più viaggi differenti).

Il programma salva tutte le spese inserite in un file di testo il cui nome è dato dalla matricola del dipendente, a cui viene aggiunto il suffisso ".txt".

In un secondo momento poi il programma leggerà il contenuto di tale file, istanzierà le opportune strutture dati di tipo Rimborso, e per ognuna di queste ne calcolerà il valore di rimborso totale.

Esercizio 1

1. Si definisca un'opportuna struttura dati **Spesa**, al fine di rappresentare i dati relativi ad una singola spesa: in particolare si dovrà tenere traccia dell'identificatore del viaggio, del tipo di spesa e dell'importo.

2. Si definisca la procedura:

   void scriviSpese(char * nomeFile);

   che legga da standard input le spese sostenute da un lavoratore e le scriva su un file di testo. In particolare, la procedura deve innanzitutto chiedere che venga specificata la matricola del lavoratore, e poi deve aprire in scrittura un file col nome opportuno, come specificato in precedenza (matricola più ".txt").

   Quindi la procedura deve chiedere all'utente di inserire i dati relativi ad ogni spesa, e salvarli sul file di testo, ogni Spesa su una singola riga, i vari campi separati da uno spazio.

   Non è noto a priori quante spese saranno introdotte: l'utente segnala l'intenzione di terminare la fase di inserimento specificando come identificatore unico del viaggio il valore "-1". Tramite il parametro **nomeFile**, la procedura deve salvare/restituire al chiamante il nome del file su cui sono state salvate le informazioni.

Esercizio 1

3. Si definisca poi la funzione:

   Spesa * leggiSpese(char * nomeFile, int * dim);

   che, ricevuto in ingresso il nome di un file di testo contenente le spese di un dipendente, e un intero dim passato per riferimento, allochi dinamicamente memoria a sufficienza (la minima necessaria) per contenere le strutture dati di tipo **Spesa** registrate nel file. La funzione deve restituire la dimensione di tale vettore tramite il parametro **dim**.

   La funzione dovrà leggere il contenuto del file e copiare i dati dentro l'apposito vettore allocato: dovrà infine restituire un puntatore all'area di memoria allocata.

Esercizio 1

4. Si definisca un'opportuna struttura dati **Rimborso**, al fine di rappresentare i dati relativi ad una richiesta di rimborso per un viaggio di lavoro: in particolare si dovrà tenere traccia dell'identificatore del viaggio, dell'elenco delle spese sostenute durante tale viaggio, e della dimensione di tale vettore.

5. Si definisca una procedura:

   void ordina(Spesa * v, int dim);

   che, ricevuto in ingresso un vettore v di strutture dati di tipo **Spesa** rappresentante le spese effettuate da un impiegato in un viaggio di lavoro, e la dimensione **dim** di tale vettore, ordini il vettore secondo il seguente criterio: gli elementi devono essere ordinati in modo crescente in base all'identificatore del viaggio; in caso di stesso identificatore, allora in ordine lessicografico rispetto al tipo di spesa; in caso di spesa uguale, in modo crescente in base all'importo.

   Il candidato, per effettuare l'ordinamento, usi l'algoritmo MergeSort visto a lezione. Ovviamente si abbia cura di de-allocare eventuali aree di memoria usate temporaneamente per l'ordinamento.

Esercizio 1

6. Il candidato definisca una funzione:

   Spesa * eliminaDuplicati(Spesa * v, int dim, int * dimNew);

   che ricevuti in ingresso un vettore di strutture dati di tipo Spesa, e la dimensione dim di tale vettore, restituisca un nuovo vettore allocato dinamicamente (non necessariamente della dimensione minima possibile), da cui siano stati eliminati eventuali spese duplicate.
   Una spesa è un duplicato di un'altra se tutti I campi sono identici. Tramite il parametro dimNew la funzione deve restituire la dimensione logica del nuovo vettore restituito come risultato.

7. Il programma infine determini quanti rimborsi sarà necessario creare (uno per ogni viaggio), allochi un vettore di Rimborsi grande a sufficienza, e lo riempa creando le opportune strutture dati rimborso.

   Il programma infine stampi a video il contenuto del vettore dei rimborsi.

Esercizio 1 – Soluzione

"element.h":

```
#ifndef ELEMENTH
#define ELEMENTH

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

#define DIMTIPO 1024
#define DIMFILE 12
#define DIMMATR 8

typedef struct {
    int viaggio;
    char tipo[DIMTIPO];
    float importo;
} Spesa;

...
```

Esercizio 1 – Soluzione

"element.h":

```
...

typedef struct {
    int viaggio;
    Spesa * spese;
    int dim;
} Rimborso;

typedef Rimborso element;

int compare(Spesa s1, Spesa s2);

#endif
```

Esercizio 1 – Soluzione

"element.c":

```
#include "element.h"

int compare(Spesa s1, Spesa s2) {
    int temp;

    temp = s1.viaggio - s2.viaggio;
    if (temp == 0)
       temp = strcmp(s1.tipo, s2.tipo);

    if (temp == 0)
       if (s1.importo-s2.importo<0)
              temp = -1;
       else
              if (s1.importo == s2.importo)
                     temp = 0;
              else
                     temp = 1;
    return temp;
}
```

Esercizio 1 – Soluzione

”rimborsi.h":

```
#ifndef RIMBORSIH
#define RIMBORSIH

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

void scriviSpese(char * nomeFile);

Spesa * leggiSpese(char * nomeFile, int * dim);

void ordina(Spesa * v, int dim);
Spesa * eliminaDuplicati(Spesa * v, int dim, int * dimNew);
Rimborsi * creaRimborsi(Spesa * v, int dim, int * dimRimborsi);

#endif
```

Esercizio 1 – Soluzione

"rimborsi.c":

```
#include "element.h"
#include "rimborsi.h"

void scriviSpese(char * nomeFile) {
     FILE * fp;
     Spesa temp;

     printf("Inserire la matricola: ");
     scanf("%s", nomeFile);
     if (strlen(nomeFile)>0 && strlen(nomeFile)<=7)
         strcat(nomeFile, ".txt");
     else {
         printf("Errore nel trattare la matricola: %s.\n",nomeFile);
         getchar();
         exit(-1);
     }

     fp = fopen(nomeFile, "wt");
     if (fp == NULL) {
         printf("Errore nell'apertura del file: %s.\n", nomeFile);
         getchar();
         exit(-2);
     }
 ...
```

Esercizio 1 – Soluzione

```
    ...

    printf("Inserire la spesa: ");
    scanf("%d%s%f", &(temp.viaggio), temp.tipo, &(temp.importo));
    while (temp.viaggio != -1) {
        fprintf(fp, "%d %s %f\n", temp.viaggio, temp.tipo, temp.importo);
        printf("Inserire la spesa: ");
        scanf("%d%s%f", &(temp.viaggio), temp.tipo, &(temp.importo));
    }

    fclose(fp);

    return;
}
```

Esercizio 1 – Soluzione

```
Spesa * leggiSpese(char * nomeFile, int * dim) {
    FILE * fp;
    int i;
    Spesa temp;
    Spesa * result;

    fp = fopen(nomeFile, "rt");
    if (fp == NULL) {
        printf("Errore nell'apertura in lettura del file %s.\n", nomeFile);
        getchar();
        exit(-3);
    }

    *dim = 0;
    while (fscanf(fp, "%d%s%f", &(temp.viaggio), temp.tipo, &(temp.importo)) == 3)
        *dim = *dim + 1;

    result = (Spesa*) malloc(sizeof(Spesa)* *dim);
    rewind(fp);
    i = 0;
    while (fscanf(fp, "%d%s%f", &(temp.viaggio), temp.tipo, &(temp.importo))==3) {
        result[i] = temp;
        i++;
    }
    fclose(fp);
    return result; }
```

Esercizio 1 – Soluzione

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

    temp = (Spesa*) malloc(sizeof(Spesa) * dim);

    mergeSort(v, 0, dim-1, temp);

    free(temp);
}
```

Esercizio 1 – Soluzione

```
Spesa * eliminaDuplicati(Spesa * v, int dim, int * dimNew) {

    Spesa * result;
    int i, j;
    int trovato;

    result = (Spesa*) malloc(sizeof(Spesa) * dim);
    *dimNew = 0;

    for (i=0; i<dim; i++) {
        trovato = 0;
        for (j=0; j<*dimNew && !trovato; j++)
                if (compare(v[i], result[j])==0)
                        trovato = 1;
        if (!trovato) {
                result[*dimNew] = v[i];
                *dimNew = *dimNew + 1;
        }
    }
    return result;
}
```

Esercizio 1 – Soluzione

```
Rimborso * creaRimborsi(Spesa * v, int dim, int * dimRimborsi) {
    Rimborso temp;
    Rimborso * result;
    int start;
    int end;
    int i, j;

    ordina(v, dim);
    *dimRimborsi = contaViaggi(v, dim);
    result = (Rimborso*) malloc(sizeof(Rimborso) * *dimRimborsi);

    start = 0; end = 0; j=0;
    while (end<dim) {
        while (end<dim && v[end].viaggio == v[start].viaggio)
                end++;
        if (start != end) {
                temp.viaggio = v[start].viaggio;
                temp.spese = (Spesa*) malloc(sizeof(Spesa) * (end-start));
                for (i=start; i<end; i++)
                        temp.spese[i-start] = v[i];
                temp.dim = end-start;
        }
        start = end;
        result[j] = temp; j++;
    }
    return result; }
```

Esercizio 1 – Soluzione

```
int contaViaggi(Spesa * v, int dim) {
    // funziona SOLO SE v e' ordianto in base al viaggio...

    int result;
    int i;

    if (dim>0) {
        result = 1;
        for (i=1; i<dim; i++)
                if (v[i].viaggio != v[i-1].viaggio)
                        result++;
    }
    else
        result = 0;

    return result;
}
```