PROPOSED EXERCISES
Fourth day

Lecturer: Concha Batanero

I propose one application that includes the all concepts we have seen in the course. An explanatory scheme of the dynamic arrays is published together with this document. Please, take into account the next considerations:

- The use of the proposed functions is not mandatory. However, if other functions are chosen, the code must be well organized and programmed in an efficient way.
- Have a look to the scheme before to read the options of the menu.
- The wording must be read more than one time to have a full understanding.
- Global variables are not allowed. Only as an exception, in the case you want to display the elements of the *enum* types, you can define two static arrays of strings as global variables, one array of string for each *enum* type.
- You must use the Memory Manager library and organize the code in several files.

1. Medical application about anatomy. The bones and muscles of the body

The Departments of Anatomy and Information Technology of Metropolia University are going to carry out a project to manage information about of the muscles and bones of the human body. The development is based on two dynamic arrays. One of them stores the data of the muscles and the other stores the data of the bones. The following data types have been designed according to the information received from the Anatomy Department:

```
typedef enum
{
  head, trunk, extremities
}tBodyParts;


typedef enum
{
  estriated, smooth, cardiac
}tMuscleType;


typedef struct
{
  char name[MAX_CHARAC_NAME];          // Ex.: “tibia” or ”trapezius”
  char description[MAX_CHARAC_DESCR]; // Ex.: “It contains several filaments…”
  tBodyParts location;                 // Area of the body where it is located
  int length;                          // Length in millimetres
}tCommonInfo;
```

Each structure of type *tCommonInfo* contains information which is needed for both bones and muscles.

```
typedef struct
{
  tCommonInfo commonInfo;            // Structure of type tCommonInfo
  tMuscleType type;
  char function[MAX_CHARAC_DESCR];  // Describes the function made by the muscle
}tMuscle;
```

Each *tMuscle* structure stores information of a muscle.

```
typedef struct
{
  tCommonInfo commonInfo; // Structure of type tCommonInfo
  float density;          // Degree of bone density. Values between 0 and 1
  int nMusc;              // Number of muscles associated with the bone
  tMuscle **ppMusc;       // Dynamic array of pointers to structures tMuscle
}tBone;
```

Each *tBone* structure stores information of a bone. Every bone has several associated muscles and will have an array of pointers pointing to the corresponding muscles in the muscles array (See schema). The *nMusc* member determinates the size of the dynamic array of pointers which is pointed by *ppMusc*. It is said that the two dynamic arrays (array of bones and array of muscles) are linked. That is, one dynamic array (bones array)

contains pointers which point to the other dynamic array (muscles array).
The header file of the program contains the following declarations:

```
// All include directives that are needed

// Constants

#define MAX_CHARAC_NAME   20
#define MAX_CHARAC_DESCR 100
#define MAX_CHARAC_FUNC  100

// The referred datatype declarations

// Prototypes of the functions
```

Using these data types it is asked to write a program that builds all dynamic arrays by showing the next menu:

1. Add muscle
2. Add bone
3. Print bones
4. Print muscles
5. Print both arrays linked
6. Muscles of an area of the body, which length is less than a value
7. Calculate bone density
8. Exit

1. Add muscle

This option of the menu adds a new muscle to the dynamic array of muscles and reads the data of the muscle from the keyboard. I propose you to use the next functions:

```
tMuscle *AddMuscle(tMuscle *pMuscle, int *pNumMuscles, tBone *pBone, int nBones);
```

This function allocates memory for a new muscle. In case of fail allocating memory, the function frees the memory previously allocated and finishes the program. It receives the pointer to the muscles array and the number its elements for the allocation of the memory. The pointer to the bones array and the number of bones together with the pointer to the muscles array, are needed to free the memory previously allocated. The function returns the pointer to the muscles array.

```
tMuscle ReadMuscle();
```

This function reads from keyboard the data of a muscle. It uses the function:

```
tCommonInfo ReadCommonInfo();
```

to read the members of the structure *tCommonInfo*.

2. Add bone

It carries out the next tasks:

- Add a new bone

  ```
  int AddBone(tBone **ppBone, int n, tMuscle *pMuscle);
  ```

It receives by reference the pointer to the array of bones, the number of bones and the pointer to the muscles array. It returns the number of bones. The function must allocate memory for a new bone. In case of error allocating memory, the function frees the memory previously allocated and finishes the program. The three parameters are required for the freeing of the memory.

- Read the data of the bone which has been added

  ```
  tMuscle ** ReadBone(tBone *pBone);
  ```

  It reads the members of the *tBone* structure from keyboard. Moreover, it allocates memory to the pointer to pointer *ppMusc*. It receives by reference the structure to be read. The function returns, either a pointer to pointer to *tMuscle*, pointing to the allocated array of pointers or *NULL* if the allocation of memory fails.

- Link the pointers of the array of pointer to *tMuscle* of the new bone, with the corresponding muscles in the array of muscles.

  ```
  void LinkBone(tBone *pBone, tMuscle *pMuscle, int nMuscles);
  ```

  It receives the arrays of bones and muscles and the number of muscles. The function makes the next actions: for each of the muscles associates with the bone, it must read from keyboard the name of the muscle and look for this name in the array of muscles. If the name is found, one of the pointers will point to this muscle. If the name is not found, the pointer will point *NULL*.

  In order the link of the pointers, that point to the muscles array, can be done, the user must enter the muscles before the bones. Other possibility can be to introduce a new option in the menu that allows link the two dynamic arrays when the user selects this option.

3. Print bones

This option of the menu displays the information of the array of bones. You can make use of these functions:

```
void PrintBone(tBone bone);
void PrintCommonInfo(tCommonInfo common);
```

4. Print muscles

This option of the menu displays the information of the array of muscles. You can use these functions:

```
void PrintMuscle(tMuscle muscle);
void PrintCommonInfo(tCommonInfo common);
```

5. Print both arrays linked

This option displays the data of every bone including the number of muscles associated to them and their data. It must follow this order:

Bone 1

---
2 muscles associates
Muscle 1

Muscle 2

****
Bone 2

---
1 muscle associate
Muscle 1

****
Bone 3
…….

The functions to print a bone and a muscle can be used to this end.

6. Muscles of an area of the body, which length is less than a value

The purpose of this option is to display the name of all muscles which length is less than a specific value and they are located in a specific area of the body. Moreover, the function must display the name of the bone to which the muscle is associated. Example:

Muscle name	Bone to which is associated
Medium abductor	femur
Tibial	tibia

……

You can use this function:

```
void MusclesLengthArea(tBone *pBone, int nBones, int length, int area);
```

Parameters:

pBone: Pointer to the bones array.

nBones: Number of bones of the array.

length: Maximum length of the muscles that they will be displayed.

area: Part of the body to which the muscle must belong in order to be displayed.

7. Calculate bone density

This option calculates the bone density of a specific area of the body according to the following formula:

$$\left(\sum_{0}^{nbones-1} \text{(degree of bone density / length)} \right) * 100 \; / \; nbones$$

nbones: Number of bones belonging to the specific area.

```
float CalculateBoneDensity(tBone *pBone, int nBones, int area);
```

The function receives the array of bones, referenced by the pointer *pBone*, the number of the bones and the specific area to which they belong bones whose bone density is to be calculated. It returns the bone density.