PROPOSED EXERCISES
Third day

Lecturer: Concha Batanero

The proposed exercises are two applications of medium complexity. The first consists on completing the student cards application, shown on the slides. In the second one the complete application needs to be programmed. The scheme of the dynamic arrays is also showed in the last slide. You must use the Memory Manager library and organize the code in several files.

1. Program for the management of an educational centre

An educational centre wants to have an application to make the management of the students together with their subjects automatic. Develop such application so that executes the next menu:

1. Add student
2. Add subject by ID of the student
3. Calculate the marks average of one student
4. Calculate the marks average of the all students
5. Data display
6. Exit

Take into account the next data types that have been exposed in the slides together with the explanation and solution of some of the options of the menu.

```
typedef struct
{
  char  name[20];
  int   year;
  float mark;
}tSubject;

typedef struct
{
  char  name[20];
  int   ID;
  int   nSubjects;
  tSubject *pSubject;
}tStudent;
```

Use both the information of the menu exercise of the first day and the information about this exercise showed in the slides. You can also run the executable file to analyse every detail of the execution. Think about the functions and their prototypes.

2. Software application for an insurer on reparations appliances

It is asked to develop an application to manage the data of an insurer on reparations of appliances. The customers who buy an appliance have the option to contract insurance for repair. This application will use all the information of customers who buy appliances, along with contracts stored as dynamic arrays of structures. Each customer may insure or not the appliance purchased and each insured appliance will result in a contract. The *tCon* structure stores the information for each appliance purchased and insured:

```
typedef struct
{
      char description[80];
      int  appliancePrice;
      int  insurancePolicy;
      char serialNumber[15];
      char contractNumber[10];
} tCon;
```

The struct *tContract* stores as many *tCon* structures as insurance appliances are by customer.

```
typedef struct
{
      tCon *pContract;
      int  nCurrentContract;
} tContract;
```

tCustomer stores the information about a customer who has purchased one or several appliances although none were insured.

```
typedef struct
{
      char name[50];
      char lastName[50];
      char address[50];
      char code[15]; // Customer personal code
      tContract custContract;
} tCustomer;
```

It was defined in the *main()* function:

```
tCustomer * pCustomer;
int nCustomers;
```

pCustomer implements the dynamic array that contains the information about the customers who have purchased an appliance. *nCustomers* stores the number of customers, that means, the number of elements in the *pCustomer* array.

Using these data types it is asked to write a program that shows the next menu:

1. Add a new customer
2. Add a contract
3. Allow a repair
4. Print customer data using its code
5. Delete a customer using its code
6. Exit

1. Add a new customer

Write a function named *AddCustomer()* that updates the dynamic array of customers adding a new element. This function will use the function *ReadCustomer()* to read the information of the customer from the keyboard: the name, last name and address of the new customer, besides its personal code. Furthermore it has to initialize the pointer of the *tContract* structure to *NULL* and the number of contracts to *0*. The prototype of both functions could be:

```
tCustomer * AddCustomer ( tCustomer * , int * );
tCustomer ReadCustomer();
```

The function *AddCustomer()* returns the address of the array or *NULL* in case the dynamic memory was not allocated. Tip: It is recommended to use the *realloc()* function to add new elements to the array of customers, even though it is empty.

2. Add a contract

Write a function named AddContract() that adds a new contract to a customer. The prototype is:

```
tCon *AddContract(tCustomer *p, int pos);
```

Where *p* is a pointer to the customers array and *pos* is the position of the customer that you are going to add the contract. The function returns the address of the array of contracts or NULL in case the dynamic memory was not allocated.

Consider to create an auxiliary function that read from the keyboard the code of the customer and looks for the location of the customer in the array of customers. Its arguments would be:

1. The dynamic array of customers
2. Number of customers

The value returned by the function would be the accurate location of the customer in the dynamic array, a number in the range [*0* .. *nCustomers-1*]

3. Allow a repair

This function has to check both if the contract exists and if the cost of the reparation does not exceed the 25% of the appliance price. In that case the repair will be allowed, otherwise not. The function must return one of the next values:

0: if the repair is not allowed.
1: if the repair is allowed.
2: if the serial number of the appliance does not exist.

Think of the prototype and implement the function.

4. Print the customer data

Write the function *DisplayCustomer()* that prints on the screen the information about a specific customer. The argument in the function call would be the structure of the customer. Think out the prototype. The auxiliary function which was programmed for case 2 it is also useful for this case to look for the customer.

5. Delete a customer using its code

Write the function *DisplayCustomer()* that deletes a customer with a specific code from the customers array. The function receives the pointer to the beginning of the array, the number of customers and the position of the customer to delete. The program must ensure that the customer exists into the array before to pass his position.

6. Free memory

Write the function *FreeMemory()* that frees the whole dynamic memory used in the application. This function must be called before exiting from the program. The function does not return any value, but receives as an argument the *pCustomer* array.