UNIVERSIDAD CARLOS III DE MADRID
PROGRAMMING. GRADE IN INDUSTRIAL TECHNOLOGY ENGINEERING
CONVOCATORIA EXTRAORDINARIA. JUNE. 2014 PROBLEMS

Surnames:______________________________________
Name:____

Segunda parte. Problemas. Duración 2h 30 min.
Problem 1. (3,5 points out of 10 of the exam)

We want to design a C program to simulate the 2014 Football World Cup that is currently held in Brazil. Let's suppose that a total of 16 national teams participate in the championship. Each one of them is modeled by means of a data structure that contains the following information: an identifier for the team (integer number between 1 and 16), name of the country, leader of a group (integer number, 0=false and 1=true), FIFA coefficient (float number that models the quality of a team), and an array with the data corresponding to the 20 footballers in the team.
In addition, each footballer is modeled by means of a data structure that contains the following information: identifier of the footballer (integer number between 1 and 20), complete name, and number of goals scored during the championship.
It is required

a) Define the data structures required to represent the described information to manage the championship. [**1 point**]

b) Define a function that uses the screen to ask for the data of a national team, including the FIFA coefficient. The number of goals scored by each player must be initialized to 0 and the identifier must be assigned to the formal parameter *numTeam*. In addition, the selections with identifier between 0 and 3 will leaders of a group. The function must have the following prototype [**1.5 points**]:

```
void add_nationalTeam(struct nationalTeam nationalTeams [MAXTEAMS], int numTeam);
```

c) The national teams must be initially divided in NUMGROUPS groups containing NUMTEAMS teams. This will be represented in our program by means of a matrix containing the identifiers of the teams, with size NUMGROUPSxNUMTEAMS. Each cell will contain the identifier of the team, and each column will correspond with a group. It is required to write the code of a function that automatically generates this matrix considering the following requirements [**1.5 points**]:
   1. Each group must only contain a national team that is leader of a group (there are only 4 in the championship).
   2. Each selection can be included in only one group.

   The prototype of the function must be the following:

```
void generate_groups(struct nationalteam nationalteams [MAXTEAMS], int groups
[NUMGROUPS][NUMTEAMS]);
```

d) Design a function to determine the winner of each group after the initial phase. We are going to consider that the winner of a group is directly the selection of the group with the higher FIFA coefficient in the corresponding field. The prototype of the function must be the following [**2 points**]:

```
void winners_initial_phase (struct nationalteam nationalteams [MAXTEAMS], int
groups [NUMGROUPS][NUMTEAMS], int winners_phase1 [NUMGROUPS]);
```

e) Let's suppose that the number of goals scored by each player has been updated at the end of the initial phase of the championship. It is required to design a function to order the players of a national team according to their number of scored goals. The prototype of the function must be the following [**2 points**]:

```
void order_players(struct nationalteam nt, struct player
ordered_player[MAXPLAYERS]);
```

f) Finally, design the *main* function of the program according to the following set of steps [**2 points**]:
   1. Require the user to complete the data of the national teams that participate in the championship.
   2. Make a simulation of the initial phase of the championship (create the matrix of groups and determine the winner of each one of them).
   3. Print the name of the winning national team of each group after the initial phase.
   4. Print the name of the maximum goal scorer of the winning national team of each group after the initial phase.

Problem 2. (3,5 points out of 10 of the exam)

The objective of this exercise is to write a C program for an e-learning application that poses several questions to the user, which are organized in turns of 2 questions and are never repeated. Each user answer is evaluated as either *right* or *wrong*. The program always begins a turn of 2 questions of the Unit 1. When the user correctly answers the two questions of the current turn, then the system moves to the next unit. If he fails a question, the system continues in the same unit. If the current Unit is 2 or higher, the system downgrades to the inferior level when the user fails the two responses.
For example, if the user is in a turn of Unit 2, and fails the two responses, then the next turn will be two questions of Unit 1. If he fails one of the two questions, then the next turn will be again two questions of Unit 2. When the user answers correctly the two questions of a turn of Unit2, then the next turn will consist of 2 questions of the next unit.
The program ends when the two questions of a turn of the last unit have been correctly answered. Then, the program will inform the user about the mark obtained in each unit and it will recommend him to review the units in which the mark is below a given threshold. After each turn of questions, the program will ask the user if they want to force quit the program. In this case, the program does not communicate the mark nor the recommendations.
The mark for a unit is calculated using the formula:

10 x number of correctly answered questions / total of questions asked by the program

The program calls the function *loadQuestions*, which completes the information related to the questions. The prototype of this function is:

void loadQuestions (struct questionCard questions[]);

We will assume that this function is already implemented, so you just have to include it using the directive *<include questions.h>*

Questions:

1. Define the data structures: **(1 point/10)**
   a. Question structure. Contains a question, with three fields: number of the unit, the question belongs to text of the question, and answer (1: true or 0: false).
   b. Vector containing all the possible questions
   c. Results structure. Results of a user in a unit, with three fields: total number of questions asked from that unit, number of questions correctly answered, and mark.
   d. vector containing the results obtained by the user at each unit (the program is designed assuming there is only one user).
2. Write the function *selectQuestion*. The formal parameters are the number of the unit, the vector with the questions, and the vector with the used questions. This function will look for the first question fromt that unit that has not been previously used and will return its position in the vector containing the questions. **(2 points/10)**
3. Write the function *askUnit*. The formal parameters are the number of the unit, a vector with the questions, a vector with the used questions, and a vector containing the results. The function *askUnit* must call the function *selectQuestion* to select the questions, ask them to the user, and update the number of correct answers in a specific unit. **(3 points/10)**
4. Write the *main* function of the program. It will contain the main loop to select the unit to ask, and the calls to the functions *loadQuestions, askUnit, mark* and *recommend.* **(3 points/10)**
5. Write the functions *mark* and *recommend.* The function *mark* will calculate and print the mark for each unit. The function *recommend* will print the list of units that must be reviewed by the user. **(1 point/10).**

Note: the maximum number of questions and units, and the threshold (minimum mark under which a unit must be reviewed) must be defined as constants in the program. You can assign to them values of your choice.