Programming – Evaluation Exercise May 2013	Universidad Carlos III de Madrid
Grado en Tecnologías Industriales	

READ CAREFULLY THESE INSTRUCTIONS BEFORE STARTING THE EXAM:

- Fill in all the pages with a pen (personal data and answers)
- Do not use a pencil or a red pen
- Do not forget your NIA and your actual group
- The duration of this exam is 2 hour and 30 minutes
- Use only the provided sheets for the answers. Additional sheets will not be considered
- Notes, books, etc. are NOT allowed

Surname	*Name*
Signature	*NIA*

Problem 1

Bingo is a game of chance played with randomly drawn numbers which players match against numbers that have been pre-printed on card stocks. The first player to achieve all the numbers printed on his/her card from the drawn numbers is the winner, and calls out the word 'bingo' to alert the other players.

Develop a C program to play a simplified version of the *Bingo* game according to the following specification:

The game will be played by 100 players at most. Player information must be stored in an array of structures, each one including:

- Name of the player (string)
- Age of the player (integer)
- Numbers in his/her card (see below)

Player's cards must be represented by a 3x9 grid with numbers in {1, 2, …, 90}. Each player plays only one card. The card will be modified during the game as numbers are matched and *crossed out.*

21	51	89	82	4	1	68	29	1
19	10	35	43	68	8	2	38	90
43	6	22	30	59	57	26	1	85

Before starting the game, the program must ask the user for the number of players, which must be between 1 and 100. Afterwards, the program reads from the keyboard the name and the age of each user, and initializes the grid that represents the board by using the pre-existing function below. (Assume that this function has been already implemented elsewhere and is available to your program.)

```
/* Function: Initialize user game card with random numbers in {1, 2, …, 90}
 * Parameters:
 * > card: Matrix representing a card grid (modified!)
 * Returning value:
 * < None */
void initializeCard(int card[ROWS][COLS])
```

After players' data are initialized, the game starts. The program must read integer values from the keyboard that simulate the drawn numbers. These numbers must be in the interval {1, 2, …, 90}. For the sake of simplicity, it is not required to check if read values have been previously drawn (i.e., repetitions are allowed).

Once a number is called, the program must cross out all the players' cards with this number. If the number is found in a card, it is crossed out by setting the corresponding matrix value to 0.

The 'call number-check cards' process is repeated until any player achieves all the numbers on his/her board. At this point, the game ends and this player is the winner. Notice that more than one player might achieve all the numbers in the same turn.

After finishing the game, the program must print on the screen a ranking with the results obtained by each player (name + how many numbers left to win).

The program **MUST DEFINE AND USE FUNCTIONS.** Follow the steps below:

1. (0.5p) Define and declare a suitable data structure to the store players' data.
2. (0.5p) Implement a function named `initializePlayersData`.

   This function has two parameters: an array storing players' data and an integer representing the number of players of the current game. It must initialize each player's data. Player's name and age is read from the keyboard. The card is initialized by using the function `initializeCard`.
3. (1p) Implement a function named `crossOutNumber`.

   This function has two parameters: a matrix representing a card and a drawn number. It must check if the number is in the card and cross out any coincident element (i.e., set them to 0).
4. (1p) Implement a function named `testRow`.

   This function has two parameters: a matrix representing a card and a row number. It must check if all the numbers of the specified row of the card have been crossed out (i.e., they are 0). If so, it returns 1; otherwise, it returns 0.
5. (1p) Implement a function named `testBingo`.

   This function has one parameter: a matrix representing a card. It must check if all the numbers of the card have been crossed out (i.e., they are 0). If so, it returns 1; otherwise, it returns 0. It MUST USE the `testRow` function.
6. (1p) Implement a function named `getNumbersToBingo`.

This function has one parameter: a matrix representing a card. It must calculate and return how many numbers remain uncrossed in this card.

7. (2p) Implement a function named `showRanking`.

   This function has two parameters: an array storing players' data and an integer representing the number of players of the current game. It must print on the screen the name of the players and the number of uncrossed numbers by each one in increasing order of uncrossed numbers (i.e., first the ones who were closer to win the game). In case of a tie, players can be ranked in any order.

8. (3p) Write the main function of the program to implement the game loop by using the previous functions. You are not required to write again the declarations and definitions performed in 1.

Problem 2

The City of Macondo has four water tanks. Each one provides water supply to one district of the city. The tanks are equipped with devices to fill up the tank, drain the tank, transfer water from one tank to another, and measure the volume of water stored in the tank. All the tanks have the same storage capacity. A tank can be empty, half-filled or full, which is represented with the values {0, 1, 2} respectively. The water demand of each district is a value in the range {0, 1, 2}, corresponding to {no demand, normal demand, high demand} respectively.

The effectiveness of the water supply procedures is measured with a numerical value named *water stress*. The water stress is calculated as the difference between the amount of water stored in a tank and the water demand of the district that is served by the tank. Notice that positive and zero water stress values mean that the stored water is enough to satisfy the district requirements, whereas negative values entail a shortage in water supply. For example, a district with normal demand (1) and a full tank (2) has a water stress equal to 2-1=1 (good supply). A district with high demand (2) and a half-filled tank (1) has a water stress equal to 1-2 = -1 (shortage).

The operators can perform 4 different actions on the water tanks:

a. Fill up a tank
b. Drain ½ of a tank
c. Empty a tank
d. Transfer water from one tank to other tank

These actions can be performed under the following conditions:

a. A tank cannot be filled up or receive a transfer if it is full
b. A tank cannot be emptied, drained or send a transfer if its empty
c. A source tank always transfers as much water as the destination tank can hold. For example:

	Source tank	**Destination tank**
Before	Half-filled	Empty
After	Empty	Half-filled

	Source tank	**Destination tank**
Before	Full	Half-filled
After	Half-filled	Full

Develop a C program to manage the water tanks according to the following specification:

1. Declare and initialize a suitable data structure to store information about water tanks and districts. The structure must store:
   - Name of the tank (string)
   - Current amount of water stored (integer)
   - Name of the district served by the tank (string)
   - Demand of the district (integer)
   - Water stress of the district (integer)

   See below the initial values for the tank names, stored water and district names.

2. Display a user menu and implement the following user options:
   1. Inspect the amount of water stored by each tank
   2. Read demand values of all districts
   3. Fill up a tank
   4. Drain ½ of a tank
   5. Empty a tank
   6. Transfer water from one tank to another
   7. Display recommended actions
   8. Exit

   The program will execute the action selected by the user. Notice that after running options 2 to 6, stress values must be updated. Afterwards, it will present again the options menu. The program finishes when the users selects exit or all the districts have a good supply (water stress is >= 0). Before ending, it must print if it ended because the operator selected exit or all the districts have a good supply.

Additional information

The *inspect* option must present information as follows (use these values to initialize the tanks):

```
Tank Alpha (North District): 2
Tank Bravo (South District): 0
Tank Charlie (East District): 1
Tank Delta (West District):  1
```

The *read demand* option must read the demand values of each district from the keyboard and calculates the water stress.

The *recommended actions* option must show a list of the tanks that serve districts currently affected by a supply shortage (i.e., those with negative water stress value). Assume that *recommended actions* will be selected only after running *read demand*.

Draining ½ reduces tank storage from full to half-filled, and from half-filled to empty.

If the user selects a non-authorized action (e.g., fill up a full tank), a proper information message must be displayed. After the message, the program must show the user menu again.

Tank numbers can be used to select the water tank(s) involved in an action.

FUNCTIONS ARE NOT REQUIRED in this exercise.