UNIVERSIDAD
COMPLUTENSE
MADRID

Fundamentals of Programming II

Help for the Practice (Part II)

Index

1. Introduction

- ✓ This presentation is to help on the development of the part II of the miner´s practice.
- ✓ Needed: some practical work on **pointers** and **dynamic arrays**.
- ✓ ¡Practice with exercises of lesson 4 if you have not done it yet!
- ✓ This help is optional. You do not need to follow this help in order to solve the part II of the practice.

2. Planning

✓ The planning for the laboratory sessions is as follows:

- 29/4: Scores Module (start)
- 6/5: Scores Module (end)
- 13/5: Main Program and Testing

3. Project and Modules

- ✓ Create a **new Project** for the part II of the practice in Visual Studio.
- ✓ Incorporate the **header files** (.h files) and **implementation files** (.cpp files) of the part I of the practice in the solution explorer. You do this adding existing items.
- ✓ Start working on part II by adding the following C++ files into the corresponding directories: **scores.h** and **scores.cpp**.
- ✓ In *scores.cpp*, use the *#include* directive for including *scores.h*. This module does not use any other module.

4. Implementation of the Scores Module

- ✓ You must include in the *scores.h* file the declarations of constants and types of the scores module, as already introduced in the assignment:

```
const int NUM_TOTAL_PLAYERS = 25;
const int NUM_TOTAL_MINES = 5;

typedef struct {
    int mineId;
    int numMovements = 0;
    int numGems;
    int numDynamites;
    int minePtos;
} tMineData;
```

```
typedef struct{
    string name = "";
    int totalScore;
    int minesTraversed;
    tMineData vTraversedMines[NUM_TOTAL_MINES];
} tPlayerScore;

typedef struct{
   int capacity = 2;
   int numPlayers = 0;
   tPlayerScore *rankingArray;
} tScores;
```

- ✓ *rankingArray* is the **dynamic array** used in the practice for storing the players scores information in the heap.

FP2: Help for the Practice (Part II)

- The prototypes of the subprograms requested in the assignment for the scores module are divided into two main groups. The first group is for loading, saving and displaying the information:
  - `bool loadRanking(tScores& ranking)`: introduces into the dynamic array all the data that is available within the file Scores.txt.
  - `bool saveRanking(tScores& ranking)`: saves the contents of the dynamic array into the Scores.txt file.
  - `void showUserMines(const tScores& ranking, int cont)`: displays the mines traversed by a user, sorted by level (Figure 2).
  - `void showAlphabetical (const tScores& ranking)`: displays the users and their total scores, sorted by alphabetical order (Figure 3).
  - `void showUsersData(const tScores& ranking)`: displays all the data of all the users (Figure 4). The users are sorted alphabetically and, the identifiers of the mines, as they are numbers, they are increasingly ordered.

- The second group is for the management of the dynamic array:
  - `void initializeRanking(tScores& ranking)`: initializes the dynamic array.
  - `void increaseCapacity(tScores& ranking)`: duplicates the size of the dynamic array.
  - `void destroy(tScores& ranking)`: frees the dynamic memory.
  - `bool find(const string& name, const tScores& ranking, int& pos)`: searches a name in the dynamic array and returns if it is found or not. If the name is within the array, it returns the position where it is found, if it is not within the array, it returns the position where it should be. The data within the dynamic array are sorted in alphabetical order. You must implement an iterative binary search.
  - `void insert(tScores& ranking, string const& name, int pos)`: inserts in a sorted way a new player (name) in the position pos. When inserting the new player, if there is not enough space in the array this is extended.

- ✓ You do not need implementing more functions for your solution. And you do not need making any modification in the implementation of part I excepts some new arrangements in the main program, as we will see at the end of this help material.
- ✓ Start implementing the dynamic array management functions in the score module.
- ✓ For the *initializeRanking* and *increaseCapacity* procedures, it is important that you understand the exercise number 12 of lesson 4 that we have seen in the class.
- ✓ With the *initializeRanking* procedure you have to initialize the list of scores putting a 0 in the number of players, a 2 in the capacity and, creating the dynamic array of size 2.

- ✓ For implementing the *increaseCapacity* procedure, consider the following pseudocode:

```
Create a dynamic array with size equal to 2 * actual capacity of the
dynamic array
Use a for loop for copying all the contents of the current dynamic
array into the new one
Delete the current dynamic array
Make current dynamic array equal to the new one
Multiply the capacity of the list by 2
```

- ✓ Implement the *destroy* procedure deleting the dynamic array, putting a 0 in the number of players and 0 in the capacity.

- ✓ You have to implement the *find* function doing a **binary search** in the list. Remember that if the *name* is not found in the list, the position that you have to return is *ini*.
- ✓ Implement the *insert* procedure using the following pseudocode:

```
If the number of players is equal to the capacity of the dynamic
array then increase the capacity of the dynamic array
Shift all the elements of the dynamic array one position to the
right, from position pos
Put the name, totalScore equal to 0 and, minesTraversed equal to 0,
in the position pos of the dynamic array
Put a 0 in the numMovements of all the mines of the position pos of
the dynamic array
```

- ✓ Now you have to implement the rest of the functions of the scores modules for loading, saving and displaying the ranking information.
- ✓ Start with the *loadRanking* function. Use “000” as a centinel for loading the information. The first thing you have to do in the loading loop is checking if you have to increase the capacity of the dynamic array. Further, you have to use an internal while loop for loading the information of the traversed mines of each player.
- ✓ Implement the *saveRanking* function saving the information of the scores list into the file *Scores.txt*. Here you also need two nested loops. The external loop is for the players and, the internal loop is for the traversed mines of each player. Do not forget writing the centinel at the end of the file.

- ✓ With the *showUserMines* procedure, you have to display all the information about the traversed mines of a player in position *cont* of the *ranking*. First, you have to display the names of the columns. Then you need a for loop for displaying all the traversed mines. Look at Figure 2 of the assignment for seeing the format of this display.
- ✓ The *showAlphabetical* procedure displays the names of the players of the ranking with a for loop. For each player, you have to show the name and the total score.
- ✓ The *showUsersData* procedure displays all the information of all the players calling the *showUserMines* within a for loop.

5. Main Program and Final Testing

- ✓ In the main program, you only need to modify the *main* function and remove the menu for asking if the user wants to play the next level.
- ✓ For the **main function**, consider the following pseudocode:

```
Incorporate a variable for the ranking (list of scores) into your main
function
Declare variables for option, game, etc.
Initialize the ranking
Load the ranking and show an error message in case you cannot load it
Display the head of the part II of the practice
Ask for the name of the player
Look for the player in your ranking
If the player is already registered, say "you are registered" and show
the traversed mines of the player.
Else, say "you are new" and show the players in alphabetical order.
```

```
Repeat while option is not equal to 0
    Ask for the number of the mine that the user wants to explore or, exit
    If the user wants to exit, option is equal to 0
    Else
        Put the number of the mine in the level of the game
        If it is not a new mine for the player, update the total score
        Show the menu for playing at different scales and read the option
        If the option is equal to 1 or 2
            Load the game
            Put the selected scale into the game
            Show the next menu and read the option
            If the option is equal to 1, call play_from_keyboard
            Else, call play_from_file
            If the player was successful with the mine
                Increase the number of traversed mines of the player
                Update the dynamic array with all the information
                Display all the players with their traversed mines
                Reset the game
            Else, option is equal to 0
Save the ranking into the file Scores.txt
```

- ✓ Do not forget to *destroy* the dynamic array at the end of the program so all the dynamic data is returned to the heap.
- ✓ Once you have made all the modifications into your main program, you can test your code.
- ✓ Use the file *Scores.txt* that we provide you with the practice.
- ✓ Test your code with already registered players (Player 1 and Player 3) and, test your code with new players (for example, Player 2).
- ✓ You must be able to play new mines and store all the data in the dynamic array.