PCS4 Assignments – Week 4

Version: 2016 May



Table of Contents

[\* Assignment 4.1: simple out parameter exercise 2](#_Toc451524231)

[Assignment 4.2: simple ref parameter exercise 2](#_Toc451524232)

[\* Assignment 4.3: Predict the output 2](#_Toc451524233)

[\* Assignment 4.4: Recursive binary search: again the deliverables 2](#_Toc451524234)

[Assignment 4.5: Count recursive calls in your binary search algorithm 3](#_Toc451524235)

[\*Assignment 4.6: Songfestival 4](#_Toc451524236)

[Optional Assignment 4.7: Implement the game “Master Mind” 5](#_Toc451524237)

**NOTE**: The assignments marked with an asterisk are the most important ones.   
 The others serve as extra practice.

## \* Assignment 4.1: simple out parameter exercise



Write a void method CalculateProduct that takes two (int) value parameters and one (int) out parameter. After performing this method the out-parameter should hold the product of the two value parameters.



The heading of this method is:

private void CalculateProduct(int a, int b, out int product)

Write a program to test its working.

## \* Assignment 4.2: simple ref parameter exercise



Write a method with two ref parameters, say a and b. After performing this method the values of a and b are swapped.

## \* Assignment 4.3: Predict the output



Try to predict and explain the output of the next program:

class Program  
{  
 private int Puzzle (int a, ref int b)

{ int c = a + b;

a = 5;

b = a \* 3;

return c;

}

static void Main(string[] args)

{ int a = 1, b = 2;

int c = Puzzle (a, ref b);

Console.WriteLine(c); 🡺 3

Console.WriteLine(a); 🡺 1

Console.WriteLine(b); 🡺 15

Console.ReadLine();

}

}

## \* Assignment 4.4: Recursive binary search: again the deliverables



During the theory-lesson we did a binary search on an array of integers. More interesting is to apply the binary search algorithm in a more real situation instead of just using numbers. Let us consider again the application written in the previous weeks (about the Deliverables and Persons).

In this assignment you will practice with several aspects.

* We ask you to extend your application with search-functionality. For instance, the user would like to have information about a certain deliverable, or the user want information about all deliverables for a certain buyer, or the user want to see all deliverables that need to be delivered at a particular address.
* You need to implement the binary search in a recursive way.



**Part 1**.

Continue with the project of week 2.

Add some controls to the Form-class to be able to search for a deliverable with a particular id. A solution could be to add a textbox and a button. The user can type a potential id in the textbox. Clicking the button should find the deliverable with that particular id. If such a deliverable exists, you should show information about it. If it does not exist show an appropriate message to the user.

Wow, that's easy: the TransportingCompany-class already has a method

public Deliverable FindDeliverable(int id)

So you just call this method for the searching-job.

This method is implemented in an iterative way. Suppose the list is sorted by Id. Then this implementation is not efficient. Now you must implement it much more efficient by using binary search. Furthermore you should implement this method by using recursion.

As you know, after performing some steps in a binary search you need to be able to allocate a range (a lowindex and a highindex) in which the searching should take place.

Now add a private method

private Deliverable BinarySearchById(int id, int lowindex, int highindex)

and write a recursive implementation for this method. If the list contains a Deliverable-object within the range "lowindex . . highindex" with an identitiy-number equal to id, this method should return that Deliverable-object; otherwise it must return null.

An implementation for the public method FindDeliverable() is very easy now: just call the method BinarySearchById().

Test if it works.

Be aware: The list of Deliverable-objects must be sorted, otherwise binary search does not work ! ! !



**Part 2**.

Now we want to find a Deliverable-object that need to be delivered at a particular buyer.

Repeat part 1, but now for searching by buyer's name instead of by id. The precondition for this method is that the list is sorted by buyer's name.



**Part 3** (optional, difficult).

In Part 2 you found one of the deliverables for a particular buyer. But there could be many deliverables for a particular buyer.

Implement a method

public List<Deliverable> FindAllDeliverables(String buyersName)

that returns a list of all deliverables for that particular person.

Hint: use a binary search to find one of these deliverables. Then find all other deliverables for that person .

## Assignment 4.5: Count recursive calls in your binary search algorithm



In a previous assignment 3.4 you wrote a recursive binary search method. This method returned a Deliverable-object for a certain search-value. Now we are curious about the number of recursive calls that were needed to find that Deliverable-object. So you need an extra counter (int) to keep track of how many times the method is called until you found the search-value (or until you know the value is not there). At the end of the recursion this value must be returned to the original caller that started the recursion (the button-click-handler) . But because there is already a return value, this count must be returned by an output-parameter.

Extend your method with an out- or ref-parameter count to pass the number of calls needed to find the search-value to the caller.

Test it and show the value of that counter at the end of the search.

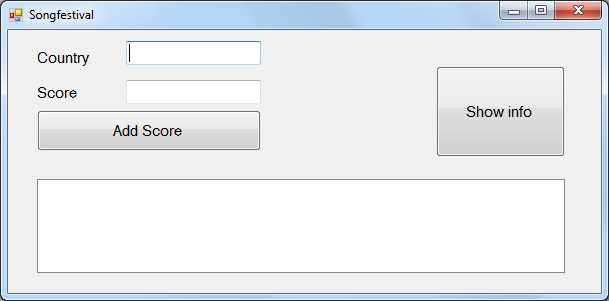
## \*Assignment 4.6: Songfestival



An important part of the songfestival is the voting: numbers are given by the jury and the audience. You are asked to implement an application by means of which these numbers can be entered for each of the contestants.

In the startup-project we already added some contestants and some votes in it, but you are free to change it. Part of the application is already implemented.

In the text boxes at the top left corner, one can fill in a country name, and a score. When pressing the button "Add score", the score will be added to the list of scores for the indicated country.



When the "Show info" button is pressed, the following information should appear:

* For the contestant, indicated by the country-name in the upper-left-textbox, we want to have the total score (the sum of all scores for that country), and
* We want to have the average total score of all countries together (to be able to compare this country's score with the average).

Add a method to the Songfestival-class to determine the required information. Since this method should determine 2 values, we do it by using out parameters. The heading for this method in the Songfestival-class is:

public void CalculateInfo(String countryName, out int scoreCountry, out double averageScoreAllCountries)

Implement this method and test if it works.

## Optional Assignment 4.7: Implement the game “Master Mind”

This week we are going to implement a real game, that can be played against the computer. It is the game “master mind” and for those who have never heard of it, here are the rules of the game:

The computer starts with the selection of a secret combination of 4 numbers (let us assume, those numbers must range between the 1 and 9 and a number can never be used more than once in that combination ):

?

?

?

?

The goal of the game is to find that combination in the fewest number of guesses (but if you haven’t found the combination in 10 guesses you lose the game).  
In every guess you mention a sequence of 4 numbers and the computer reveal certain information about how close the guess was to the actual hidden sequence.

The information you get from the computer after each guess consists of 2 values:  
- a first value (score1) indicates how many numbers in you guess were on the right place   
- a second value (score2) indicates how many numbers were correct, but on the wrong place.

So, for instance, if our guess is 1,2,3,4 and the computer has the secret combination 1,9,2,3, than there is one number correct (the 1) and two numbers correct but on the wrong place (the 2 and the 3). So the scores given by the computer will be 1 resp. 2.

Another guess (with the same secret combination): 4,2,3,6. Now the response will be 2,1 (2 because there are 2 numbers at the correct place, and 1 because there is one correct value but on the wrong place).

Of course, this information will help you to guide your next guess.

If your guess scores the values 4,0 it means you have found the secret combination and if this is done within 10 tries, you win the game.

Now we are going to implement all this.  
You can start with the form design given in the project available on sharepoint.



STEP 1:  
First we will need a new class SecretCombi, with an array of 4 integers as a private field. (No properties are necessary for this array because it must remain secret for others.) It also needs a Random object to fill the secret combination randomly.   
In the constructor of this class the secret combination must be created. And make sure that each of the 4 numbers must be different from each other! (Come up with something to accomplish this.)

And finally we need a method can be used every time we enter a new guess. This method must calculate the 2 scores based on your guess and must publish these 2 scores as output parameters. So this method has several parameters:  
- your new guess (an array of 4 integers), an input parameter  
- score1 (an integer), an output parameter   
- score2 (an integer), an output parameter  
If the two scores are 4,0 (which means that the hidden combination is found) the method should return true, in all other situations the value false should be returned.



STEP 2:

Now you must add code to the form. First create a SecretCombi object. And next, implement the button “check my guess”. (The 4 numbers of each new guess must be entered in the 4 textboxes, before you press that button.) In this button-click the method written in step1 must be invoked to get information about this guess. After this call the information received must be shown on the listbox. Don’t clear the information of previous guesses, but append a new line every time, so you will be able to get an overview of all the information received until now, which is important because it might help to find the secret combination as soon as possible. For every next guess add one new line, containing both the guessed combination itself and the 2 scores received by the computer based on that guess.  
(See the screen shot on the picture below.)



If you have found the secret combination (the method invoked in the button-click returns a true), you win and the game is over. You can show a messageBox or something else to make clear that you have found the secret numbers and won the game.

Test the code written so far.



STEP 3:

The program is not ready yet, there are a few extensions that have to be added:

* There was the rule that not more than 10 guesses were allowed. So add some code to keep track of how many guesses have been done. And once this is equal to 10 (and the secret combination is not found yet) another message must be displayed to make clear that the player lost the game. (Is it possible to display the secret combination also at that moment? Add some code to do so.)



* To make it possible to compete against other players, we add another feature to the game: credits. At the moment you start the game you get 100 credits. Every guess decrements your credits in some way (depending on how close the guess was to the actual hidden combination). And now the goal of the game is not only to find the secret combination within 10 guesses but also to keep as many credits as possible at the end of the game.

Add the code in your application to implement this.   
One of the things that must be extended is the method written in Step 1 (in the class SecretCombi). This method needs one extra parameter (credits) . This parameter has already a value at the start of the method-call and must be decremented at the end of the method-call. So this must be a call-by-reference-parameter.

Rules about how many credits will be lost in each guess are:

• if the score is (4,0) (that means you found the code), no credits will be subtracted, otherwise:   
• if the two scores generated by the computer (based on your guess) have a total   
 of 3 or more, only 5 credits are subtracted.   
• if their sum is 1 or 2, 10 credits are subtracted

• if both are 0, 15 is subtracted

• and if the score is (4,0) (that means you found the code), nothing is subtracted   
 anymore.

And of course you must display the remaining number of credits in one way or another at the end of the game.

