Middle East Technical University Department of Computer Engineering

CENG 213

Data Structures

Fall 2018-2019

Programming Assignment 2

Due date: 10 December 2018, Monday, 23:55

1 Objectives

In this assignment you are expected to implement a data structure that will be called *Two-Phase Binary Search Tree (TPBST)*, in which each node of the main BST keeps a pointer to its own uniquely associated secondary BST. The data is contained in the nodes of that secondary BST. The details of the structure is explained further in the following sections.

You will use this specialized two-phase binary search tree structure as the in-memory database of a stock photo webstore, where copyrighted photographs are sold.

Keywords: *Binary Search Tree, Stock Photo*

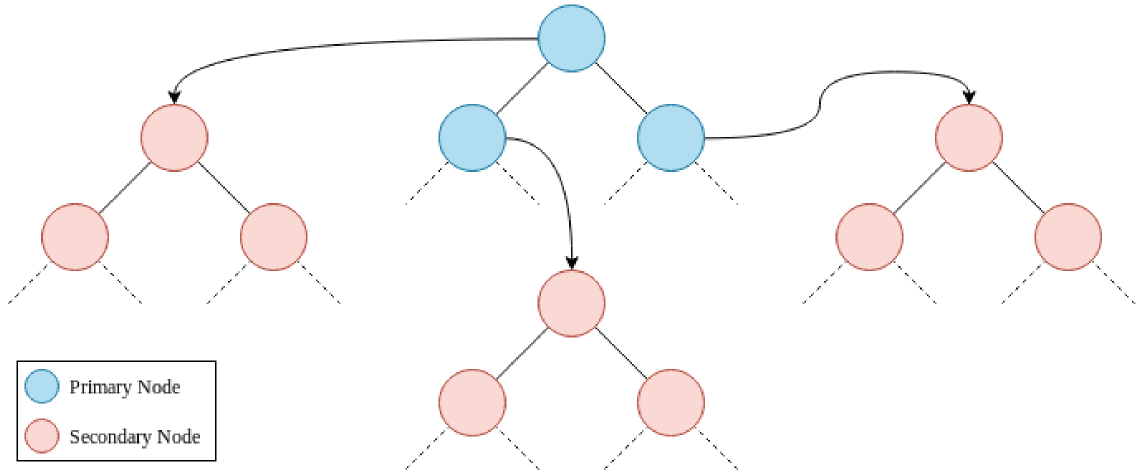

Figure 1: Two-Phase Binary Search Tree structure

2 Two-Phase Binary Search Tree (80 pts)

In Figure 1, the tree consisting of the blue nodes (called primary nodes) is the main tree of the TPBST structure. Each of those primary nodes includes a key, two pointers to its left and right primary node children, and also a pointer to its auxiliary secondary tree which consists of secondary nodes. A secondary node includes a key, two pointers to its left and right secondary node children, and also the data.

To illustrate the structure, one can think of the primary nodes in the main tree as "categorization" nodes where any data that belong to this category are added into that node's secondary BST structure as secondary nodes, where they are ordered using a unique key - called the secondary key. Primary nodes of the main tree are also ordered among themselves based on another unique key - called the primary key. A secondary tree will not include nodes - say *Node A* and *Node B* - with the same secondary keys; however, another secondary tree that belongs to another primary node (category) can include an element that has the same secondary key with *Node A* and *Node B*.

In our Stock Photo Webstore scenario, primary nodes in the main tree will have the categories of the inserted photos as keys to apply lexicographical BST ordering among category names; and secondary nodes will use photo names, that are unique in that category, as keys to apply lexicographical BST ordering among photos in that secondary tree. Photo objects will be kept by secondary nodes in their data field. This overall approach will constitute the database structure of our webstore. By using this approach, we will be able to access and print the details of each photo individually; as well as printing all information currently contained in our TPBST. Moreover, the structure will enable us to access and print the details of all photos that belong to the category that we are interested in.

The TPBST structure used in this assignment is implemented as the class template `TwoPhaseBST` with the template argument `T`, which is used as the type of the data stored in the secondary nodes.

Outline of `TwoPhaseBST` class template implemented in *tpbst.hpp* file is summarized in the following code block:

```
template <class T>
class TwoPhaseBST {
private:
    struct SecondaryNode
    {
        /*...*/
        T data;
    };
    struct PrimaryNode
    {
        /*...*/
        SecondaryNode *rootSecondaryNode;
    };
public:
    /* public TwoPhaseBST functions */
private:
    /* private utility functions */
private: //data
    PrimaryNode *root;
};
```

The basic building blocks of the `TwoPhaseBST` class are `struct PrimaryNode` and `struct SecondaryNode` node structures that are defined in the `private` section, resulting in making all of their data and functions being publicly available to `TwoPhaseBST`, yet totally hiding its existence from the outside world. `TwoPhaseBST` class has a `PrimaryNode` pointer, which points to the root primary node of the main tree, in its private data field.

2.1 PrimaryNode

`PrimaryNode` struct represents nodes that constitute the main tree. A `PrimaryNode` keeps a *key* variable which is an instance of `std::string` to `uniquely` identify the node in the main tree, two pointers to its left and right `PrimaryNode` children, and a pointer to the root of its own secondary BST, which consists of `SecondaryNode`s. The struct also has a constructor that is already implemented in the *tpbst.hpp* file. Do not change the implementation of this constructor in the file.

2.2 SecondaryNode

`SecondaryNode` struct represents nodes that constitute secondary trees. A `SecondaryNode` keeps a *key* variable which is an instance of `std::string` to `uniquely` identify the node in its tree, two pointers to its left and right `SecondaryNode` children, and the data variable of type `T` to hold the data. The struct also has a constructor that is already implemented in the *tpbst.hpp* file. Do not change the implementation of this constructor in the file.

2.3 TwoPhaseBST

Previously, data members of `TwoPhaseBST` class template have been briefly described. Their use will be elaborated in the context of utility functions discussed in the following subsections.

Constructor and destructor functions that reside within the `public` section of the class have already been **implemented**. You must **not** modify these implementations. You may want to pay specific attention to the implementation of the destructor function that utilize **recursive** private utility functions that usually begin their computation at the designated `root` position. This programming style may be helpful in your coding of the uncompleted functions. Alternatively, you may devise iterative solutions. You **can** add other variables and functions to the `private` part of the class.

Copy constructor and assignment operator signatures are also included in the `private` section. We will not allow their use in this assignment and in order to block compiler defaults, prototypes are provided and you should perform **no** implementation. This is a trick utilized in C++ world so as to make objects of the class type uncopiable or unassignable.

You must provide implementations for the following `public` interface methods that have been declared under indicated portions of *tpbst.hpp* file. Those methods are illustrated in the following sample figures using the Stock Photo Webstore scenario.

2.3.1 `TwoPhaseBST & insert(const std::string & primaryKey, const std::string & secondaryKey, const T & data);`

- If `TwoPhaseBST` is empty, insert a `PrimaryNode` created by using the given `primaryKey` to the empty tree, then create a `SecondaryNode` by using the given `secondaryKey`, and make the rootSecondaryNode pointer of the newly created PrimaryNode point to that SecondaryNode. Store the **data** in the SecondaryNode.

- If TwoPhaseBST is **not** empty, search for the node - say *Node A* - with the key value primaryKey among the PrimaryNodes in the main tree.
  - If found, create a SecondaryNode with the key value secondaryKey, store the **data** in it, and insert it to the appropriate location of lexicographical order in the secondary tree of *Node A*. Secondary keys are guaranteed to be unique in the secondary tree of a primary node; so, no need to check for duplications.
  - If **not** found, then create a new PrimaryNode (*Node A*) with the given primaryKey and insert it to the main tree using lexicographical ordering. Then, create a SecondaryNode with the key value secondaryKey, store the data in it, and make rootSecondaryNode pointer of *Node A* point to that SecondaryNode.

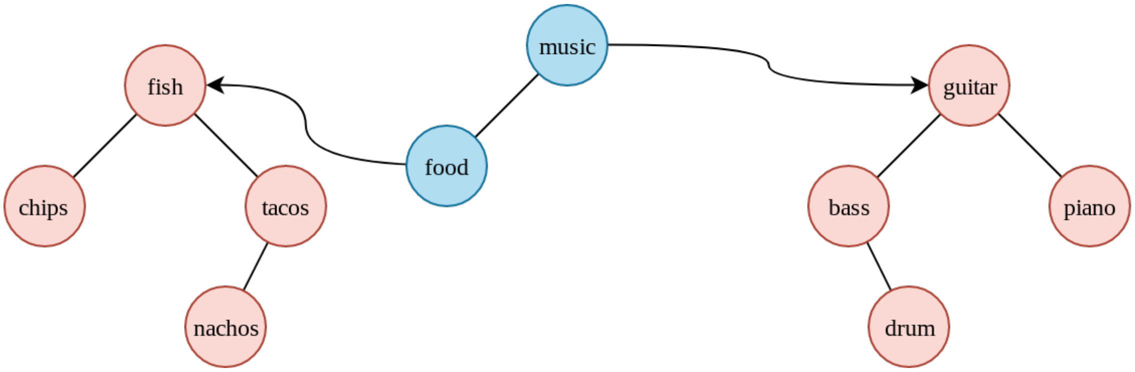

Figure 2: Before the insertion of the photo, named ”harp”

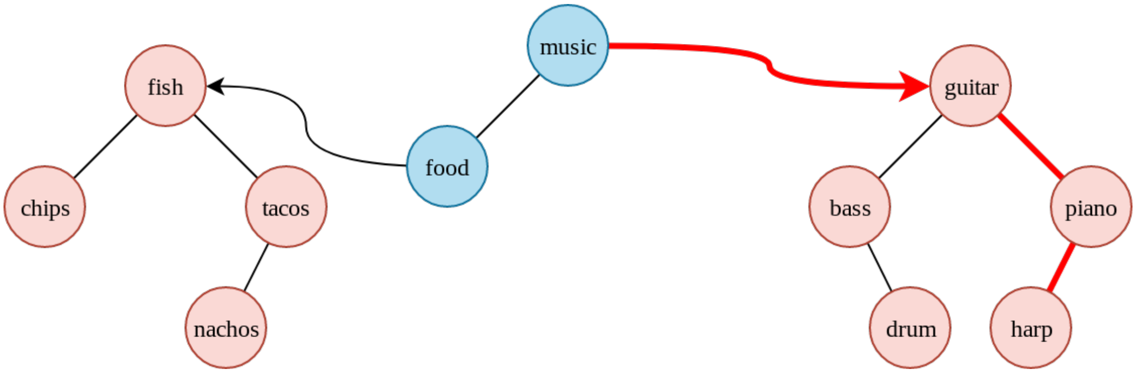

Figure 3: After the insertion of the photo, named ”harp”

In Figure 2, we have two primary nodes as categories in the main tree, together with their secondary trees, whose nodes use photo names as keys. We add a new photo with its category being *music*, and name being *harp*. We first search the main tree for the key *music*, then we create the SecondaryNode that keeps the photo data, and add that SecondaryNode to its proper place: as the left child of the *piano* SecondaryNode. Figure 3 shows the resulting TPBST after the insertion operation is completed.

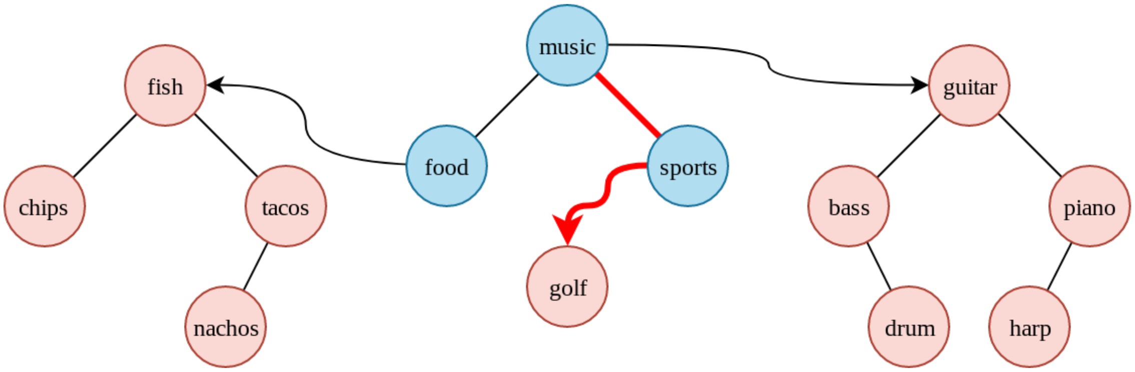

Figure 4: After the insertion of the photo, named ”golf”

In Figure 4, we add a new photo with its category being *sports*, and name being *golf*. We first search the main tree for the key *sports*, but it does not exist. So, we first create the *sports* `PrimaryNode`, then we create the `SecondaryNode` that keeps the photo data, and make the `rootSecondaryNode` of the *sports* node to point to that `SecondaryNode`. Figure 4 shows the resulting TPBST after the insertion operation is completed.

2.3.2 `TwoPhaseBST & remove(const std::string & primaryKey, const std::string & secondaryKey);`

Find the `SecondaryNode` by using the `primaryKey` to search the main tree, and the `secondaryKey` to search the corresponding secondary tree if the `PrimaryNode` exists. If the `SecondaryNode` is not found, do nothing. If it exists, remove that `SecondaryNode` by applying BST deletion procedures to preserve the lexicographical ordering among the nodes of the secondary tree (see Figure 5, then Figure 6). Note that only the `SecondaryNode`s can be removed; `PrimaryNode`s cannot be removed once created. In other words, the secondary tree of a `PrimaryNode` can become `NULL` if all of its `SecondaryNode`s are removed; but the `PrimaryNode` will still exist (see Figure 4, then Figure 5).

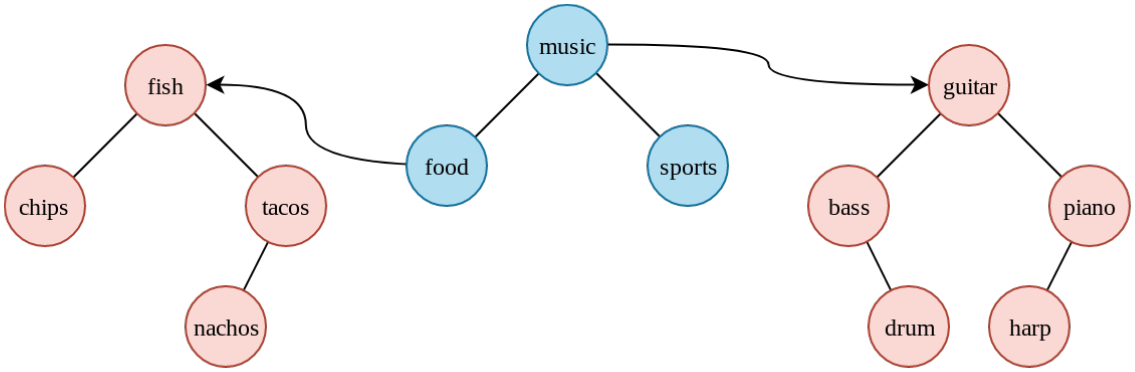

Figure 5: After the removal of the photo, named ”golf”.

2.3.3 `TwoPhaseBST & print(const std::string & primaryKey = "", const std::string & secondaryKey = "");`

Print function must print the details of TPBST by following the rules given below. For printing, ascending alphabetical order must be followed among the primary nodes of the main tree, and also among the secondary nodes of each secondary tree. Other alphabetical orderings will not be accepted. See the example outputs provided below. In all printing tasks, information will be written to the standard output in one line, which should end with a newline character.

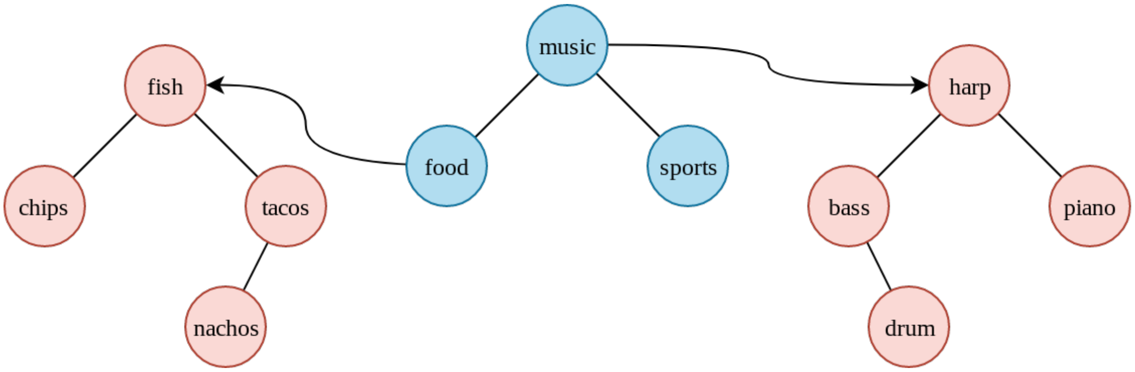

Figure 6: After the removal of the photo, named ”guitar”

- If both of the primary and secondary keys are not given (i.e. both parameters are empty strings), then information about all the items in the tree is printed. The strict printing format is as follows (this is a one-line output, it is shown in two lines to make it fit into the page):

```
{"p_key_1"␣:␣{"s_key_11"␣:␣"data_111",␣"s_key_12"␣:␣"data_112",␣...},␣
"p_key_2"␣:␣{"s_key_21"␣:␣"data_221",␣"s_key_22"␣:␣"data_222",␣...},␣...}
```

  In this format, p_key's are primary keys, s_key's are secondary keys, data's are string representations (returned by the overloaded `operator<<`) of objects stored in tree, and ␣ symbols represent **single** spaces. If the tree is empty (i.e. no primary and secondary nodes), then print as follows:

```
{}
```

  As an example, for the sample tree in Figure 6, the output of the `print()` function call must be as follows (data representations may change):

```
{"food"␣:␣{"chips"␣:␣"food;chips;res_low;10",␣"fish"␣:␣
"food;fish;res_high;20",␣"nachos"␣:␣"food;nachos;res_high;20",␣"tacos"␣:␣
"food;tacos;res_medium;15"},␣"music"␣:␣{"bass"␣:␣
"music;bass;res_high;25",␣"drum"␣:␣"music;drum;res_low;15",␣"harp"␣:␣
"music;harp;res_medium;20",␣"piano"␣:␣"music;piano;res_medium;20"},␣
"sports"␣:␣{}}
```

- If primary key is not given but secondary key is given, then you should not do anything (do not even print a newline). This case is not included in this assignment.

- If primary key is given but secondary key is not given, then print information about all items indexed with the given primary key. The strict printing format is as follows:

```
{"p_key"␣:␣{"s_key_1"␣:␣"data_1",␣"s_key_2"␣:␣"data_2",␣...}}
```

  If no primary node with such primary key exists in the tree, then print as follows:

```
{}
```

  If primary key exists in the tree, but it has no secondary nodes, then print as follows:

```
{"p_key"␣:␣{}}
```

For the sample tree in Figure 6, the output of the `print("music")` function call must be as follows (data representations may change):

```
{"music"␣:␣{"bass"␣:␣"music;bass;res_high;25",␣"drum"␣:␣
"music;drum;res_low;15",␣"harp"␣:␣"music;harp;res_medium;20",␣"piano"␣:␣
"music;piano;res_medium;20"}}
```

- If both of the primary and secondary keys are given, then print information about a specific item indexed with those keys. The strict printing format is as follows:

```
{"p_key"␣:␣{"s_key"␣:␣"data"}}
```

If no such item exist in the tree, then print as follows:

```
{}
```

For the sample tree in Figure 6, the output of the `print("music", "harp")` function call must be as follows (data representations may change):

```
{"music"␣:␣{"harp"␣:␣"music;harp;res_medium;20"}}
```

2.3.4 `T * find(const std::string & primaryKey, const std::string & secondaryKey);`

Search the main tree with the given `primaryKey` to find the `PrimaryNode` that we are interested in, and then search its secondary tree with the given `secondaryKey`. If the `SecondaryNode` is found, return a pointer to the data, which is of type `T`, stored in that `SecondaryNode`. If the `PrimaryNode` or the `SecondaryNode` do not exist, then return `NULL`.

3 Stock Photo Webstore Implementation (20 pts)

3.1 `Photo`

`Photo` class represents the information needed to be stored for each photo item of the webstore. A `Photo` object has four attributes: `category`, `name`, `resolution`, and `price`. Particular accessors and mutators together with the overloaded `operator<<` function for printing have already been implemented.

There is no single attribute that uniquely identifies photo items. However, no two photos with the same category and name may exist. All attributes of an item are set during the object construction and all of them other than `price` can not be changed later.

In the TPBST that will be utilized in the webstore application, `category` attributes will be used as primary keys and `name` attributes will be used as secondary keys. Inspect the complete implementations in *photo.hpp* and *photo.cpp* files. These files should not be edited.

3.2 `StockPhotoWebstore` Interface

In `StockPhotoWebstore` class, there is a single member variable named as `tpbst`, which is a `TwoPhaseBST` object. Information of all photos will be stored in this tree and no other information will be stored in webstore. All member functions should utilize this tree to operate as described in the following subsections.

Default constructor and `insert` functions have already been implemented. As stated before, `category` and `name` attributes of photos are given as keys to insert `Photo` objects to the tree. Do not change these functions. All member functions in `StockPhotoWebstore` class returns reference to the `StockPhotoWebstore` instance to allow method chaining. For all functions that you are required to implement, return statements have already been coded as the last line. Do not edit or delete them.

In *stock_photo_webstore.cpp* file, you need to provide implementations for following functions declared under *stock_photo_webstore.hpp* header to complete the assignment.

3.2.1 `StockPhotoWebstore & addPhoto(const Photo & photo);`

Inserts the given `Photo` object into the tree using its **category** and **name** attributes as keys. Note that actual data will be stored in the tree as a copy of the parameter of this function. This function has already been implemented. No photos that have the same category and name with the ones already in the tree will be given as parameter.

3.2.2 `StockPhotoWebstore & removePhoto(const std::string & category, const std::string & name);`

Remove the photo with the given **category** and **name** parameters. Do nothing (e.g. indicating it in output etc.) if no such photo exists.

3.2.3 `StockPhotoWebstore & updatePrice(const std::string & category, const std::string & name, int newPrice);`

Update the `price` variable of the `Photo` object that is specified by the given **category** and **name** parameters. Set it to the new price value given as parameter. Do nothing if no such photo exists.

3.2.4 `StockPhotoWebstore & printAllPhotos();`

Print information of all `Photo` objects in the tree. You should directly use the format described in `TwoPhaseBST` section. You can make use of the print function of TPBST. No extra formatting is required.

3.2.5 `StockPhotoWebstore & printAllPhotosInCategory(const std::string & category);`

Print information that belongs to all `Photo` objects in the tree with the given **category** parameter. You should directly use the format described in `TwoPhaseBST` section.

3.2.6 `StockPhotoWebstore & printPhoto(const std::string & category, const std::string & name);`

Print information of a single `Photo` object that is specified by the given **category** and **name** parameters. You should directly use the format described in `TwoPhaseBST` section.

4 Driver programs

To enable you to test your TPBST implementation and Stock Photo Webstore application, two driver programs, *main_tpbst.cpp* and *main_webstore.cpp* are provided. Their expected outputs are also provided in *output_tpbst.txt* and *output_webstore.txt* files, respectively.

5 Regulations

1. **Programming Language:** You will use C++.
2. Standard Template Library is allowed only for `list` and `stack`.
3. External libraries other than those already included are not allowed.
4. Those who do the operations (insert, remove, search, print) without utilizing the tree will receive 0 grade.
5. Those who modify already implemented functions and those who insert other data variables or public functions and those who change the prototype of given functions will receive 0 grade.
6. Those who use STL `vector` or compile-time arrays or variable-size arrays (not existing in ANSI C++) will receive 0 grade. Options used for `g++` are `-ansi -Wall -pedantic-errors -O0`. They are already included in the provided Makefile.
7. You can add private member functions whenever it is explicitly allowed.
8. **Late Submission:** A penalty of $5 * (daysLate)^2$ will apply for late submissions. Your assignment will not be accepted if you submit more than 3 days (72 hours) late.
9. **Cheating:** We have zero tolerance policy for cheating. In case of cheating, all parts involved (source(s) and receiver(s)) get zero. People involved in cheating will be punished according to the university regulations. Remember that students of this course are bounded to code of honor and its violation is subject to severe punishment.
10. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

6 Submission

- Submission will be done via Moodle (cengclass.ceng.metu.edu.tr).
- Do not write a *main* function in any of your source files.
- A test environment will be ready in Moodle.
  - You can submit your source files to Moodle and test your work with a subset of evaluation inputs and outputs.
  - Additional test cases will be used for evaluation of your final grade. So, your actual grades may be different than the ones you get in Moodle.
  - Only the last submission before the deadline will be graded.