DS-EN-3rd Year & Supplementary Year / (150 min) | **22.06.2021**

Implement a C-language application that manages the clients' reservations for a concert hall.

1. Write the source code sequence to create a ***Binary Search Tree,*** *refered as* ***BST****,* that shall store clients' reservations as items of type ***Event*.*** Insertion key into the BST structure should be ***idEvent***, next to other mandatory attributes like ***ticketPrice(float)***, ***clientName(char*)***, ***eventDate(char[10])***, and 2 other attributes at your choice. Inserting an event into the BST structure is made in a function which is called when creating the tree. The structure will be populated with at least 10 records read from an input file. ***(2p)***

   Implementation requirements:
   • Defining ***Event*** structure. (0,25p)
   • Strings from the file must accept blanks when read. (0,25p)
   • No memory leaks. (0,25p)
   • Implementing the logic of creating the BST structure. (0,75p)
   • Complete and correct implementation, all data inserted into the structure. (0,25p)
   • Testing the implementation in the **main()** function and displaying the content of the newly created BST. (0,25p)

2. Write and call the function for retrieving the reservations from the BST structure created above, which have the event date equal with a specified parameter of the function. Reservations are saved in an array and DO NOT share heap memory areas with the ones from the BST structure. The array is returned in ***main()*** by using the return type or the list of paramaters from the function. **(2p)**

   Implementation requirements:
   • Header function definition with I / O parameters, completely and correctly. (0,25p)
   • Deep-copy implementation for items saved in the array. (0,25p)
   • Implementing the logic of finding and saving the items in the array. (1p)
   • Complete and correct creation of the array. (0,25p)
   • Testing the implementation by calling the function and displaying the results returned after the call. (0,25p)

3. Write and call the function for calculating the total cost per each client for all the reservations found in the BST structure. The **values** that are calculated are saved in an array as pairs of two attributes ***(clientName, totalCost).*** The array and its size are returned in ***main()*** by using the return type or the list of paramaters from the function. **(2.5p)**

   Implementation requirements:
   • Header function definition with I / O parameters, completely and correctly. (0,25p)
   • Calculating the size of the array. (0,25p)
   • Implementing the logic of calculating the total cost for each client. (1,50p)
   • Complete and correct creation of the array. (0,25p)
   • Testing the implementation by calling the function and displaying the results returned after the call. (0,25p)

4. Write and call the function for **transforming** the BST structure from the 1) requirement, into two complementary BST structures, based on one of the optional attributes defined by you. The chosen attribute should have binary significance in order to be able to split the main tree into two complementary structures. The resulted structures are to be returned in ***main()*** and their content displayed at the console. **(2.5p)**

   Implementation requirements:
   • Header function definition with I / O parameters, completely and correctly. (0,25p)
   • Determining the elements that need to be inserted into the complementary BSTs. (0,25p)
   • Implementing the logic of creating two new complementary BST structures **without new memory allocations**. (1,25p)
   • Complete and correct creation of the BSTs. (0,50p)
   • Testing the implementation by calling the function and displaying the results returned after the call. (0,25p)

5. Write and call the functions that free the main structure, **2xBST** and **2xArrays**, as well as all the auxiliary structures used in the implementation of the requirements (if applicable). **(1p)**

   Implementation requirements:
   • Header function definition with I / O parameters, completely and correctly. (0,15p)
   • No memory leaks. (0,15p)
   • Update structure management variables in the main () function. (0,20p)
   • Logical implementation of freeing the data structures. (0,30p)
   • Testing the implementation, complete and correct freeing of structures and displaying the results at the console. (0,20p)
   • Absence of freeing the memory for auxiliary structures used. (-0,20p)

NOTES:

- **Projects with compilation errors won't be evaluated.**
- **Implementations must not contain globally defined or static variables.**
- **Implementations must not use predefined structures such as STL or 3rd party libraries.**
- **Plagiarized implementations will be evaluated with 0 points, regardless of the source.**
- **All requirements must be called and demonstrated in the main () function to be evaluated.**
- **Art. 72 (1) For the following facts, students will be expelled without the right to re-enroll in the Academy of Economic Studies in Bucharest:**
  - **(c) attempting to fraudulently pass examinations or other assessments;**