Implement a C-language application that stores all students' evaluations from the evaluation period.

1. Write the source code sequence to create a circular simple linked list that shall store students' evaluations for **different exams**. The structure will be populated with at least 15 records read from the file. The ***Evaluation*** structure will be defined so that it should contain 5 attributes: ***int (evaluationId)***, ***int (noCredits)***, ***pointer of type char (examName)***, ***int (student id), double (grade)***. **(2p)**

   Implementation requirements:
   • Defining ***Evaluation*** 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 circular simple linked list. (0,75p)
   • Complete and correct implementation, all data inserted into the structures. (0,50p)

2. Write and call the function for creating a one dimensional array containing the total number of students' evaluations per each exam found in the list. The array, returned in **main()** by the return type of the function, should be used to display the number of evaluations per exam. **(2p)**

   Implementation requirements:
   • Header function definition with I / O parameters, completely and correctly. (0,25p)
   • No memory leaks. (0,25p)
   • Dynamic allocation of structures. (0,25p)
   • Logical implementation for building dynamic arrays. (0,75p)
   • Testing the implementation in the **main()** function. (0,50p)

3. Write and call the function to delete all student's evaluations from the available circular simple linked list based on the student id, given as a parameter. **(1.5p)**

   Implementation requirements:
   • Header function definition with I / O parameters, completely and correctly. (0,25p)
   • Logical implementation for deleting the evaluations. (0,75p)
   • Testing the implementation in the main () function. (0,50p)

4. Write and call the function to transform the list structure into a binary search tree using the ***evaluation id*** field. The function will return in **main ()** the root of the tree. The content of the tree must be displayed using the inorder traversal. **(1.5p)**

   Implementation requirements:
   • Header function definition with I / O parameters, completely and correctly. (0,25p)
   • Structure conversion without reallocationg new elements. (0,25p)
   • Logical implementation for the transformation into the new form of representation. (0,75p)
   • Testing the implementation and displaying the elements as requested in the **main()** function. (0,25p)

5. Write and call the function to transform the tree structures into the ***left child - right sibling*** representation model. The function will return in **main ()** the arrays that are specific to the representation of the binary search tree. The content of the arrays must be displayed according to the following model: **NODE ←→ LIST_OF_DESCENDANTS**. **(2p)**

   Implementation requirements:
   • Header function definition with I / O parameters, completely and correctly. (0,25p)
   • No memory leaks. (0,25p)
   • Dynamicaly allocated arrays. (0,25p)
   • Logical implementation for the transformation into the new form of representation. (0,75p)
   • Testing the implementation and displaying the elements as requested in the **main()** function. (0,50p)

6. Write and call the functions that free the main structure, 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. (0,25p)
   • 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. (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.**
- **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;**