Task

Coursework2 is actually the continuation of Coursework1. It uses the same data structure, items and *ICS0017DataSource.dll*.

Implement class *DataStructure* containing the following public functions (text printed in blue depends on the type of item, specify it yourself):

1. *DataStructure();*
   Constructor that creates empty data structure.
2. *DataStructure(char *pFilename);*
   Constructor that reads data from a binary file. The file was created by function *Write* (see below). Fails if there occur problems with file handling.
3. *~DataStructure();*
   Destructor that deletes all the items, vectors of pointers and headers.
4. *DataStructure(const DataStructure &Original);*
   Copy constructor.
5. *int GetItemsNumber();*
   Returns the current number of items in data structure.
6. *pointer_to_item GetItem(char *pID);*
   Returns pointer to item with the specified ID. If the item was not found, returns 0.
7. *void operator+=(pointer_to_Item);*
   Operator function to add a new item into data structure. Fails if the data structure already contains an item with the specified ID. Usage example:
   *DataStructure *pds = new DataStructure;*
   *ITEM5 *p = (ITEM5 *)GetItem(5);*
   **pds += p;*
8. *void operator-=(char *pID);*
   Operator function to remove and destroy item with the specified ID. Fails if there is no item with the specified ID. Usage example:
   **pds-= (char *)"Dark Khaki";*
9. *DataStructure &operator=(const DataStructure &Right);*
   Operator function for assignment. Do not forget that before the assignment you have to destroy all the existing contents. Usage example:
   DataStructure ds;
   *ds = *pds;*
10. *Int operator==(DataStructure &Other);*
    Operator function for comparison. Two data structures are equal if they contain the same number of items and for each item in the first structure there is a 100% matching item in the second structure. The order in the linked lists may be different. Returs 0 (not equal) or 1 (equal).

Usage example:
*cout << (ds == *pds) << endl;*

11. *void Write(char *pFilename);*
Serializes the data structure and writes into the specified file. Fails if there occur problems with file handling or if the data structure is empty.

12. *friend std::ostream &operator<<(std::ostream &ostr, const DataStructure &str);*
Prints all the items into command prompt window. Usage example:
*cout << *pds << endl << endl;*

Requirements

1. For memory allocation and release use operators *new* and *delete*.
2. The new items must be created with function *GetItem()* from *ICS0017DataSource.dll.* It guarantees that the item is correct.
3. For string copy use function *strcpy_s*.
4. For input and output use methods from *iostream* (*cin*, *cout* , etc.).
5. For file operations use methods from *fstream*.
6. In case of failure any of the functions must throw an object of standard class *exception*.
7. You may add into data structure attibutes and private fubctions as you consider feasible. But all the attributes must be *private*.
8. Write also a simple command prompt menu for testing and checking.

Evaluation

The deadline is the end of examinations in January. However, it is strongly advised to present the results of coursework during the semester. The students can do it after each lecture.

To get the assessment the students must attend personally. Electronically (e-mail, etc.) sent courseworks are neither accepted nor reviewed.

The students must prepare test functions (see below) verifying that all the implemented class functions are working correctly.

Presenting the final release is not necessary. It is OK to demonstrate the work of application in Visual Studio environment.

Test cases

Write function *main()* that:

1. Creates empty data structure.

2. Inserts 10 items into data structure (in a simple loop that calls 10 times *operator+*). To create items use function *GetItem* from DLL (read *ICS0017DataSource.h*)
3. Prints the resulting data structure.
4. Prints the number of elements in data structure (using *GetItemsNumber*).
5. Retrieves item with ID *Light Cyan*.
6. Tries to retrieve non-existing element *X X*.
7. Using the copy constructor creates the copy of current structure.
8. Removes items *Banana Mania*, *Persian Green* and *Vegas Gold* from the initial structure.
9. Compares the initial structure (now with 7 items) with the copy structure.
10. Writes the initial structure (now 7 items) into a data file.
11. Creates a new structure from this data file and compares it with initial stucture.
12. Asssigns to the structure just created (7 items) the copy created in step 7 (10 items) and prints the result.