Task

Write a C/C++ application containing the following four classes and additional functions supporting the running and testing.

1. Class Date

Class *Date* is implemented by instructor. Files *Date.h* and *Date.cpp* are in folder *General*.

2. Class Time

Class *Time* is implemented by instructor. Files *Time.h* and *Time.cpp* are in folder *General*.

3. Class Item

```
class Item
{
private:
	char Group;                           // Any from range 'A'...'Z'
	int Subgroup;                         // Any from range 0...99
	string Name;                          // Any, but not empty
	variant<Date, Time> Timestamp;        // Any object of class Date or class Time
	.............                         // To do
public:
	Item();                               // Creates a pseudo-random item, implemented by
	                                      // instructor
	Item(char, int, string, Date);
	Item(char, int, string, Time);
	Item(char, int, string, variant<Date, Time>);
	Item(const Item &);                   // copy constructor
	~Item();
	.............                         // To do
};
```

4. Class Data

Class *Data* must contain a C++ STL container containing objects of class *Item*. Its methods must allow the user to see the contents of container and to edit it.

The private member implementing the container must be:

```
map<char, map<int, vector<Item> *> > DataStructure;
```

It is a C++ map in which the *Item* members *Group* are the keys. The values are inner C++ maps in which the keys are *Item* members *Subgroup* and values are pointers to vectors. The vectors contain items belonging to the specified group and subgroup. Members of an *Item* vector may have the same *Names* and *Timestamps*.

Class *Data* must contain the following public methods:

1. `Data(int n);`
   Constructs the object and fills the container with *n* random items.

2. Data();
   Constructs the object with empty container;
3. `~Data();`
   Destructs the object and releases all the memory occupied by the container and the items in it.
4. `void PrintAll();`
   Prints all the items stored in the container in command prompt window in easily readable format (see Appendix). Items from the same group and subgroup must be ordered by their names.
5. `int CountItems();`
   Returns the total number of items in the container.
6. `map<int, vector<Item> *>* GetGroup(char c);`
   Returns the pointer to *map* containing all the items from group *c*. If the group does not exist, returns *nullptr*.
7. `void PrintGroup(char c);`
   Prints all the items from group *c* in command prompt window in easily readable format (see Appendix). Items from the same subgroup must be ordered by their names. If the group was not found, throws *invalid_argument_exception*.
8. `int CountGroupItems(char c);`
   Returns the current number of items in group *c*. If the group does not exist, returns 0.
9. `vector<Item>* GetSubgroup(char c, int i);`
   Returns the pointer to vector containing all the items from subgroup *I* from group *c*. If the subgroup does not exist, returns *nullptr*.
10. `void PrintSubgroup(char c, int i);`
    Prints all the items from subgroup i from group *c* in command prompt window in easily readable format (see Appendix). Items must be ordered by their names. If the subgroup was not found, throws *invalid_argument_exception*.
11. `int CountSubgroupItems(char c, int i);`
    Returns the current number of items in subgroup *i* from group *c*. If the subgroup does not exist, returns 0.
12. `optional<Item> GetItem(char c, int i, string s);`
    Returns the first of items specified by group *c*, subgroup *i* and name *s*. If the item was not found returns *nullopt.*
13. `void PrintItem(char c, int i, string s);`
    Prints the first of items specified by group *c*, subgroup *i* and name *s*. If the item was not found throws *invalid_argument_exception*.
14. `optional<Item> InsertItem(char c, int i, string s, optional<variant<Date, Time>> v = nullopt);`
    Creates, inserts and returns the specified item. If the input parameters are not correct, returns *nullopt*. If necessary, creates the missing group and subgroup. If *Timestamp* is not specified, use *Date::CreateRandomDate(Date, Date)* to create. Parameter *s* cannot be empty string.
15. `vector<Item>* InsertSubgroup(char c, int i, initializer_list<tuple<string, optional< variant<Date, Time>>>> items);`
    Creates and inserts the specified subgroup. The initializer_list *items* contains tuples specifying the *Names* (cannot be empty strings) and *Timestamps* (if not specified use *Data::CreateRandomDate(Date, Date)* to create) of new items. Returns the pointer to new

subgroup. If the specified subgroup already exists or the input parameters are not correct, returns *nullptr*. If necessary, creates the missing group.

16. `map<int, vector<Item> *>* InsertGroup(char c, initializer_list<int> subgroups, initializer_list<initializer_list<tuple<string, optional<variant<Date, Time>>>>> items);`
    Creates and inserts the specified group. The *subgroups* initializer_list presents the subgroups to be included into the new group. The *items* initializer_list contains initializer_lists presenting tuples specifying the *Names* (cannot be empty strings) and *Timetamps* (if not specified use *Data::CreateRandomDate(Date, Date)* to create) of new items. The first initializer_list from *items* corresponds to the first integer in *subgroups*. Returns the pointer to new group. If the specified group already exists or the input parameters are not correct, returns *nullptr*.
17. `bool RemoveItem(char c, int i, string s);`
    Removes the specified item. If after removing the subgroup has no members, remove it too. If after that the group is empty, remove it also. All the not used memory must be released. Return value: *false* if the item was not found, otherwise *true*.
18. `bool RemoveSubgroup(char c, int i);`
    Removes the specified subgroup. If after removing the corresponding group has no members, remove it too. All the not used memory must be released. Return value: *false* if the subgroup was not found, otherwise *true*.
19. `bool RemoveGroup(char c);`
    Removes the specified group. All the not used memory must be released. Return value: *false* if the group was not found, otherwise *true*.

General requirements

1. Start the project as Visual Studio Empty Application.
2. You must use the software implemented by instructor (*main(),* the preparatory and testing functions, implementation of class *Date*, constructor of class *Item*).
3. Instead of simple *for* loops try to use range-based *for* loops, methods built into C++ containers and algorithms like *for_each* from *<algorithm>*.

Evaluation

The deadline is the lecture on week 10. However, the students can present his / her results earlier.

A student can get the assessment only if he / she attends personally. Electronically (e-mail, cloud, GitHub, etc.) sent courseworks are neither accepted nor reviewed. Presenting the final release is not necessary. It is OK to demonstrate the work of application in Visual Studio environment.

The coursework is approved if the test function implemented by instructor runs until end and produces results matching the presented pattern.

Appendix: printout example

A:

2: Great Crested Grebe 14:59:29

6: Baikal Teal 08 Sep 2018

B:

73: Ruddy Turnstone 03 Dec 2018

C:

6: Redwing 14 May 2018

26: Moorland Francolin 16:08:29

E:

50: Two-barred Greenish Warbler 04 Jan 2018

H:

80: Red-billed Tropicbird 00:25:60

I:

26: Cuckoo 04 Aug 2018

J:

2: Summer Tanager 21:08:38

K:

3: Song Sparrow 27 Feb 2018

27: Brambling 16 Mar 2018

L:

4: Grey-breasted Spurfowl 06:30:17

76: Alpine Swift 18 May 2018

O:

71: Guillemot 30 Sep 2018

P:

9: Black-throated Green Warbler 01 May 2018

18: Upland Sandpiper 02:55:38

79: Icterine Warbler 14:11:13

T:

93: Nubian Nightjar 13 Aug 2018

U:

91: Crag Martin 06:22:34

V:

61: Lapland Bunting 15 Sep 2018