

Initial data

The initial data is located into a data structure consisting of arrays of pointers, headers and items. Declarations of items as C / C++ *structs* are presented in file *Items.h*. There are 10 different types of items (*ITEM1*, *ITEM2*, ..., *ITEM10*). Declarations of headers as C / C++ *structs* are presented in file *Headers.h*. There are 5 different types of headers (*HEADER_A*, *HEADER_B*, *HEADER_C*, *HEADER_D*, *HEADER_E*). The both files are stored in [Instructor's stuff](#).

There are 5 different types of data structures (*Struct1*, *Struct2*, *Struct3*, *Struct4*, *Struct5*). To generate the initial data structure you have to use functions from *ICS0017DataSource.dll*. This DLL is implemented by instructor and stored in [Instructor's stuff](#). It needs auxiliary file *Colors.txt*, created from https://en.m.wikipedia.org/wiki/Lists_of_colors.

To understand the building principles of our data structures analyse the examples on the following pages. Let us emphasize that they are just examples: the actual presence and absence of items and headers is determined by the work of item generator built into *ICS0017DataSource.dll* and is largely occasional.

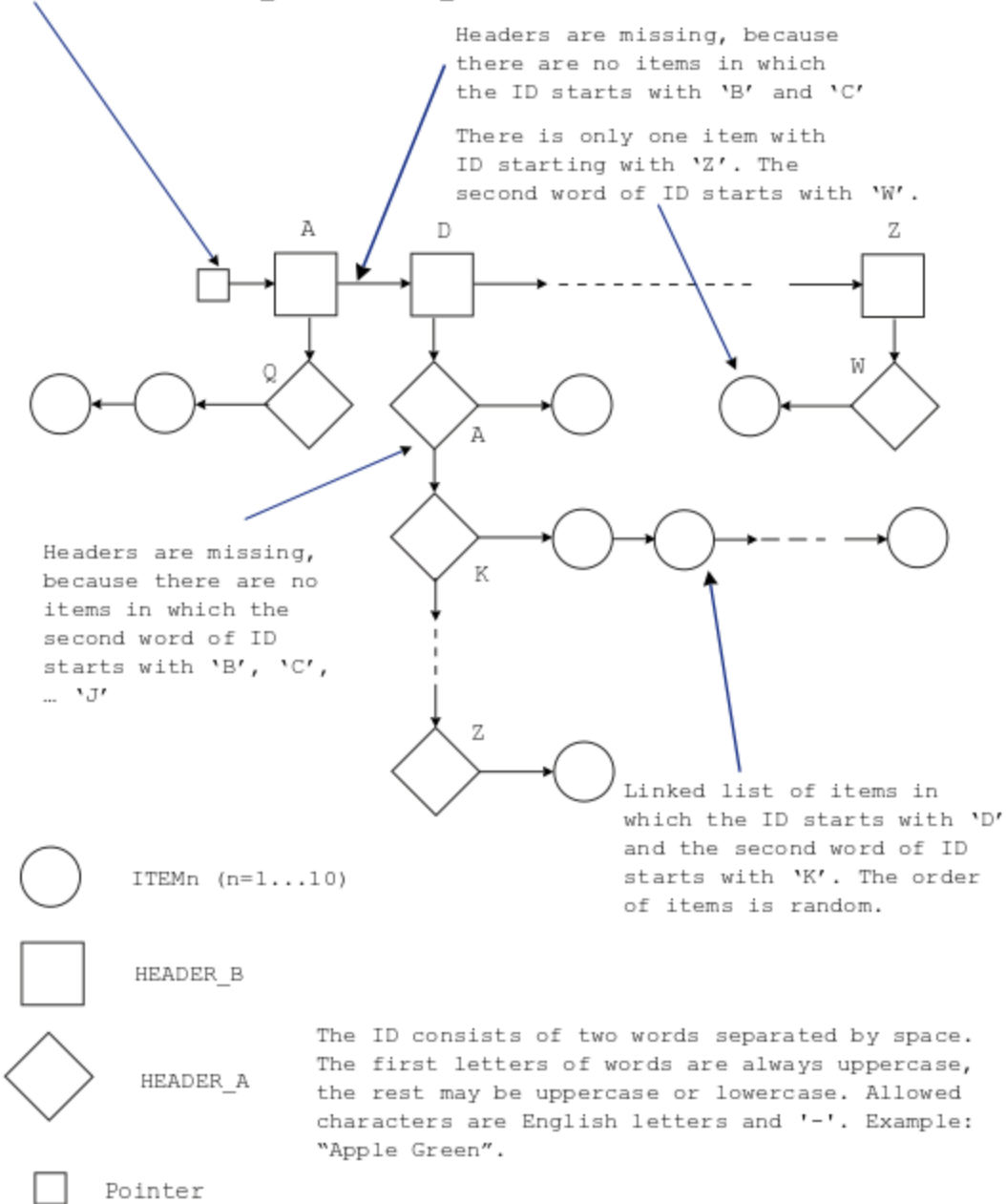
The DLL imports 6 public functions declared in file *ICS0017DataSource.h* (also stored in [Instructor's stuff](#)). Five of them create data structure and return the pointer to it. The sixth function (*GetItem()*) constructs a stand-alone item and returns the pointer to it. There is also a password-protected function for the instructor. Comments explaining the usage of public functions are in *ICS0017DataSource.h*.

To know which item and data structure you have to use contact the instructor.

Struct1 example:

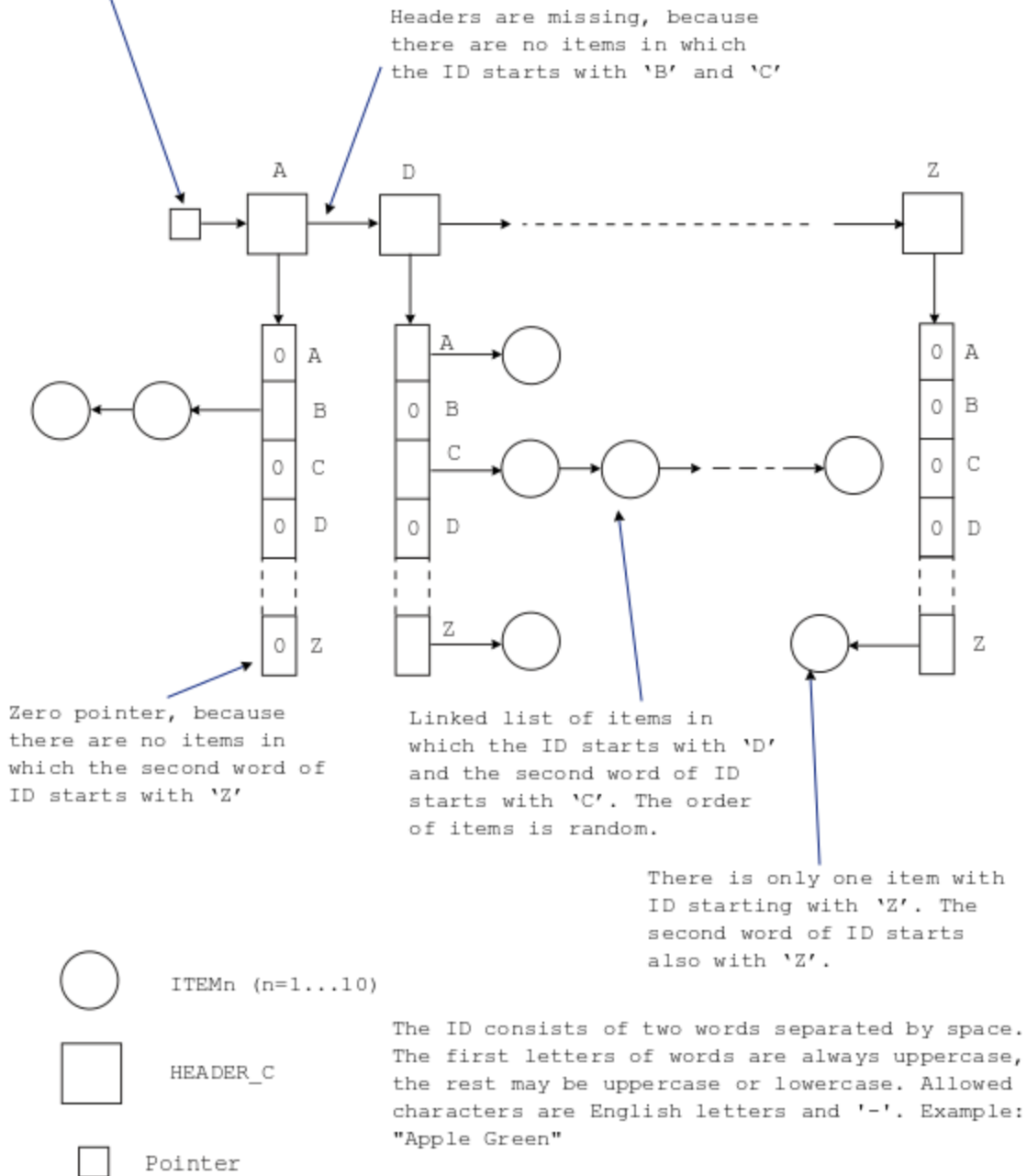
```
HEADER_B *pStruct1 = GetStruct1(nItemType, nItems);
```

The lists of HEADER_A and HEADER_B are ordered



Struct2 example:

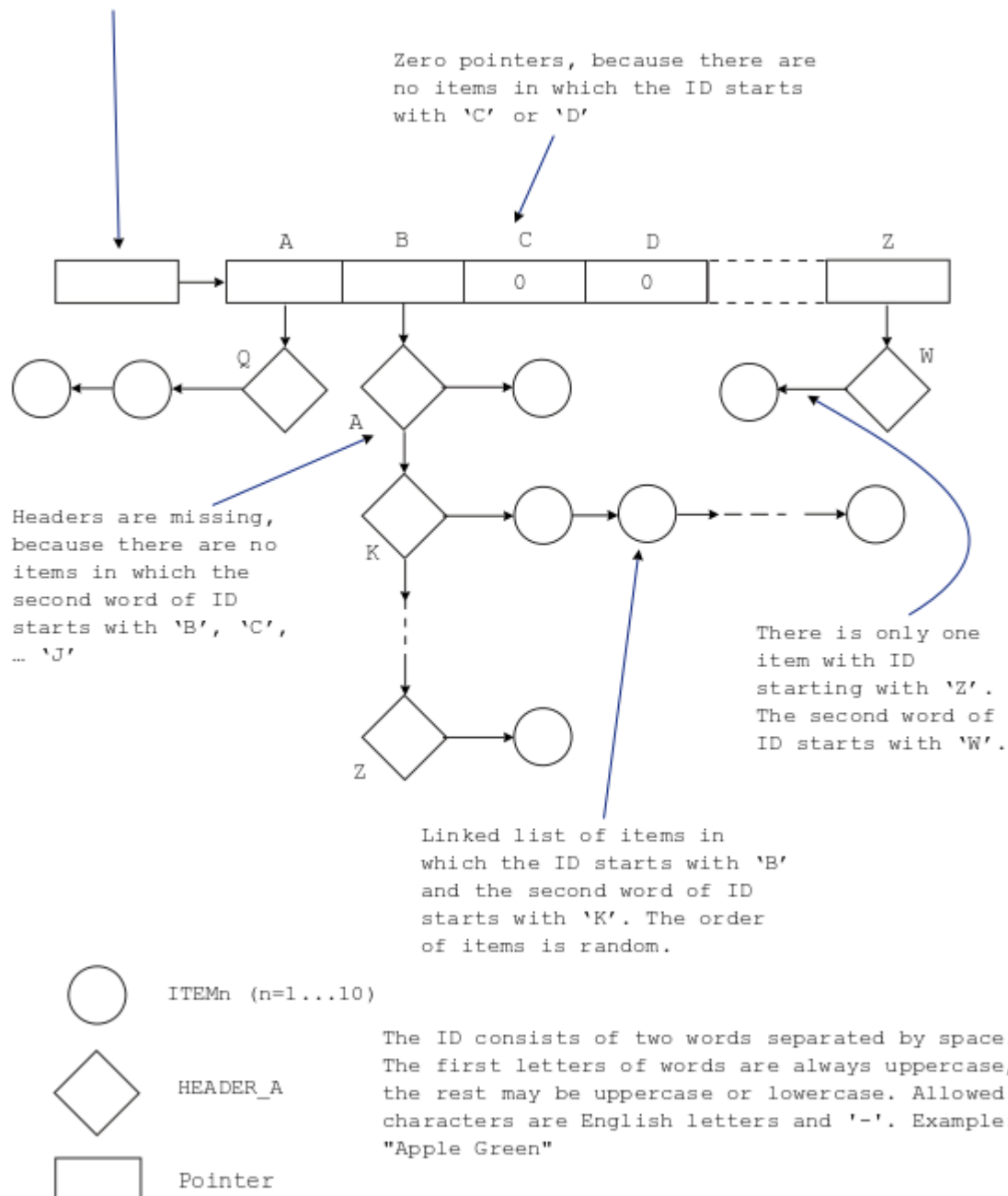
```
HEADER_C *pStruct2 = GetStruct2(nItemType, nItems);
The list of HEADER_C is ordered
```



Struct3 example:

```
HEADER_A **ppStruct3 = GetStruct3(nItemType, nItems);
```

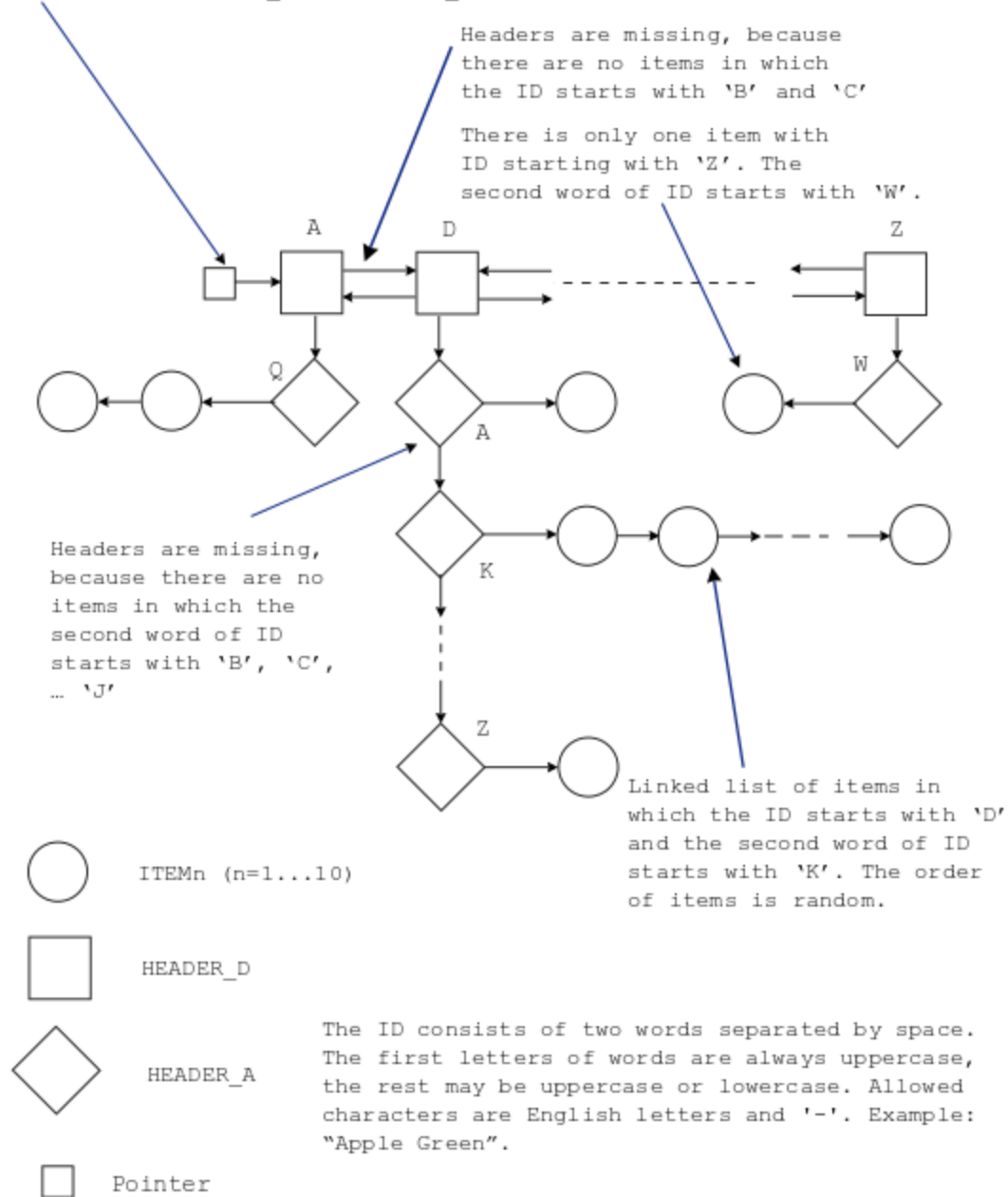
The lists of HEADER_A are ordered



Struct4 example:

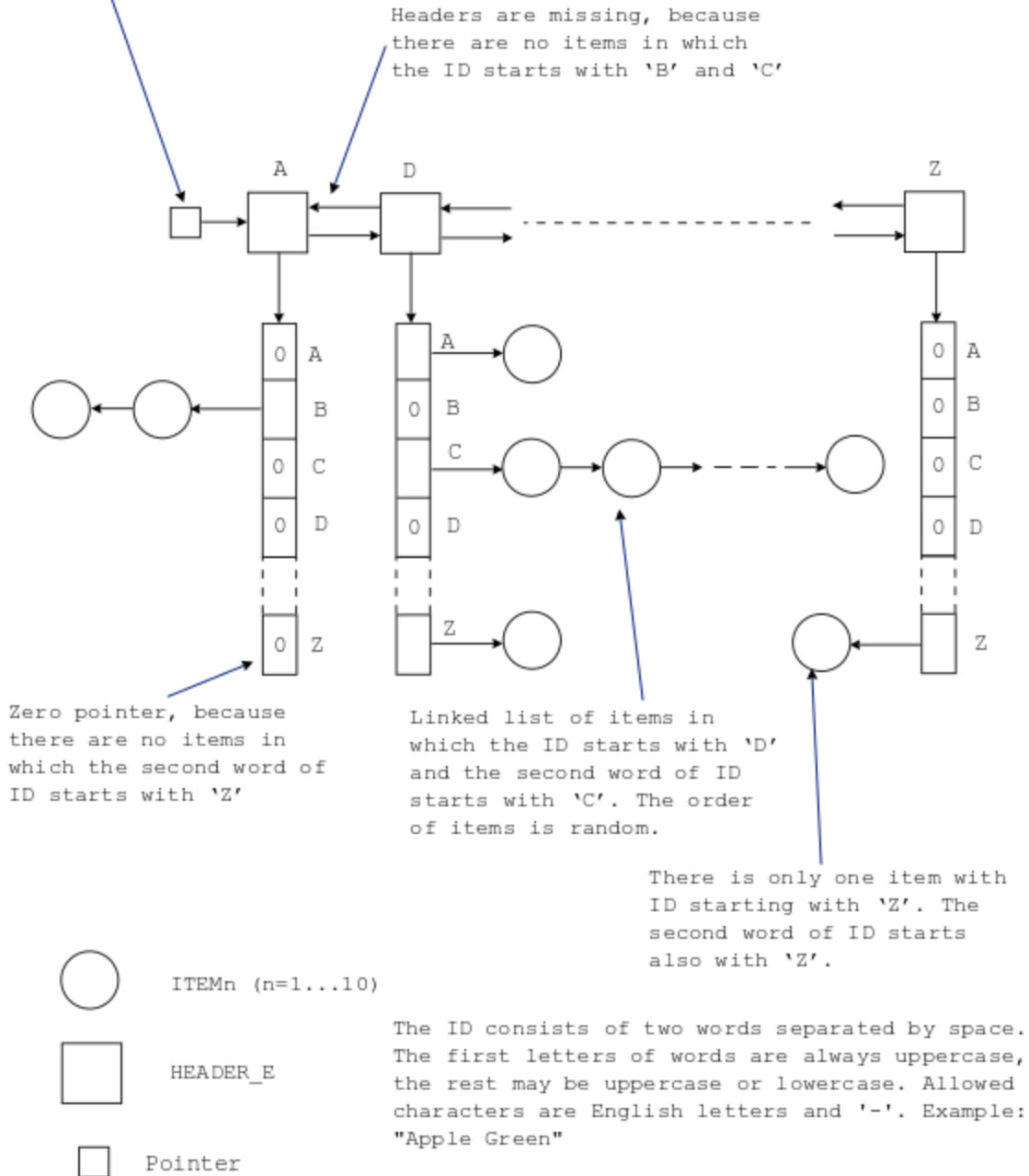
```
HEADER_D *pStruct4 = GetStruct4(nItemType, nItems);
```

The lists of HEADER_A and HEADER_D are ordered



Struct5 example:

```
HEADER_E *pStruct5 = GetStruct5(nItemType, nItems);
The list of HEADER_E is ordered
```



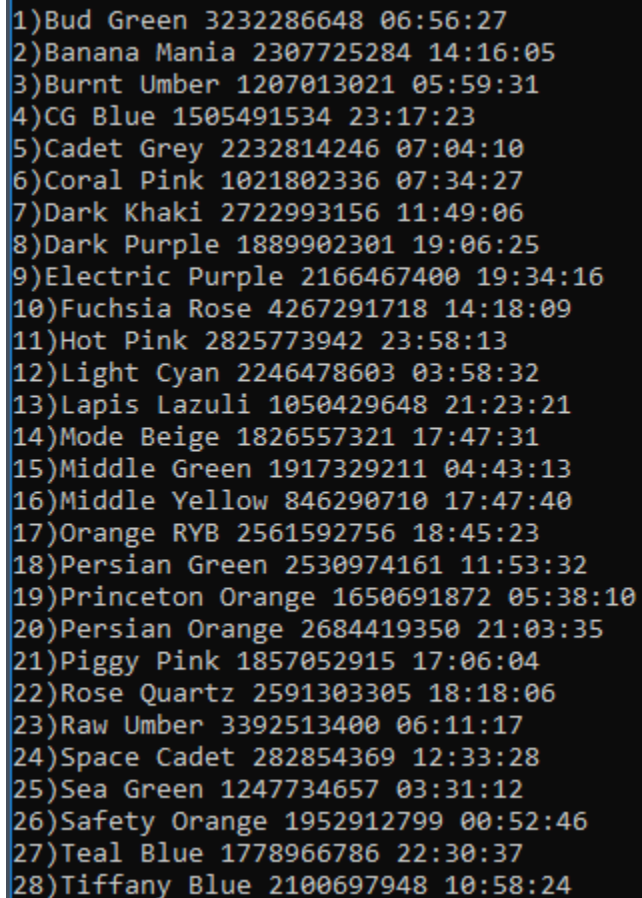
Task #1

Write a C / C++ function that prints in the command prompt window the contents of all the items from the current data structure.

Prototype (text printed in blue depends on the type of data structure, specify it yourself):

```
void PrintDataStructure(pointer_to_data_structure);
```

The output should be similar to the following:



```
1)Bud Green 3232286648 06:56:27
2)Banana Mania 2307725284 14:16:05
3)Burnt Umber 1207013021 05:59:31
4)CG Blue 1505491534 23:17:23
5)Cadet Grey 2232814246 07:04:10
6)Coral Pink 1021802336 07:34:27
7)Dark Khaki 2722993156 11:49:06
8)Dark Purple 1889902301 19:06:25
9)Electric Purple 2166467400 19:34:16
10)Fuchsia Rose 4267291718 14:18:09
11)Hot Pink 2825773942 23:58:13
12)Light Cyan 2246478603 03:58:32
13)Lapis Lazuli 1050429648 21:23:21
14)Mode Beige 1826557321 17:47:31
15)Middle Green 1917329211 04:43:13
16)Middle Yellow 846290710 17:47:40
17)Orange RYB 2561592756 18:45:23
18)Persian Green 2530974161 11:53:32
19)Princeton Orange 1650691872 05:38:10
20)Persian Orange 2684419350 21:03:35
21)Piggy Pink 1857052915 17:06:04
22)Rose Quartz 2591303305 18:18:06
23)Raw Umber 3392513400 06:11:17
24)Space Cadet 282854369 12:33:28
25)Sea Green 1247734657 03:31:12
26)Safety Orange 1952912799 00:52:46
27)Teal Blue 1778966786 22:30:37
28)Tiffany Blue 2100697948 10:58:24
```

If the data structure is empty, print an error message.

Task #2

Write a C / C++ function that inserts a new item into the current data structure.

Prototype (text printed in blue depends on the type of data structure, specify it yourself):

```
pointer_to_data_structure = InsertItem(pointer_to_data_structure, char *pNewItemID = 0);
```

The new item must be constructed by function *GetItem()* from *ICS0017DataSource.dll*. The user may set the ID itself or set the pointer to it to zero. In the last case the ID is selected by *GetItem()*.

You may freely select the position of new item in the linked list of items.

The return value is the pointer to the modified data structure.

The function must keep the current contents of data structure and throw an exception if:

- An item with same ID is already in the data structure.
- The ID presented by user does not follow the formatting rules.

Task #3

Write a C / C++ function that inserts a new item into the current data structure.

Prototype (text printed in blue depends on the type of data structure, specify it yourself):

pointer_to_data_structure = RemoveItem(*pointer_to_data_structure*, char *pItemID);

The memory fields occupied by the item to be removed must be released (use operator *delete*). Do not forget that if a header has lost all the items or other headers associated with it, this header itself must also disappear.

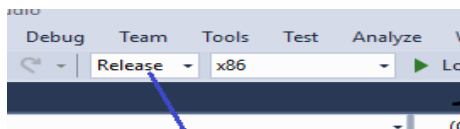
The return value is the pointer to the modified data structure.

The function must keep the current contents of data structure and throw an exception if:

- Item with the specified ID does not exist.
- The ID presented by user does not follow the formatting rules.

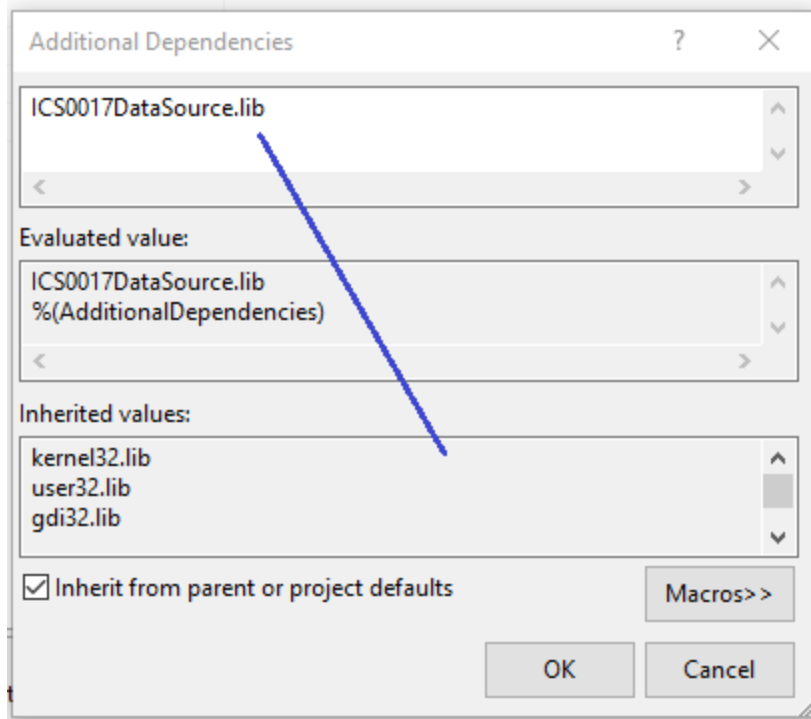
First steps

1. Launch Visual Studio and start the new project. The project template must be *Visual C++ Windows Console Application*. Suppose that the project name you have selected is *Coursework1* and the location folder is *C:\ICS0017*. The wizard creates project file *C:\0017\Coursework1.sln* and source code folder *C:\ICS0017\Coursework1\Coursework1*. Into source code folder it puts files *Coursework1.cpp* containing a simple *main()* function and also some auxiliary files.
2. Build your solution. Then set the configuration to *Release* and build once more. Now you have all the folders you need.



Configuration setting

3. Set the configuration back to *Debug*. From [Instructor's stuff](#) extract *DateTime.h*, *Headers.h*, *Items.h*, *ICS0017DataSource.h*, *Colors.txt* and *ICS0017DataSource.lib*. Store them in source code folder *C:\ICS0017\Coursework1\Coursework1*. In the Visual Studio *Solution Explorer* right-click Header Files and from the pop-up menu select *Add* → *Existing Item*. From the existing file list select all the four **.h* files and click *Add*.



7. Now you may test if your project is well prepared. Suppose you must use Struct2 and ITEM3 and you want to create the structure containing 100 items. Then write:

```
#include <iostream>
#include "DateTime.h"
#include "Items.h"
#include "Headers.h"
#include "ICS0017DataSource.h"
// IMPORTANT: follow the given order of *.h files: ICS0017DataSource.h must be
// the last
int main()
{
    HEADER_C *p = GetStruct2(3, 100);
    ITEM3 *pNewItem = (ITEM3 *)GetItem(3);
    return 0;
}
```

This program should run. To see the results you have to implement *PrintStruct()* function.

To run your application without Visual Studio, put your release *.exe together with *Colors.txt* and *ICS0017DataSource.dll* into a separate folder.

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 3 implemented 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. Sets the number of items to 30 and prints the data structure.
2. One after another inserts new items with identifiers: *Z A, Z Z, Z K, A Z, A A, A K, G Z, G A, G K, M A, M Ba, M Bb, M Z*.
3. Tries to insert items with identifier *M Ba* and *Mba* and prints the error messages.
4. Prints the new data structure of 43 items.
5. One after another removes the items that were just inserted.
6. Tries to remove items with identifier *M Ba* and *Mba* and prints the error messages.
7. Prints the data structure.