Overview

First read section *Kursusetööd* from Ülevaade

Initial data

The initial data is located in a data structure consisting of arrays of pointers, headers and items. Declarations of items as C / C++ *struct*s are presented in file *Items.h*. There are 10 different types of items (*ITEM1, ITEM2, …, ITEM10*). Declarations of headers as C / C++ *struct*s 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 Files for coursework #1.

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 *DataSource.dll*. This DLL is implemented by instructor and stored in Files for coursework #1. It needs auxiliary file *Colors.txt*, created from https://en.m.wikipedia.org/wiki/Lists_of_colors.

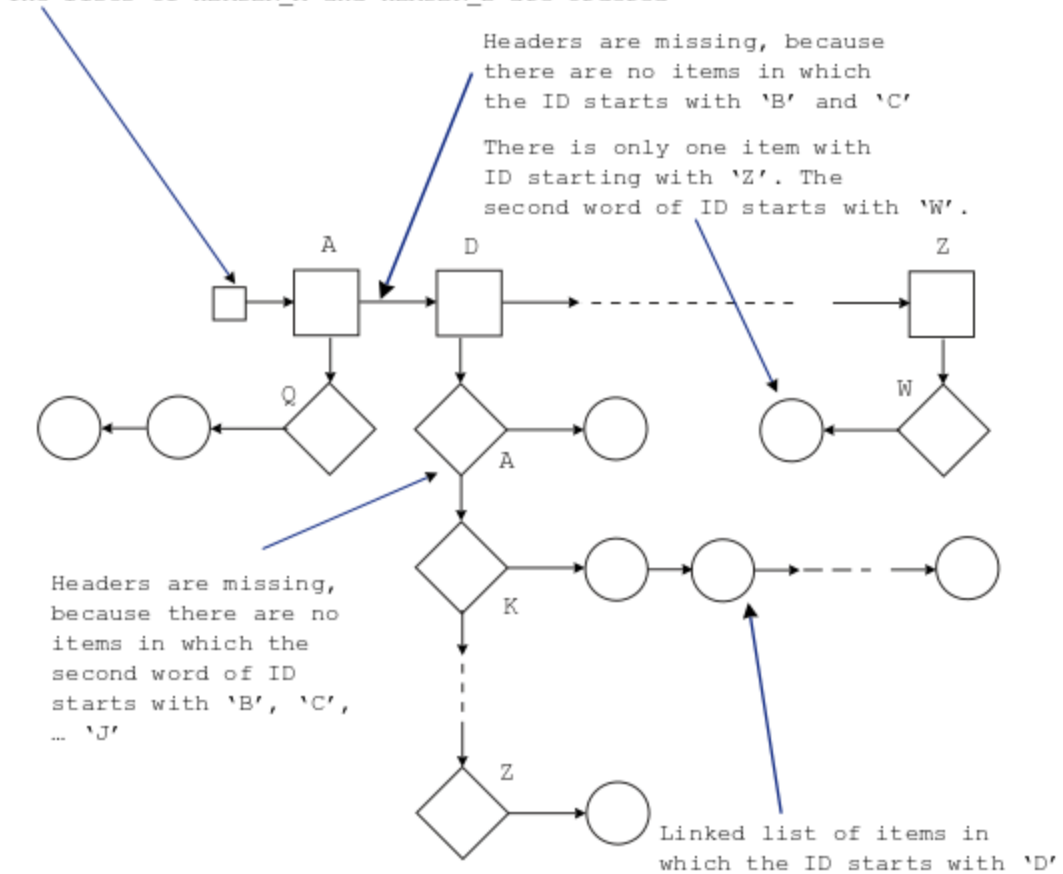
To understand the building principles of our data structures analyse the examples on the following pages. Let us emphasize that they are just examples: the actual presence and absence of items and headers is determined by the work of item generator built into *DataSource.dll* and is largerly occasional.

The DLL imports 6 public functions declared in file *DataSource.h* (also stored in Files for coursework #1). 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 *DataSource.h.*

To know which item and data structure you have to use see the table.

Struct1 example:

HEADER_B *pStruct1 = GetStruct1(nItemType, nItems);
The lists of HEADER_A and HEADER_B are ordered



ITEMn (n=1...10)

HEADER_B

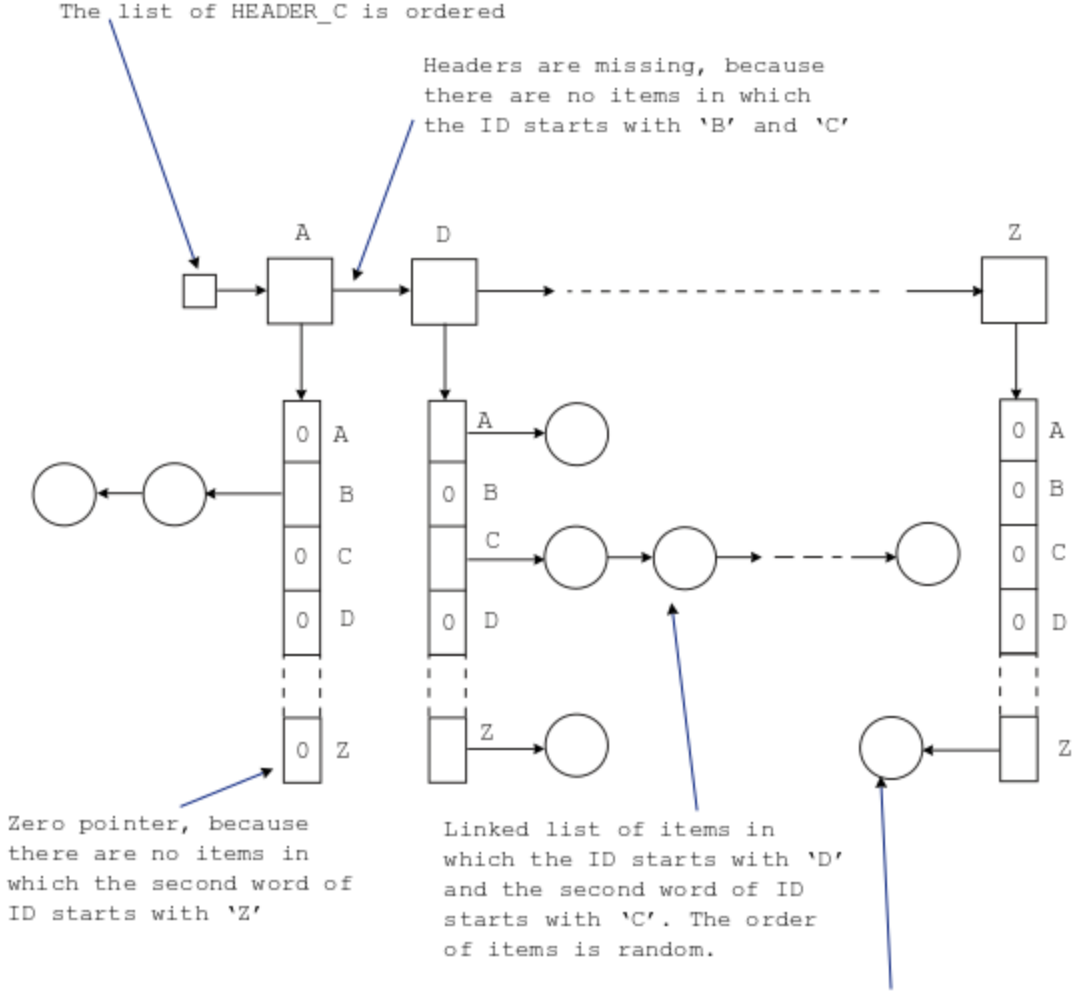
HEADER_A

The ID consists of two words separated by space. The first letters of words are always uppercase, the rest may be uppercase or lowercase. Allowed characters are English letters and '-'. Example: "Apple Green".

Pointer

Struct2 example:

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



ITEMn (n=1...10)

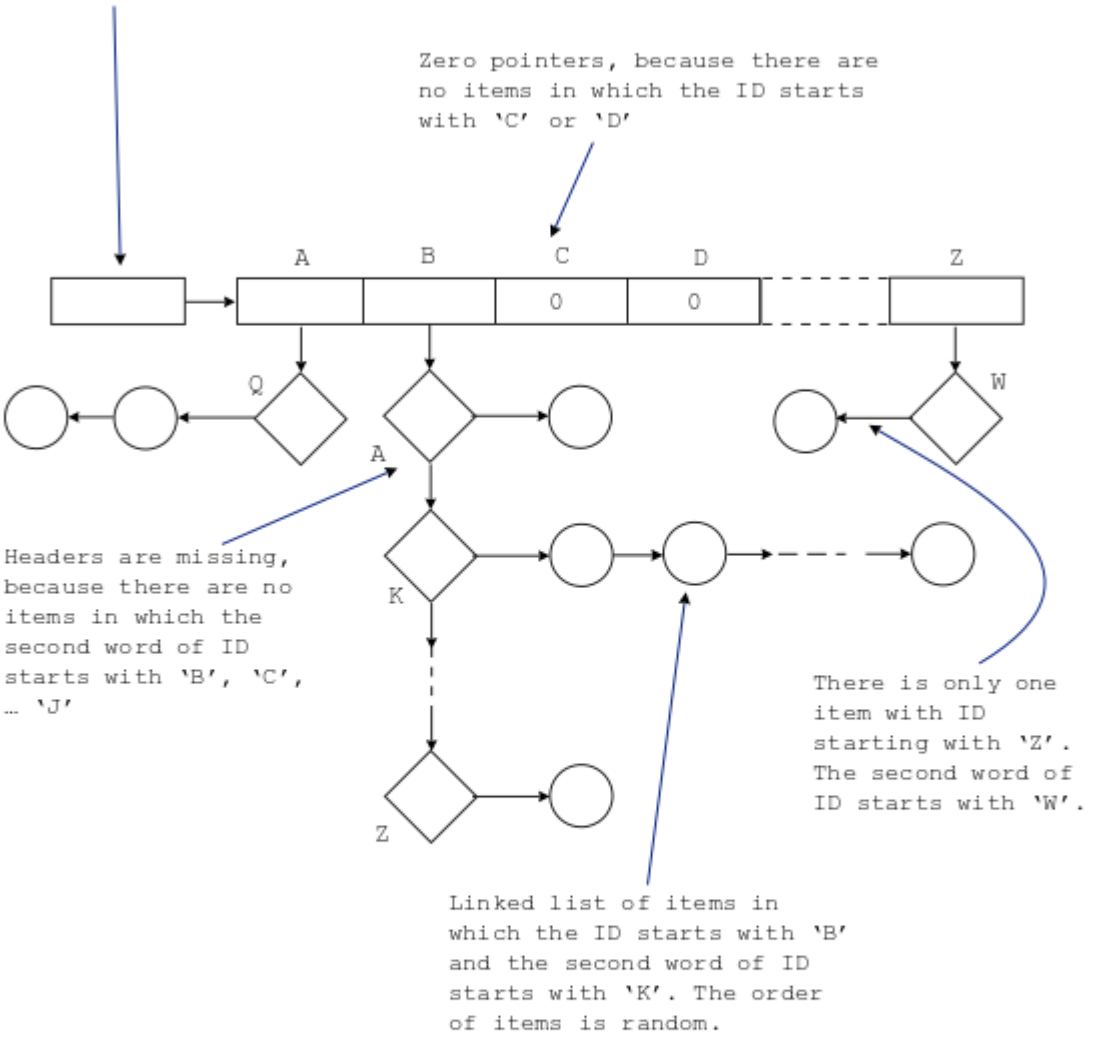
HEADER_C

Pointer

The ID consists of two words separated by space. The first letters of words are always uppercase, the rest may be uppercase or lowercase. Allowed characters are English letters and '-'. Example: "Apple Green"

Struct3 example:

HEADER_A **ppStruct3 = GetStruct3(nItemType, nItems);
The lists of HEADER_A are ordered



ITEMn (n=1...10)

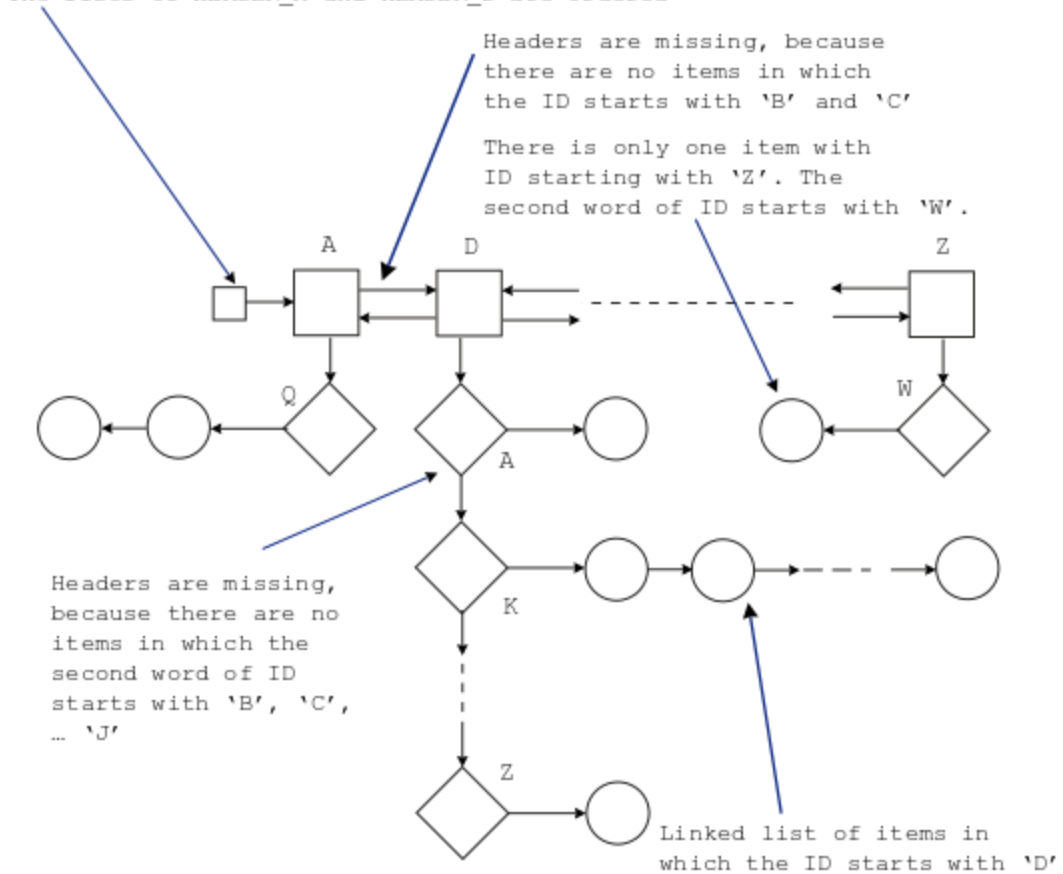
HEADER_A

Pointer

The ID consists of two words separated by space. The first letters of words are always uppercase, the rest may be uppercase or lowercase. Allowed characters are English letters and '-'. Example: "Apple Green"

Struct4 example:

HEADER_D *pStruct4 = GetStruct4(nItemType, nItems);
The lists of HEADER_A and HEADER_D are ordered



ITEMn (n=1...10)

HEADER_D

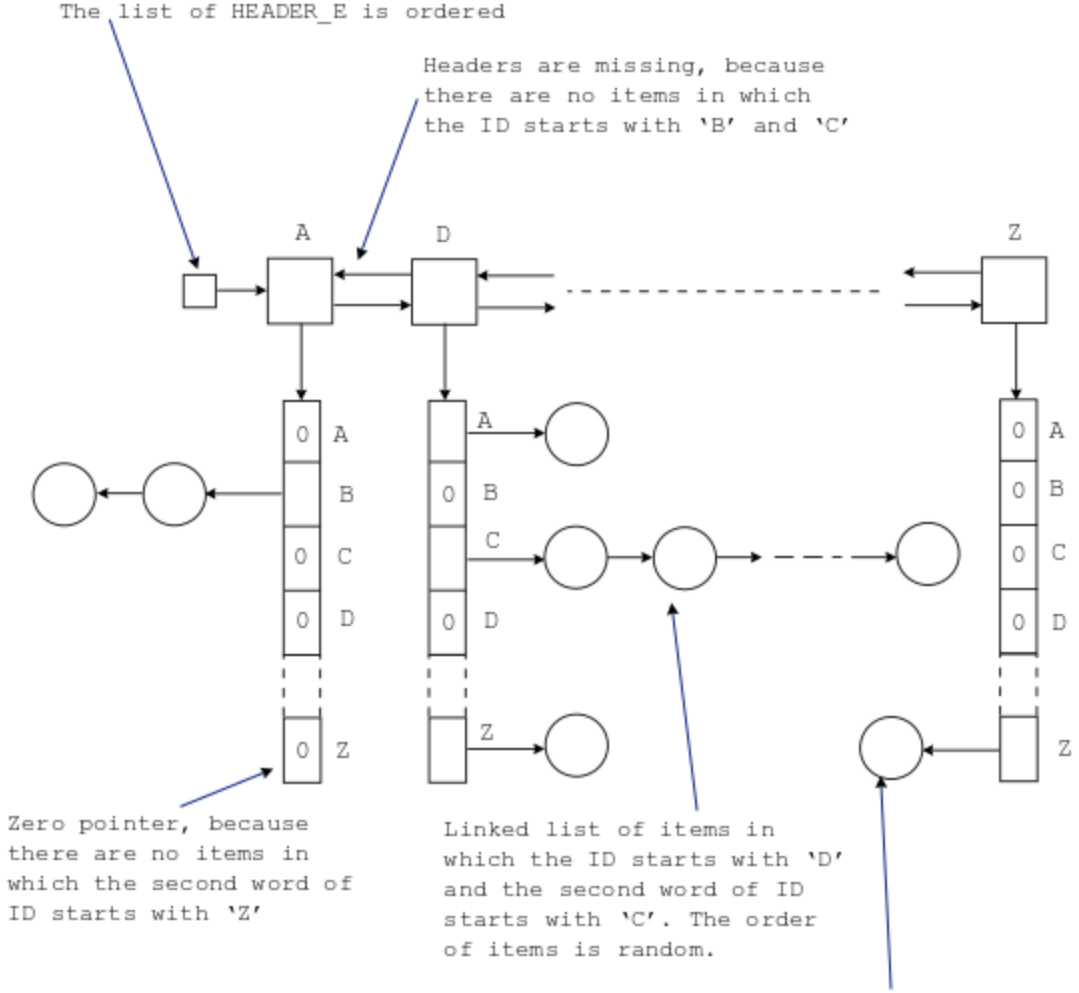
HEADER_A

The ID consists of two words separated by space. The first letters of words are always uppercase, the rest may be uppercase or lowercase. Allowed characters are English letters and '-'. Example: "Apple Green".

Pointer

Struct5 example:

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



ITEMn (n=1...10)

HEADER_E

Pointer

The ID consists of two words separated by space. The first letters of words are always uppercase, the rest may be uppercase or lowercase. Allowed characters are English letters and '-'. Example: "Apple Green"

Task

Implement class *DataStructure* containing the following members (text printed in blue depends on the type of your item (1…10), specify it yourself):

1. Depending on the number of your struct (1…5)[1]:
   *HEADER_B *pStruct = nullptr;*
   or
   *HEADER_C *pStruct = nullptr;*
   or
   *HEADER_A **ppStruct = nullptr;*
   or
   *HEADER_D *pStruct = nullptr;*
   or
   *HEADER_E *pStruct = nullptr;*
2. *DataStructure();*
   Constructor that creates empty data structure.
3. *DataStructure(int n);*
   Constructor that creates data structure of n items. n cannot exceed 100.
4. *DataStructure(std::string Filename) throw(std::exception);*
   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.
5. *~DataStructure();*
   Destructor that deletes all the items, vectors of pointers and headers.
6. *DataStructure(const DataStructure &Original);*
   Copy constructor.
7. *int GetItemsNumber();*
   Returns the current number of items in data structure.
8. *pointer_to_item GetItem(char *pID);*
   Returns pointer to item with the specified ID. If the item was not found, returns 0.
9. *void operator+=(pointer_to_Item) throw(std::exception);*
   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;*
10. *void operator-=(char *pID) throw(std::exception);*
    Operator function to remove and destroy item with the specified ID. Fails if there is no item

[1] This attribute must be private. The following functions must be public.

with the specified ID. Usage example:
**pds-= buf; // array buf contains the ID*

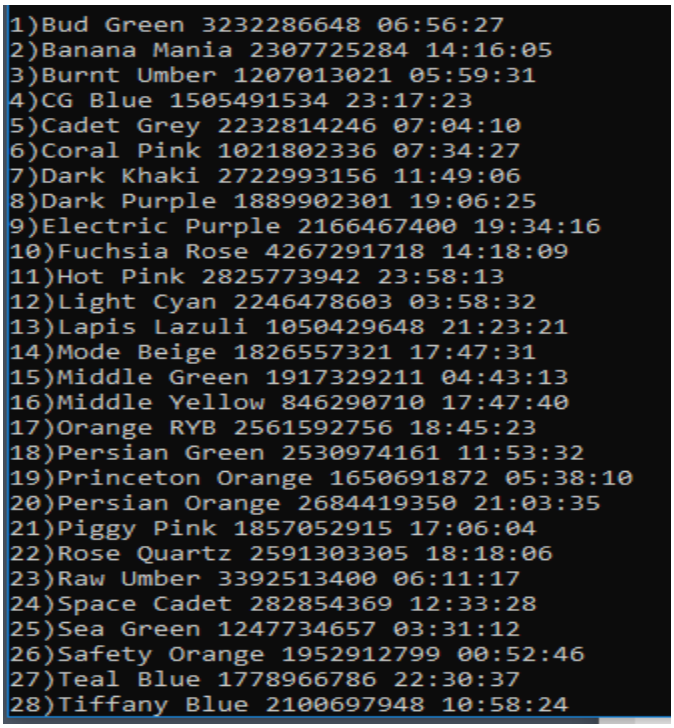
11. *DataStructure &operator=(const DataStructure &Right);*
    Operator function for assignment. Do not forget that before the assignment you have to destroy all the existing contents. Usage example:
    DataStructure ds;
    *ds = *pds;*

12. *bool operator==(DataStructure &Other);*
    Operator function for comparison. Two data structures are equal if they contain the same number of items and for each item from the first structure there is an item from the second structure so that the item IDs match. The order of items in the linked lists may be different. Returns *false* (not equal) or *true* (equal). Usage example:
    *cout << (ds == *pds) << endl;*

13. *void Write(std::string Filename) throw(std::exception);*
    Serializes the data structure and writes into the specified binary file. Fails if there occur problems with file handling or if the data structure is empty.

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

The printout should be similar to the following example:



Requirements

1. For memory allocation and release use operators *new* and *delete*.
2. The new items must be created with function *GetItem()* from *DataSource.dll.* It guarantees that the item is correct.
3. For C 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. Use *#pragma warning(disable : 4290)* to avoid surplus warnings.
8. You may add into data structure attibutes and private functions as you consider feasible. But all the attributes must be *private*.

Evaluation

The student's work is accepted if the evaluation test funcion runs correctly and produces all the supposed results. The template of evaluation test function is in file *Test.h* stored in Files for coursework #1. Usage example:

EvaluationTest<struct item5>(5, std::string("c:\\Temp\\DataStructure.bin"));

(here the item is ITEM5 and *c:\\Temp\\DataStructure.bin* is the file for string the data structure).

The deadline is the week 14 of the semester (i.e. May 5). However, it is strongly advised to present the results of coursework earlier. The students can do it after each lecture.

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

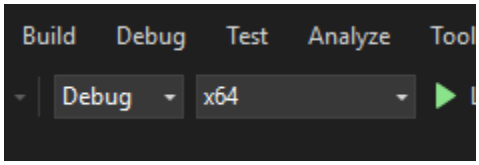
To get the assessment the students must attend personally. Electronically (e-mail, GitHub, etc.) sent courseworks are neither accepted nor reviewed. The students may be asked to explain their code or even right on the spot write a small modification.

Read also section *Hindamine* from Ülevaade.

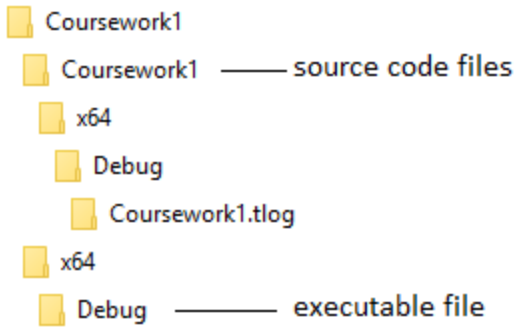
First steps

1. Launch Visual Studio and a 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:\Projects*. The wizard creates project file *C:\Projects\Coursework1.sln* and source code folder *C:\Projects\Coursework1\Coursework1*. Into source code folder it puts files *Coursework1.cpp* containing a simple *main()* function and also some auxilary files.

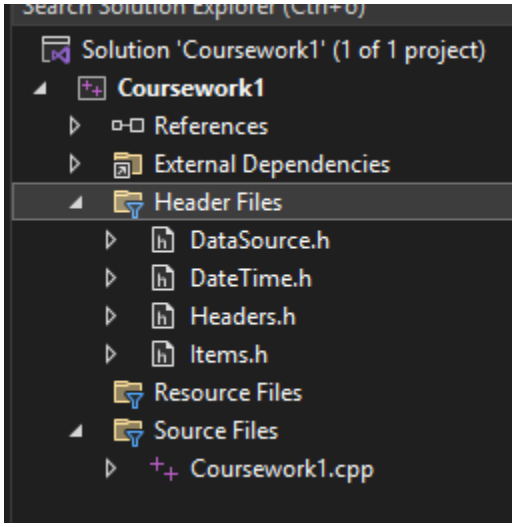
2. Check the project configuration. It must be[2]



3. Buid you solution. Now you have folders

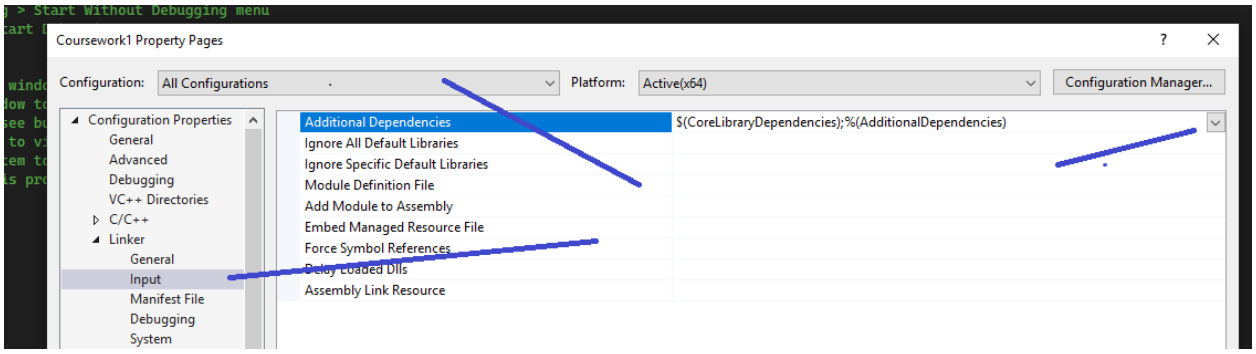


4. From Files for coursework #1 extract *DateTime.h*, *Headers.h*, *Items.h*, *DataSource.h*, *Test.h*, *Colors.txt* and *DataSource.lib*. Store them in the source code folder *C:\Projects\Coursework1\Coursework1.* In the Visual Studio *Solution Explorer* right-click Header Files and from the pop-up menu select *Add* → *Existing Item*. From the file list select all the four **.h* files and click *Add*.

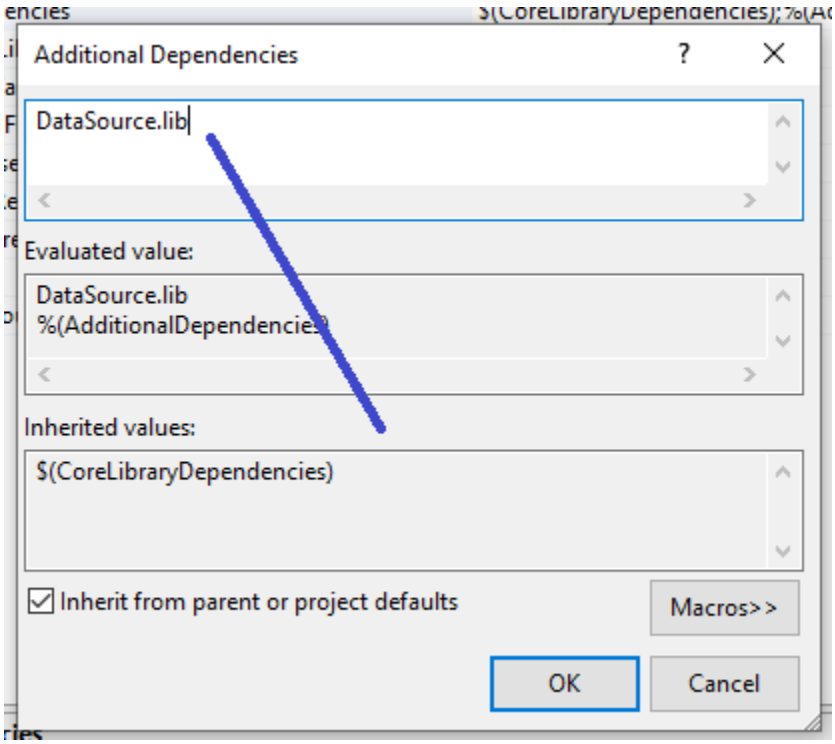


5. From Files for coursework #1 extract *DataSource.dll* and strore to the folder containing the executable *Coursework1.exe*.
6. In the solution folder right-click *Coursework1* and from pop-up menu select *Properties*. In the *Property Pages* box set configuration to *All Configurations*. Then open the *Linker properties* list, select *Input* and click on row *Additional Dependences*:

[2] We stay in debug mode. Building of the final release is not needed.



7. Click on the button at the right edge of *Additional Dependences* list. A menu opens, from it select *<Edit…>*. The *Additional Dependeces* box opens, write into it *DataSource.lib* (not **.dll*). Select *OK* and once more *OK*.



8. Now you may test is your project well prepared. The following code should run:

```
#include <iostream>
#include <exception>

#include "DateTime.h"
#include "Items.h"
#include "Headers.h"
#include "DataSource.h"

using namespace std;
// IMPORTANT: follow the given order of *.h files: DataSource.h must be the last
#define NITEM // define you item
int main()
{
    try  // uncomment you Struct
```

```
	{
		//HEADER_B* p1 = GetStruct1(NITEM,100);
		//HEADER_C* p2 = GetStruct2(NITEM, 100);
		//HEADER_A** pp3 = GetStruct3(NITEM, 100);
		//HEADER_D* p4 = GetStruct4(NITEM, 100);
		//HEADER_E* p5 = GetStruct5(NITEM, 100);
	}
	catch (exception &e)
	{
		cout << e.what() << endl;
	}
	return 0;
}
```

If the program fails to run contact the isntructor.