

Overview

In this coursework the students will create C++ software for recording and processing the results of birdwatching in Estonia.

Initial data

The results of conceivable birdwatching are generated by C++ random number generators in a DLL called the DataProvider. This DLL uses two initial data files:

- *Eesti linnud.txt* downloaded from <https://www.eoy.ee/ET/16/31/eesti-lindude-nimekiri/>
- *Eesti omavalitsused.txt* downloaded from https://et.wikipedia.org/wiki/Eesti_valdade_loend

The place where a bird was met can be specified by the community name (for example "Tallinna linn") or by GPS coordinates:

```
class GPS
{
private: tuple<int, int, int> Latitude;
        tuple<int, int, int> Longitude;
public:
    GPS() {}
    GPS(tuple<int, int, int> lat, tuple<int, int, int> lon) :
        Latitude(lat), Longitude(lon) {}

    string GetLatitude();
    string GetLongitude();
    bool operator<(const GPS& g) const;
    bool operator==(const GPS& g) const;
};
```

The latitude and longitude are presented in format DD° MM' SS" N and DD° MM' SS" E, for example 59°21'34" N 22°37'45" E.

The DataProvider DLL exports the birdwatching results as objects from class *Item*:

```
class Item
{
private:
    string Order { "" }; // selts, näiteks värvulised
    string Family { "" }; // sugukond, näiteks vintlased
    string Species { "" }; // liik, näiteks metsvint
    char Category = 0; // esinemise kategooria A, B, C, D, E (vt.
                      // originaali web-st)
    vector<tuple<variant<string, GPS>, year_month_day, hh_mm_ss<seconds>,
                optional<int>>> Observations{};
    // This is a vector of tuples:
    // get<0> specifies the location as object of type variant<string, GPS>
    // where string presents the community and object from class GPS the
    // location coordinates
    // get<1> specifies the date of observation as object of type
    // year_month_day
    // get<2> specifies the time of observation as object of type
    // hh_mm_ss<seconds>
```

```

        // get<3> specifies the number of observed birds. nullopt means that
        // the counting failed
        // The size of vector Observations may be any except 0.
public:
    Item() {}
    operator Entry() // operator function to cast an object of Item
                    // to object of Entry (see below)
    {
        return Entry(Order, Family, Species, Category, Observations);
    }
};

```

IMPORTANT: it is not allowed to change the definition of class *Item*.

The DataProvider DLL has three public functions (see also file *DataProducer.h*):

```

void Initialize(); // Reads initial data from text files and prepares
                  // the random number generators. In case of failure
                  // throws exception.

Item* GetItem(); // Generates a random birdwatching result and
                 // returns the pointer to it.

Item* GetTestItem(int i); // Returns pointer to an item prepared by
                          // instructor. Used for testing and evaluation of
                          // the coursework. i is the index of test item

```

Task #1

Implement class *GPS*.

Task #2

Implement class *Entry*:

```

class Entry
{
private:
    string Order { "" }; // As in class Item
    string Family { "" }; // As in class Item
    string Species { "" }; // As in class Item
    char Category = 0; // As in class Item
    vector<tuple<variant<string, GPS>, year_month_day, hh_mm_ss<seconds>,
                optional<int>>> Observations{}; // As in class Item

public:
    //
    // Constructors
    //
    Entry() {}
    Entry(string s1, string s2, string s3, char c,
          vector<tuple<variant<string, GPS>, year_month_day,
                      hh_mm_ss<seconds>, optional<int>>> v) :
        Order(s1), Family(s2), Species(s3), Category(c)
    {

```

```

        Observations.resize(v.size());
        ranges::copy(v, Observations.begin());
    }
    Entry(const Entry &Original);
    //
    // Accessor functions
    //
    string GetOrder();
    void SetOrder(string s);
    string GetFamily();
    void SetFamily(string s);
    string GetSpecies();
    void SetSpecies(string s);
    char GetCategory();
    void SetCategory(char c);
    vector<tuple<variant<string, GPS>, year_month_day, hh_mm_ss<seconds>,
        optional<int>>> GetObservations();
    void SetObservations(vector<tuple<variant<string, GPS>, year_month_day,
        hh_mm_ss<seconds>, optional<int>>> o);
    //
    // Operator functions
    //
    bool operator<(const Entry &) const;
    Entry& operator=(const Entry& Right);
    friend ostream& operator<<(ostream&, const Entry&); };

```

Tips and requirements:

- You can add into class *Entry* your own attributes and methods. But you may not try to derive class *Entry* from class *Item*.
- Study carefully the code of constructor above (implemented by the instructor). It will help you to write the operator functions and copy constructor.
- Do not forget that in a map the elements are ordered. Therefore *operator<()* is the most important method. To compare the *Observations* vectors you must at first compare the locations, if they are identical, the dates and so on. However, if in one of the comparable the location is specified by the string and the other by GPS skip the first stage. Do not forget that objects from class *year_month_day* are compared with the spaceship operator and the direct comparison of objects of class *hh_mm_ss* is not possible. *Nullopt* is less than any of the integers.
- The printout of an entry should be similar to the following example:

Kirjuhahk (selts: hanelised, sugukond: partlased)

1)Asukoht: 58°52'14" N 26°54'22" E

Vaatlus toimus: 02:27 12-10-2019

Nähtud 7 lindu

2)Asukoht: Muhu vald

Vaatlus toimus: 21:09 06-02-2021

Lindude arvu ei õnnestunud kokku lugeda

- Use C++ v. 20 print formatting capabilities.

Task #3

Implement class *Data*:

```
class Data
{
private:
    map<string, map<string, multimap<string, shared_ptr<Entry>>>>>
    DataStructure { };
    // It is a C++ map in which the Entry members Order are the keys.
    // The values are inner C++ maps in which the keys are Entry members
    // Family and the values are multimaps. The keys in multimaps are the
    // species of birds, the values are smart pointers to complete entries

    int nEntries = 0;
    // total number of entries in DataStructure

public:
    Data() {}
    // default constructor

    ~Data() {}
    // default destructor

    int CountAllEntries();
    // returns the total number of entries in DataStructure

    void PrintAll();
    // prints all the entries that are currently stored in the DataStructure

    void InsertEntry(Item *pI);
    // creates the entry, inserts into the data structure and releases the
    // memory of input value

    void InsertEntries(initializer_list<Item *> items);
    // creates the entries and inserts into the data structure and releases
    // the memory of input values

    void PrintEntries(string species) noexcept(false);
    // prints all the entries presenting the given bird species

    Entry* GetSummaryEntry(string species);
    // returns a new entry in which all the observations concerning the
    // given bird species are combined into one vector

    list<Entry*> GetEntries(string species);
    // returns the pointer to list containing all the entries presenting
    // the given bird species

    int CountEntries(string species);
    // returns the number of entites presenting the given bird species

    bool RemoveEntries(string species);
    // removes all the entries presenting the given bird species

    void PrintFamilyEntries(string family) noexcept(false);
```

```

// prints all the entries presenting the birds from the specified family

    int CountFamilyEntries(string family);
// returns the number of entites presenting the birds from the specified
// family

    bool RemoveFamilyEntries(string family);
// removes all the entities presenting the birds from the specified family

    void PrintOrderEntries(string order) noexcept(false);
// prints all the entries presenting the birds from the specified order

    int CountOrderEntries(string order);
// returns the number of entites presenting the birds from the specified
// order

    bool RemoveOrderEntries(string order);
// removes all the entities presenting the birds from the specified order
};

```

Requirements:

- C-style *for* loops are not allowed. Use range-based *for* loops only.
- The standard algorithms you will use must be from *std::ranges* namespace.
- In methods performing the printing or counting operations use *ranges::for_each* algorithm.
- For searching use *ranges::find_if* algorithm or *map::find* method..
- In method *GetSummaryEntry()* use insert iterators and *ranges::copy* algorithms.

Remarks:

- You can add into class *Data* your own attributes and methods.

Evaluation

The student's work is accepted if the evaluation test function runs correctly and produces all the supposed results. The tests (used for debugging as well as for evaluation) are in file *Test.cpp*, the results are in file *Test results.txt*.

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

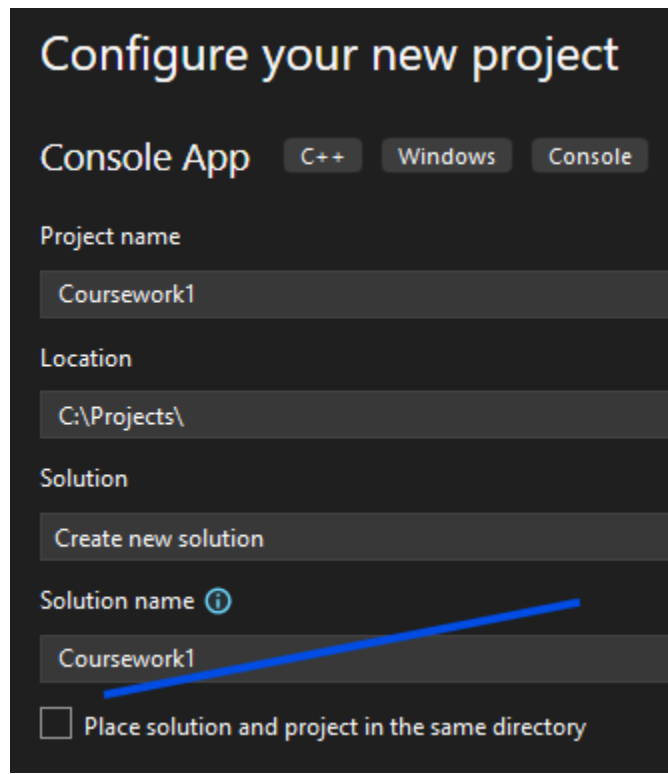
Presenting the final release is not necessary. It is OK to demonstrate the work of application in debug mode of the Visual Studio environment.

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 write a small modification right on the spot.

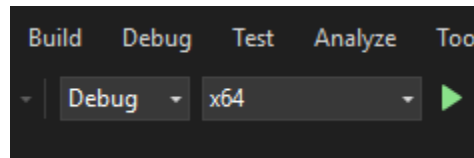
First steps

1. Launch Visual Studio and start the new project. The project template must be *Visual C++ Windows Console App*. Below we suppose that the project name you have selected is *Coursework1* and the

location folder is `C:\Projects`. Turn attention to checkbox *Place solution and project in the same directory*: uncheck it.

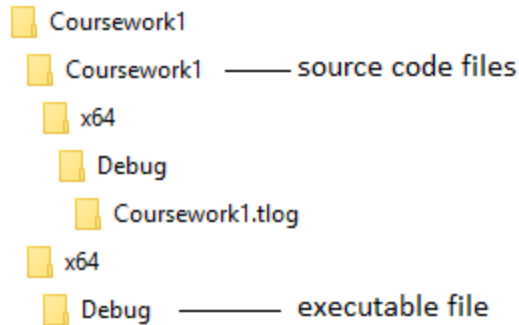


2. The wizard creates project file `C:\Projects\Coursework1.sln` and source code folder `C:\Projects\Coursework1\Coursework1`. Into source code folder it puts files `Coursework1.cpp` containing a simple `main()` function and also some auxiliary files.
3. Check the project configuration. It must be¹



4. Build you solution. Now you have folders

¹ We stay in debug mode. Building of the final release is not needed.



5. In the Visual Studio *Solution Explorer* right-click Header Files and from the pop-up menu select *Add* → *New Item*. Insert into project new header file *Main.h* and type into it the following code:

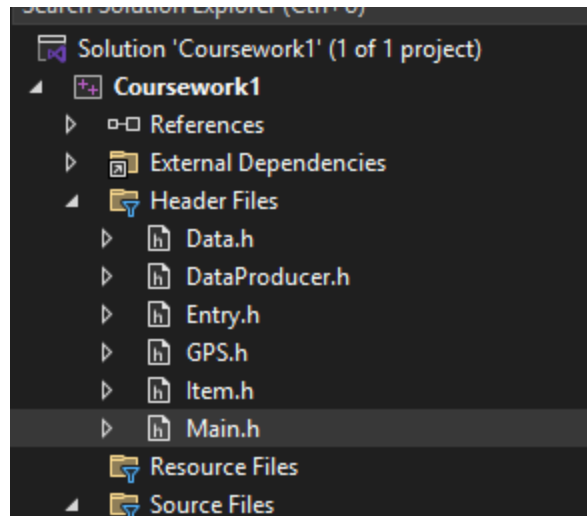
```
#pragma once
#include <iostream>
#include <tuple>
#include <variant>
#include <optional>
#include <chrono>
#include <list>
#include <map>
#include <algorithm>
#include <set>
#include <format>
#include <iterator>
#include <initializer_list>

using namespace std;
using namespace chrono;

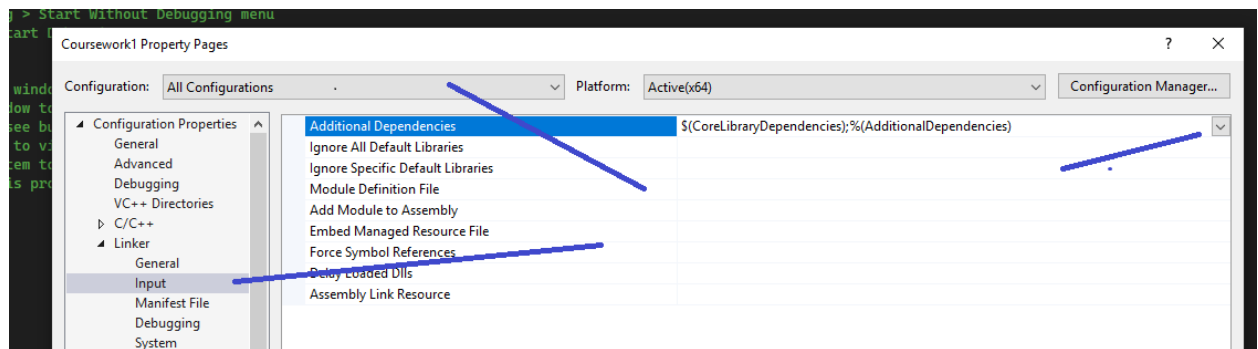
#define WIN32_LEAN_AND_MEAN
#include <Windows.h>

#include "GPS.h"
#include "Entry.h"
#include "Item.h"
#include "DataProducer.h"
#include "Data.h"
```

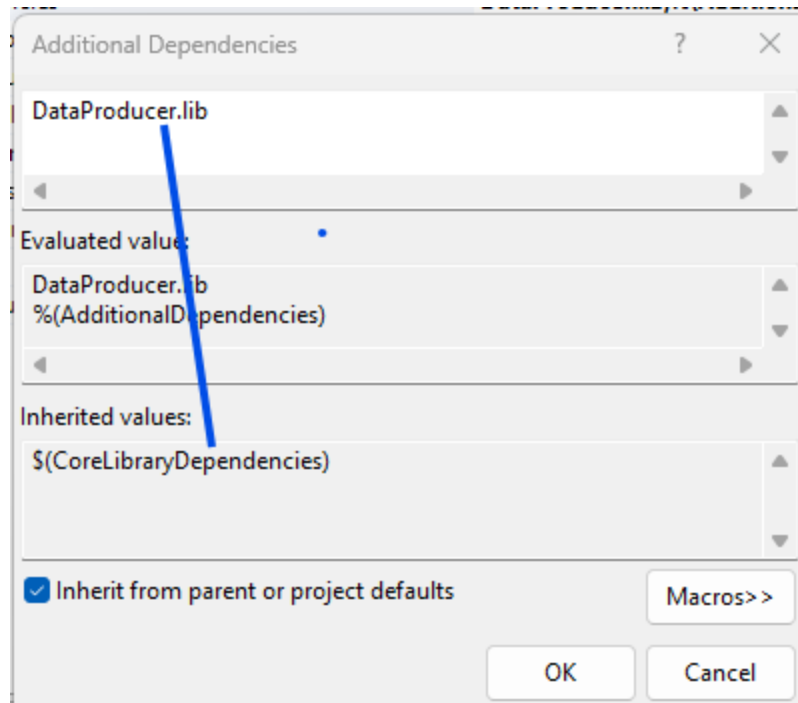
6. Using the code from initial data file *Classes.h* create files *GPS.h*, *Item.h*, *Entry.h* and *Data.h*. Store them in the source code folder *C:\Projects\Coursework1\Coursework1*. Into the same folder copy files *DataProducer.h*, *Eesti linnud.txt*, *Eesti omavalitsused.txt* and *DataProducer.lib*.
7. 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 five **.h* files and click *Add*.



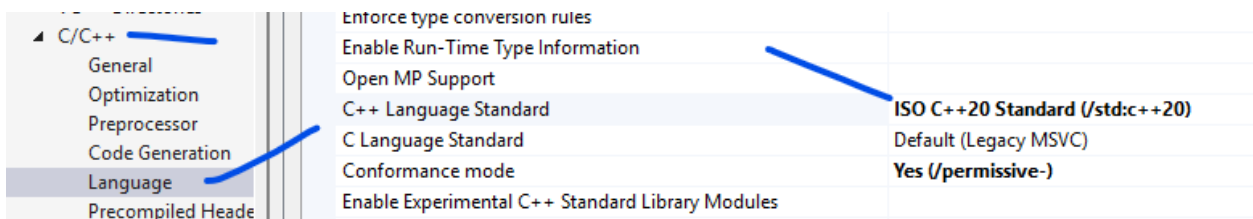
8. Copy file *DataSource.dll* into folder containing the executable *Coursework1.exe*.
9. In the solution folder right-click *Coursework1* and from pop-up menu select *Properties*. In the *Property Pages* box set configuration to *All Configurations*. Then open the *Linker properties* list, select *Input* and click on row *Additional Dependencies*:



10. Click on the button at the right edge of *Additional Dependencies* list. A menu opens, from it select *<Edit...>*. The *Additional Dependencies* box opens, write into it *DataProducer.lib* (not **.dll*). Click *OK* and then *Apply*.



11. Follow with properties. Set C++ Language Standard to C++ 20:



12. Now you may test is your project well prepared. The following code should run and print just OK without any additional error messages:

```
#include "Main.h"

int main()
{
    try
    {
        Initialize();
    }
    catch (exception& e)
    {
        cout << e.what() << endl;
    }
    cout << "OK" << endl;
    return 0;
}
```

In case of troubles contact the instructor immediately.