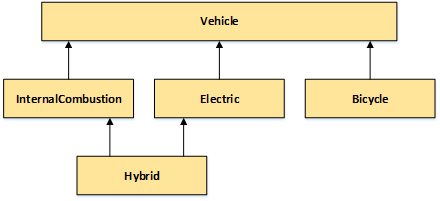
## Lab work #8

**Topics**

* More exercises about classes

**Exercises**

1. Consider the following entities:



* 1. Create classes to represent these entities, taking into account:
     1. the common attributes of all classes: year, color, number of wheels;
     2. the specific attributes of each class: cylinder volume (cubic centimeters), battery life (Km), bicycle type (“racing”, “exercise” or “mountain”).
     3. Bicycles have always 2 wheels.
  2. Electric vehicles should provide a member function **double getBatteryLife()**. On the other hand, internal combustion vehicles should provide a member function **double getCO2emission()**. For simplicity, consider that CO2 emission may be computed by multiplying the cylinder volume by 0.1 for pure internal combustion vehicles, and by 0.05 for hybrid vehicles.
  3. Consider also that vehicles are naturally sorted by year and should have a serial number assigned automatically.
  4. The classes developed should make the following program work. However, you can add any other relevant tests.

int main() {

cout << "Insertion for Bicycle:\nyear,color,type\n";

Bicycle b;

cin >> b;

cout << b << endl;

cout << "\nInsertion for Hybrid:\nyear,number of wheels,color,battery life,cylinder volume\n";

Hybrid h;

cin >> h;

cout << h << endl;

//~ Available bikes

vector<Bicycle> bikes;

bikes.push\_back(Bicycle(2012, "red", "mountain"));

bikes.push\_back(Bicycle(2005, "yellow", "racing"));

bikes.push\_back(Bicycle(2012, "red", "mountain")); //another identical bike (not the same bike)

sort(bikes.begin(), bikes.end());

cout << "\nBikes:\n";

for(auto b: bikes)

cout << b << endl;

//Hybrid vehicles

vector<Hybrid> hv;

hv.push\_back(Hybrid(2018, 4, "blue", 1800, 100));

hv.push\_back(Hybrid(2016, 4, "green", 1600, 120));

hv.push\_back(Hybrid(2016, 4, "black", 1900, 90));

sort(hv.begin(), hv.end());

cout << "\nHybrid vehicles:\n";

for(auto h: hv)

cout << h << endl;

return 0;

}

1. Consider now a vehicle rental business. Consider also that the vehicles are to be loaded from csv files (one file for each type of vehicle).
   1. Recall the function templates addressed in “Pratical class N.º 3”. Write a function template with the following signature:

template <class T>

void readVehicles(vector<T>& v, const string& fname);

This function should be able to accept as argument a vector of any class of vehicles and fill it in with the vehicles from the corresponding file whose name is the second argument. Consider as sample files:

| bikes.csv | hVehicles.csv |
| --- | --- |
| 2012,red,mountain  2010,white,exercise  2005,yellow,racing  2012,red,mountain | 2018,4,blue,1800,100  2019,4,green,1600,120  2016,4,black,1900,90  2020,3,red,900,50  2017,4,white,1500,80 |

* 1. Consider the following program and complete it according with the instructions below:

int main() {

vector<Bicycle> bikes;

vector<InternalCombustion> icVehicles;

vector<Electric> eVehicles;

vector<Hybrid> hVehicles;

//~ Reading available vehicles;

readVehicles(bikes, "bikes.csv");

readVehicles(icVehicles, "icvehicles.csv");

readVehicles(eVehicles, "eVehicles.csv");

readVehicles(hVehicles, "hVehicles.csv");

set<Vehicle\*> rented;

set<Vehicle\*> available;

//~ Fill in the available vehicles set with all vehicles loaded

//~ ...

//~ Complete here the main program

//~ ...

return 0;

}

* + 1. first, fill in the **available** vehicles set with all vehicles loaded;
    2. then, complete the program in order to manage interactively the renting and returning of vehicles, based in a menu with the following options:

1 - Rent a vehicle

2 - Return a vehicle

3 - List available vehicles

4 - List rented vehicles

0 - Exit

Menu options from 1 to 4 should have a submenu like:

1 - Bicycle

2 - InternalCombustion

3 - Electric

4 - Hybrid

Vehicles listing should be presented properly ordered.

1. Consider the files voos.txt and companhias.txt accessible on e-learning. The first one contains a list of flights that arrived at Porto Airport in a specific day. The structure of the file is the following (the information is separated by tabs):

Hora Voo Origem Atraso

00:50 TP 1944 Lisboa

07:00 AEA1147 Madrid

07:35 IB 8720 Madrid 00:25

...

The second one contains a table with the full name of the companies:

Sigla Companhia

A5 HOP!

AE Air Europa

DT TAAG

EV Evelop Airlines

EZ EasyJet Airlines

FR Ryanair

...

Implement a program to load the information about these two files into adequate data structures. You should implement at least the classes Time and Flight to represent the information about a flight. Consider the use of a Set container to store the flights. Take into consideration the treatment of all possible exceptions.

1. Print on the terminal the list of flights with the complete information about the companies, as presented next:

Hora Voo Companhia Origem Atraso Obs

00:50 TP 1944 TAP Portugal Lisboa

07:00 AEA1147 Air Europa Madrid

07:35 IB 8720 Iberia Madrid 00:25 Previsto: 8:00

07:35 TO 3408 Transavia Paris

07:40 FR 5451 Ryanair Faro

07:55 EZY3771 EasyJet Airlines Paris 00:33 Previsto: 8:28

08:05 TP 459 TAP Portugal Paris 00:13 Previsto: 8:18

08:15 FR 1386 Ryanair Brussels

08:20 FR 4171 Ryanair Frankfurt

08:30 S4 370 Sata Int Toronto 00:35 Previsto: 9:05

...

1. Save this table on a file with the name info.txt.
2. Calculate and print on the terminal the average of delays per company, sorted by the biggest delay to the smallest.
3. Save on a file with name cidades.txt a table with the information regarding the number of flights that arrived from a specific city of origin. Follow the following format:

Origem Voos

Lisboa 11

Madrid 9

Paris, Orly 8

...

1. Consider the following entities:
   1. Country: characterized by a name (String), the capital (City) and a set of districts (District).
   2. District: characterized by a name (String), the population (int) and the capital (City).
   3. City: characterized by a name (String) and the population (int).

Implement a program to represent these entities. Implement the necessary constructors, get and set methods and others that you consider to be necessary, as well as the most important operators. Take into consideration the treatment of all possible exceptions that can occur. Test the implemented classes using the program presented in the next page. The expected output should be the following:

----Iterar sobre o conjunto

Country name: Atlantis capital: Lisbon population: 999999 districts:

Country name: Aurelia capital: Wadesdah population: 23423 districts:

District name: Afrinia population: 232475 capital: Laurania population: 30000

District name: Eriador population: 100000 capital: RedRiver population: 5555

District name: South Borduria population: 84321 capital: Wadesdah population: 23423

Country name: Borduria capital: Szohod population: 31212 districts:

District name: North Borduria population: 223133 capital: Szohod population: 31212

Country name: Khemed capital: BedRock population: 234234 districts:

**//main.cpp**

#include <iostream>

#include "Country.h"

#include "City.h"

#include "District.h"

#include <iostream>

using namespace std;

int main() {

City cid1("Szohod", 31212);

City cid2("Wadesdah", 23423);

City cid3("BedRock", 234234);

District est1("North Borduria", 223133, cid1);

District est2("South Borduria", 84321, cid2);

Country p1("Borduria", est1.getCapital());

Country p2("Aurelia", est2.getCapital());

Country p3("Khemed", cid3);

Country p4("Atlantis", City("Lisbon", 999999));

p1.addDistrict(est1);

p2.addDistrict(est2);

p2.addDistrict(District("Afrinia", 232475, City("Laurania", 30000)));

p2.addDistrict(District("Eriador", 100000, City("RedRiver", 5555)));

set<Country> org;

org.insert(p1);

org.insert(p2);

org.insert(p3);

org.insert(p4);

cout << "----Iterar sobre o conjunto" << endl;

for(auto c : org)

cout << c << endl;

**// ToDo:**

**// adicionar, remover, ordenar, garantir elementos únicos**

return 0;

}

Test the remaining methods of your classes as suggested in the end of the main program.

**Programming challenge**

**PART A**

**It is intended to implement a set of classes to represent users and messages for a management system. Represent adequately all of the entities requested below, creating constructors, requested getters, setters and other member functions, as well as operators that are fundamental to each class. Using the provided main program, test the developed classes incrementally, trying to reproduce the output described at the end of this statement. (Remove the comments in the main program as you progress in your code.)**

1. Start by developing a **User** class, characterized by: a **name** (string); an **email** (string) address, and a **mobile** (string) phone number.
2. Next, develop the following classes:
   * **Msg**: a message characterized by the message body, **info** (string). This class is intended to be an abstract class, and it should provide a pure virtual member function **getType()**;
   * **EmailMsg**: a message also characterized by an **id** (unsigned int), generated automatically, starting at 1 and incremented by one unit each time a new object is created; by **srcAddr** and **dstAddr** (both string), representing respectively, the source and destination email addresses;
   * **MobileMsg**: a message also characterized by an **id** (unsigned int), generated automatically, starting at 100 and incremented by one unit each time a new object is created; by **srcPhoneNo** and **dstPhoneNo** (both string), representing respectively, the source and destination mobile phone numbers.

**Expected output for the given main file:**

Parte A

Maria +++ m@ua.pt \*\*\* +351234370500

new user (name email mobile)? Ana a@ua.pt +351910910910

Ana +++ a@ua.pt \*\*\* +351910910910

Maria +++ m@ua.pt \*\*\* +351234370500

Manuel ; mm@ua.pt ; +351234370522

first is Maria +++ m@ua.pt \*\*\* +351234370500

------------------------------------------------

Email 1 from m@ua.pt to mm@ua.pt with content: The PpO exam is today!

new email (first the content in one line, then the from and to addresses)?

I am ready for the PpO exam :-)

a@ua.pt m@ua.pt

Email 2 from a@ua.pt to m@ua.pt with content:I am ready for the PpO exam :-)

e3 id: 2

Mobile Msg100 from +351234370500 to +351234370522 with content: The PpO exam is indeed today!

new mobile (first the content in one line, then the from and to mobiles)?

The exam is about PO.

+351234370500 +351910910910

Mobile Msg101 from +351234370500 to +351910910910 with content: The exam is about PO.

m3 id: 101 and type mobile

**PART B**

**It is now intended to implement a message management system. Again, try to reproduce the output described at the end of this statement.**

1. Develop a **MsgManager** class and test it using the function, **partB()**, provided in the main program. This class is characterized by:
   * one data structure to store users: it is suggested to use a map, where the key is the user’s email address and the value is the corresponding user object;
   * another data structure to store messages that are created using dynamic memory allocation (vd. main01.cpp); the class must guarantee the release of this memory when the management system is destroyed;
   * a member function, **getEmailsFrom**, that should print the list of emails sent from a given email address (passed as parameter): each line contains the name of the recipient and the text of the corresponding message;
   * another member function, **saveOnFile**, that should save the contents of the management system in a file with the following format:

* for users: **user name email mobile**

[example: **user Ana a@ua.pt +351234370555**];

* for email messages: **email srcAddr dstAddr info**

[example: **email m@ua.pt a@ua.pt The PpO exam is today!**];

* for mobile messages: **mobile srcPhoneNo dstPhoneNo info**

[example: **mobile +351234370500 +351234370555 I love programming.**].

1. Finally, complete the function **partC()** in the main01.cpp file, in order to retrieve, into an instance of the **MsgManager** class, the contents of a management system saved in a file. Assume the same file format as specified in the previous question. If you were unable to create the file in the previous question, use the file “messages01.txt”, provided in the exam base files.

**Expected output for the given main file:**

\*\*\* Part B \*\*\*

Users List (sorted by email):

Ana +++ a@ua.pt \*\*\* +351234370555

Bernardo +++ b@ua.pt \*\*\* +351234370111

Maria +++ m@ua.pt \*\*\* +351234370500

Ze +++ z@ua.pt \*\*\* +351234234234

Message List:

The PpO exam is today!

I am ready :-)

The PpO exam is indeed today!

The content is what I studied.

I love programming.

Sent emails from a@ua.pt:

to Maria: I am ready :-)

to Ze: The content is what I studied.

\*\*\* Part C \*\*\*

Users List (sorted by email):

Ana +++ a@ua.pt \*\*\* +351234370555

Bernardo +++ b@ua.pt \*\*\* +351234370111

Maria +++ m@ua.pt \*\*\* +351234370500

Ze +++ z@ua.pt \*\*\* +351234234234

Message List:

The PpO exam is today!

I am ready :-)

The PpO exam is indeed today!

The content is what I studied.

I love programming.