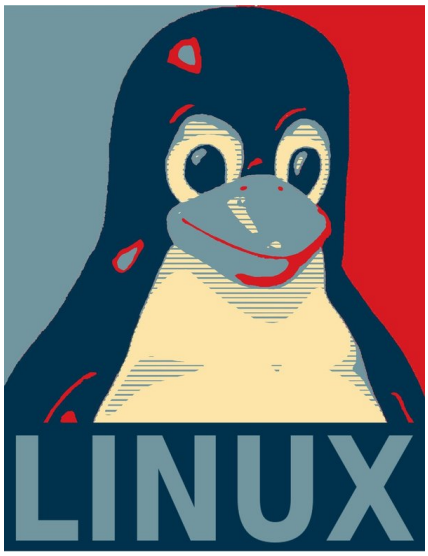
And Now C++



Introduction Challenge

Part 1

Guideline: ± 0,5 day of work

Intro Linux

This part is optional: you should be familiair with the Linux command line from NT2. If you want to refresh your memory, please go through one of these Linux/Unix tutorials:

- http://info.ee.surrey.ac.uk/Teaching/Unix/
- https://www.tutorialspoint.com/unix/index.htm

Part 2

Guideline: ± 3,5 day of work

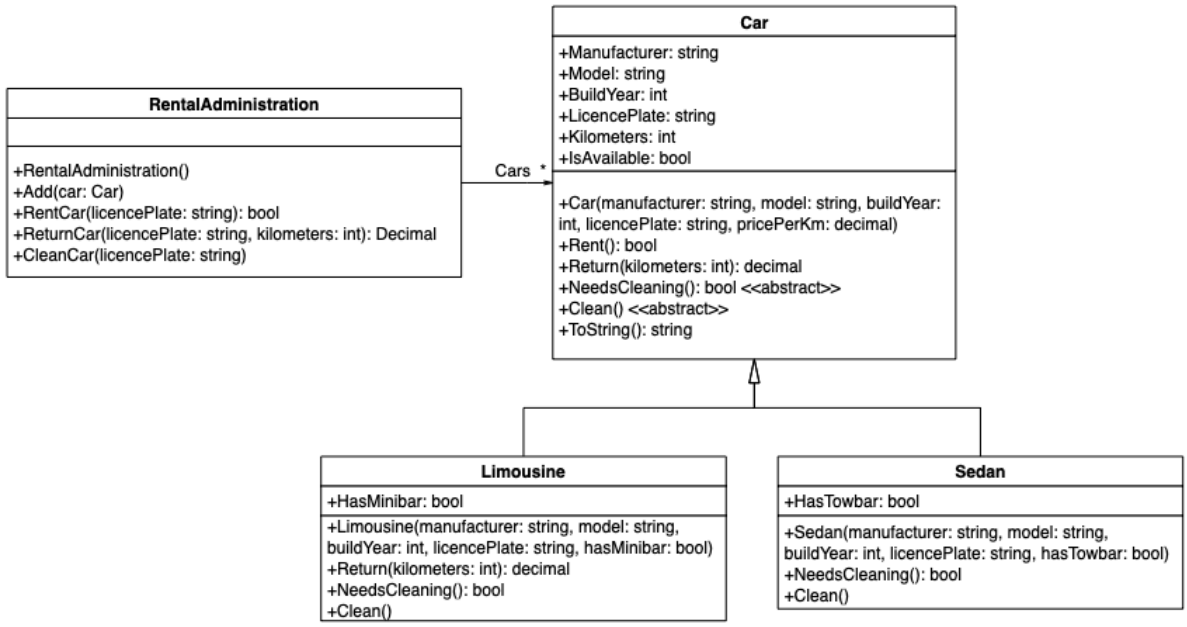
Part 2.1: And Now C++

Please take a look at the code in `AndNowC++.zip`. This zipfile contains a C# project that you are going to translate to C++. While translating the code you will run into lots of details, some of which are ignored for now. The important details are highlighted below.

This challenge is about getting familiar with C++ and `make`. You have studied both C# and C before, so the main focus is on what you already know and how that works in C++.

You probably remember the Car Rental Went Bad example from PRC2. As this example contains a lot of learning goals for PRC2 we use a slightly modified version as start point for learning C++. Please take a look at the class diagram below:

This class diagram is completely implemented in a C# project (in the `CarRental_OOP2` folder) that you can use as reference. Please take a look at the code and check if it works properly. If at some point you are in any doubt as to what the C++ code should do: take a look at what the C# code does!



You are now going to implement this project in C++. While doing this you will run into some challenges, in the description below you'll only find hints on *where* you will find challenges. Do your research well, you are expected to learn how stuff works in C++! Use the `AndNowC++_AnswerTemplate` document to answer the questions in bold below.
Hint: you will get feedback on your code, your answers and how well you've done your research.

Part 2.2a

Open `Car.cpp` and `Car.h` in the `CarRental_cpp/product` folder, you will find these files pretty much empty, except for some includes and such in the cpp file. Please open `Car.cs` as well.

A bit of explanation: `Car.h` must start with a multiple include protection, which is exactly the same as in C. **Why was that needed again (1)?**

You will also need to include `<string>`, which looks a bit weird compared to C. C++ include files are included without the `.h` extension.

```
#include <string>
```

After the include line you will need a line that says: `using namespace std`, similar to the Hello World example you have seen in class. Please ignore that for now. You will learn what this means (and why you don't want to use this) later on. For now, just remember that this is a bit of a dirty shortcut.

On to our C# port:

- the class definition in C# looks a bit different than in C++, and in C++ we don't have the `abstract` keyword. For classes this is a bit of an annoyance, as you cannot explicitly make your class abstract (a class is abstract if it contains abstract methods). So, in order to start your class definition: you'll have to figure out what the class definition in C++ looks like.
- In C#, each item in a class definition explicitly says if it's public or private. **How does this work in C++ (2)?**
- In the C# example, pricePerKm is a `Decimal`. C++ doesn't have a decimal type, **what could be a decent replacement and why (3)?**
- As you learned in PRC2: any class attribute that starts with a Capital is a `property`. As you might have guessed: C++ doesn't have `properties`. **How is this solved (4)?**
- Similar to C: in C++ we use header files and code files (.cpp). Header files only contain definitions (usually only your class definition) and code files contain implementations.
- Just like in C#, your constructor has the exact same name as your class. It does not have a return type.
- Methods that are `virtual` in C# must be virtual in C++: it means the same. **What does `virtual` mean (5)?**
- Methods that are `abstract` in C# must `abstract` in C++, however as stated before: C++ lacks an `abstract` keyword. **How is this done in C++ (6)?**
- Your `ToString` method in C# needs the `override` keyword, as you must tell the compiler that you don't want the implementation that is provided by your parent class (please remember: in C# all classes inherit from `Object`). C++ doesn't have an `override`

keyword but in this case, you would not need one: in C++ your class doesn't inherit from anything. **How does `override` work in C++ (7)?**

- A final note: in C++ it is customary (required even if you program wisely) to make any method that doesn't change your class data a `const` method. **Why is this considered "programming wisely" (8)? How do you make a method `const` (9)? Which methods should be made `const` (10)?**

  Bearing these notes in mind, please try to make your Car class definition. You should end with the public methods mentioned in the class diagram and additionally:

  - GetManufacturer
  - GetModel
  - GetBuildYear
  - GetLicencePlate
  - GetNeedsCleaning
  - GetKilometers
  - GetIsAvailable

  You should now have a complete `Car.h` header file so you are now ready to start implementing. Oh, the following implementation:

```
int GetKilometers()
{
    return kilometers;
}
```

  does not work. Your compiler will complain that kilometers is not known. **Why is that (11)?**

- The first method to implement is the Car constructor.
  Some notes:

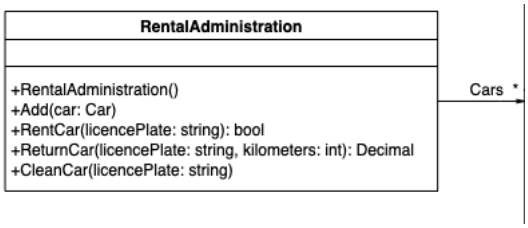
  - In the C# code you'll find exceptions for `null` references. In C++ however, same as in C, there is no such thing as reference types and value types. By default, everything is a value type. **How could you again make a parameter behave like a reference type (12)? Do you need that here (13)?**
  - In the C# code you'll also find exceptions for illegal values. **How does this work in C++ (14)? What exception would be a good choice (15)?**
  - All other functions should be rather straightforward, just be aware that `const` functions must be declared `const` in both header and implementation.

Part 2.2b

Please open `Limousine.h`, `Limousine.cpp` and `Limousine.cs`. Limousine inherits from Car, **how do you do this in C++ (16)?**

The rest of the Limousine header file should be more of the same in comparison to Car. The cpp file is more of the same as well, although calling the base constructor from the Limousine constructor works slightly different than in C#. **Why can't we just call `base`, just like in C# (17)?**

You can now implement the `Sedan` class as well.



Part 2.2c

It is now time to implement `RentalAdministration`. As you can see, `RentalAdministration` has multiple `Cars`, which is implemented in C# using a `list`. In C++ we also have a `list` type, but that is something different than a `list` in C# (**do you know why (18)?**), please use a C++ `vector` instead.

Oh, one thing: the following declaration will not work:

```
vector<Car> cars;
```

This is because Car is an `abstract` class (thus cannot be constructed). **How would you solve this (19)?** You'll need the same solution for all methods in `RentalAdministration` that need a `Car` parameter or return type.

Part 2.2d

Ok, so now our complete class diagram is implemented. Only one task left to finish: a user interface. Most of what you need here should be familiar by now, except for printing stuff to the screen and reading user input from the keyboard.

In C++ we prefer not to use `printf`. Please take a look at `cout` and `cin`. In the `main` function you will find some examples. Hint: when using `cin`: please use both lines (`cin >> yourVariable` and `cin.ignore()`)! **Why is this last line needed (20)?**

Part 3

Guideline: ± 0,5 - 1 days of work

Part 3.1: Make + variables

In the C++_1_Intro presentation you wrote a "hello world" program that consists of:

- main.cpp (calls `Print()` in `Hello` class)
- HelloLib.cpp and .h (implements `Hello` class)

Create a `Makefile` to build this project. Make sure that your `Makefile` has the following targets:

- the `default` target simply compiles everything together into 1 executable
- it has a target `obj` that compiles all cpp files into objects and links those into 1 executable
- it has a target `clean` that deletes all files that are created using this Makefile

Part 3.2: Libraries

Lots of real world applications are build only partially from source. Often we make use of libraries. In Windows we have .dll files (Dynamically Linked Libraries), the Linux equivalents are .so files (Shared Objects), both .dll files and .so files are shared libraries. **Please explain what shared libraries are (21) and why you would use them (22).**

In Linux you can also use static libraries (.a, or archive files). **Please explain what static libraries are (23) and why you would use them (24).**

Add the following targets to the `Makefile` you created in part 3.1:

- `static`, that compiles HelloLib into a static library (.a) and compiles that static library together with main.cpp into 1 executable
- `shared`, that compiles HelloLib into a shared library (.so) and compiles main.cpp so it uses this shared library
- make sure you update your `clean` target!

Please keep in mind that:

- your building process must only build that what is needed. If only 1 file has changed you must try to rebuild your executable without the need of rebuilding all files
- your Makefile uses variables to prevent multiple copies of the same information (DRY!) and uses variables to make subtle changes easy.

Part 4

Guideline: ± 1 - 1,5 days of work

Part 4.1: Makefile

Change the default Makefile for your Car Rental Went Bad project. Make sure:

- all required .cpp files are compiled into objects
- only objects of which the source is changed are rebuild
- it uses variables to prevent duplication and to enable easy customisation
- it uses wildcards where appropriate
- it has a decent `clean` target

Part 4.2: Memory Managment

In part 2 you have created a C++ version of Car Rental. Run your executable using `valgrind`, make sure you test all functions. Does your program have memory errors? If so: solve them.

Also run your program against Klocwork. Does your program has any issues? If so: solve them.

Part 4.3: Finalise your program

Make sure all required features are implemented, that memory problems and code analysis issues have been solved and that your program runs without crashes or other weird bugs.

Demonstrate

Please demonstrate:

- your Hello World project (part 3)*
- your AndNowC++ project (parts 2 and 4)*
- your filled in answer template as .pdf

*) please run `make clean` before you zip everything together for delivery in Canvas