Exercise: Introduction to C++

Within the study course ADAS we have students with very different backgrounds: computer science, engineering, electrical engineering, and related fields. Accordingly, competence levels in programming in procedural and object oriented programming (OOP) languages are quite different. However, acquiring programming knowledge in C and C++ is very important and you should learn and practice it aside your "normal" modules.

The first exercise shall help you to assess your individual programming knowledge, identify your individual deficits, i.e., show you where you should learn and practice more.

1. Set up your work environment

In the computer room T317 (and T315) you can find "Visual Studio" installed, which we will use as an IDE for this and the following exercises.

However, if you have an own notebook, I strongly advise you to use it for the exercises and bring it along. It will ease the use of additional tools, since you have no rights in T317 and T315 to install your own software. If you want to use your own notebook, you should first install Visual Studio Community. It is free.

2. A first programming task: An object list simulator

Often the result of the environmental perception process in an ADAS is an object list, i.e., some data structure that stores information about all road users currently "near" to the car.

In this first programming task, your task is to write a simple object list simulator. In each simulation step a random decision shall be made, whether we detect a new road user (a pedestrian or a car) and add it to the object list or whether we delete some randomly chosen road user from the object list. If a new road user is added to the list, store its relative position to the car and its absolute speed. Choose random values for both the relative position and the absolute speed.

Your C++ program shall output the simulation step number, what currently happened (has a new road user been added or deleted?) and which road users are currently in the object list.

An example program run could look like this:

```
Simulation step: 1
        New car detected in our perception field! (id: 1)
        List of all road users:
                Car (id 1) at (relx=22.00,-12.00) with speed 120.00 km/h
Press a key to continue.


Simulation step: 2
        New pedestrian detected in our perception field! (id: 2)
        List of all road users:
                Car (id 1) at (relx=22.00,-12.00) with speed 120.00 km/h
```

```
                Pedestrian (id 2) at (relx=18.00,46.00) with speed 5.00 km/h
Press a key to continue.


Simulation step: 3
        Pedestrian (id 2) vanished from our perception field!
        List of all road users:
                Car (id 1) at (relx=22.00,-12.00) with speed 120.00 km/h
Press a key to continue.


Simulation step: 4
        Car (id 1) vanished from our perception field!
        List of all road users:
Press a key to continue.


Simulation step: 5
        New car detected in our perception field! (id: 3)
        List of all road users:
                Car (id 3) at (relx=16.00,-26.00) with speed 28.00 km/h
Press a key to continue.


Simulation step: 6
        New car detected in our perception field! (id: 4)
        List of all road users:
                Car (id 3) at (relx=16.00,-26.00) with speed 28.00 km/h
                Car (id 4) at (relx=10.00,14.00) with speed 4.00 km/h
Press a key to continue.


Simulation step: 7
        New pedestrian detected in our perception field! (id: 5)
        List of all road users:
                Car (id 3) at (relx=16.00,-26.00) with speed 28.00 km/h
                Car (id 4) at (relx=10.00,14.00) with speed 4.00 km/h
                Pedestrian (id 5) at (relx=-1.00,-19.00) with speed 4.00 km/h
Press a key to continue.


Simulation step: 8
        Pedestrian (id 5) vanished from our perception field!
        List of all road users:
                Car (id 3) at (relx=16.00,-26.00) with speed 28.00 km/h
                Car (id 4) at (relx=10.00,14.00) with speed 4.00 km/h
Press a key to continue.


Simulation step: 9
        New car detected in our perception field! (id: 6)
        List of all road users:
                Car (id 3) at (relx=16.00,-26.00) with speed 28.00 km/h
                Car (id 4) at (relx=10.00,14.00) with speed 4.00 km/h
                Car (id 6) at (relx=-26.00,1.00) with speed 113.00 km/h
Press a key to continue.


Simulation step: 10
        Car (id 4) vanished from our perception field!
        List of all road users:
                Car (id 3) at (relx=16.00,-26.00) with speed 28.00 km/h
                Car (id 6) at (relx=-26.00,1.00) with speed 113.00 km/h
Press a key to continue.


Simulation step: 11
```

```
        Car (id 3) vanished from our perception field!
        List of all road users:
                Car (id 6) at (relx=-26.00,1.00) with speed 113.00 km/h
Press a key to continue.


Simulation step: 12
        Car (id 6) vanished from our perception field!
        List of all road users:
Press a key to continue.


Simulation step: 13
        New car detected in our perception field! (id: 7)
        List of all road users:
                Car (id 7) at (relx=49.00,-32.00) with speed 142.00 km/h
Press a key to continue.


Simulation step: 14
        Car (id 7) vanished from our perception field!
        List of all road users:
Press a key to continue.


Simulation step: 15
        New car detected in our perception field! (id: 8)
        List of all road users:
                Car (id 8) at (relx=21.00,-11.00) with speed 58.00 km/h
Press a key to continue.
```

3. Hints for implementing the simulator

- Use a class Pedestrian and a class Car to represent instances of the two different road user types

- If you generate a new instance of a Pedestrian or a Car, store the corresponding pointer to that object in a std::vector data container

- You can store all pointers in the same std::vector, if you use a base class, e.g., class RoadUser and derive class Pedestrian and class Car from this base class. Then you can store simply pointers to RoadUser instances in std::vector. This avoids the need for a separate std::vector for each road user category! Think about the situation in which you want not only to model pedestrians and cars, but also cyclists, trucks, animals, etc.

- Note: if you have severe problems with implementing this simulator, you should start with an even simpler programming task. Here you can find a large list of C and C++ programming exercises for different programming topics:

  http://www.juergenbrauer.org/teaching/programming1/programming1.html#exer_pro1

  All solutions for these small C/C++ exercises are already available at GitHub:

  https://github.com/juebrauer/Solutions-Exercises-Programming1