Exercise 1 – Navigation System (Core system)

In this exercise, you have to implement a first version of a navigation system.

In the first part of the exercise, you have to implement the class CWaypoint. Once this has been done, you will design the complete navigation system. For this, print out this document and write the answers to the questions in this document. **Note: Only handwritten answers and diagrams will be accepted.**

In the final part of the exercise, you have to implement the code for all navigation system classes. It is recommended use the following sequence:

- Create a new project in Eclipse
- Implement and test the CWaypoint class
- Create the new classes using Together
- Implement and test the classes one by one on Eclipse (provide stubs as required)

Marking:

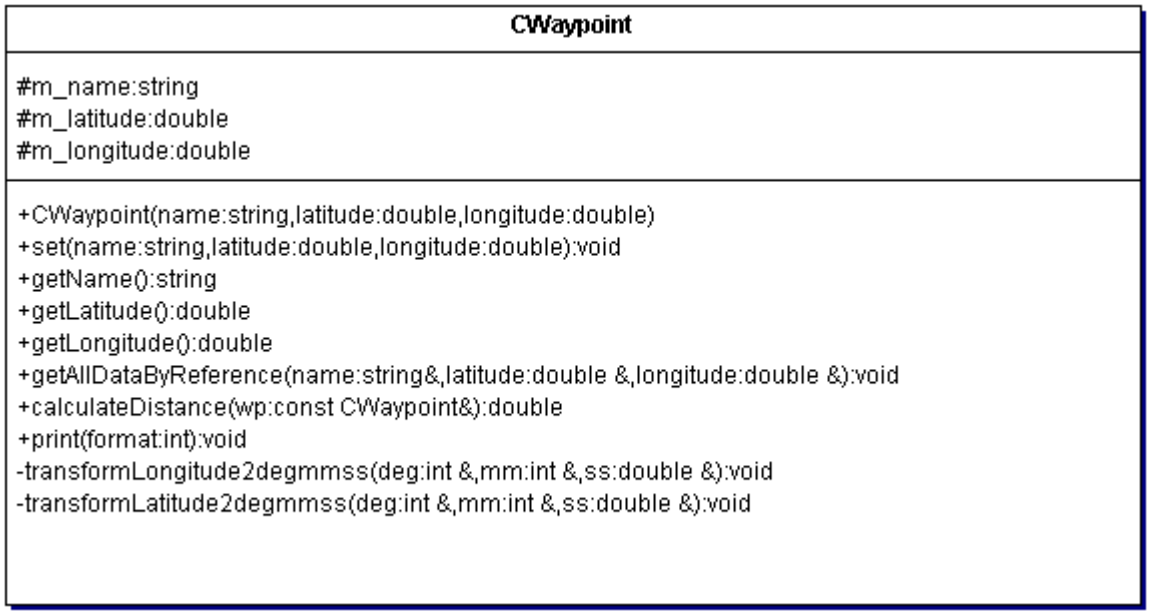
1.1	• Implementation of class CWaypoint	3
1.2	• Design and Questions	3
1.3	• Implementation of all other Navigation system classes	3

Source, Internet

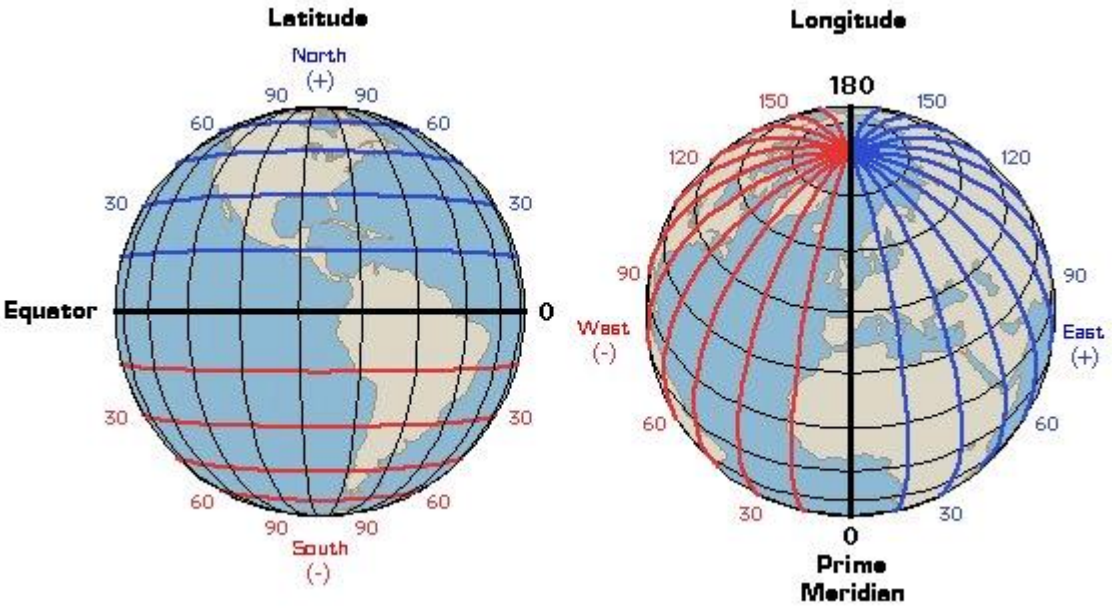
1.1 CWaypoint: A first class

Develop a class which stores a geodetic position and provides some simple operations for geodetic data.

UML Diagram



Background Polar Coordinates



$$dist = R * \cos^{-1}(\sin(latitude\_1) * \sin(latitude\_2) + \cos(latitude\_1) * \cos(latitude\_2) * \cos(longitude\_2 - longitude\_1)) \quad (1\text{-}1)$$

$$-90° \le latitude \le 90° \quad (1\text{-}2)$$
$$-180° \le longitude \le 180°$$

Hint: Earth Radius R = 6378.18km

Note: Write the answers to the questions as comments into the code

Hint: Compile the program after every task and correct the errors before continuing with the next task.

a) Declare the class based on the UML diagram in the file `CWaypoint.h`.

b) Implement the `set()` method, which checks the validity of the parameter values (see formula 1-2 above) and writes correct values to the attributes. In case of invalid parameter values, a 0 shall be written to the attributes.

c) Implement the `constructor` in the file `CWaypoint.cpp` (provide default values). Use the `set()`-method to store the parameter values. Print the address of the object using the `this` Pointer as well as all attribute values and addresses. How much data is occupied by the different attributes? Add a corresponding comment to the code! Hint: Use a the preprocessor directive `#ifdef SHOWADRESS` to turn verbose mode on and off (by `#define'ing SHOWADRESS`)

d) Create four objects `amsterdam`, `darmstadt`, `berlin` and `tokio` in the main file. Google the internet to find the latitude and longitude of these cities. Create another object `newWaypoint` without parameter list.

e) Implement the `print()` method, which takes a parameter `format`. Depending on the format value (1 or 2), the latitude and longitude are printed in decimal format or in deg mm ss format. Implement the two private functions `transformLongitude2degmmss`() and `transformLatitude2degmmss`() which translate the decimal attribute values into the deg mm ss format. Hint: Check the printout at the end of the exercise for the required format. Another hint: 60min or 3600s are equal to 1 deg.

f) Add the code for printing `berlin` in both formats to the main file. Add two constants `DEGREE` (value 1) and `MMSS` (value 2) using a `#define` statement. Which is the best file to put the define?

g) Add the three getter functions `getName`(), `getLatitude()` and `getLongitude`() to your code and provide a testcase for them in the main file.

h) Define three local variables `name`, `longitude` and `latitude` and use the method `getAllDataByReference`() to return all waypoint data of `tokio` in a single call. Analyze the addresses of the local variables name, longitude and latitude defined in the main file and compare them with the parameter addresses and values (using `cout` or the debugger). Describe your findings as comment in the code!

i) Implement the function `calculateDistance`() using the formulas above. Add a testcase to the main file calculating the distance between `amsterdam` and `berlin and berlin` and `tokio`.

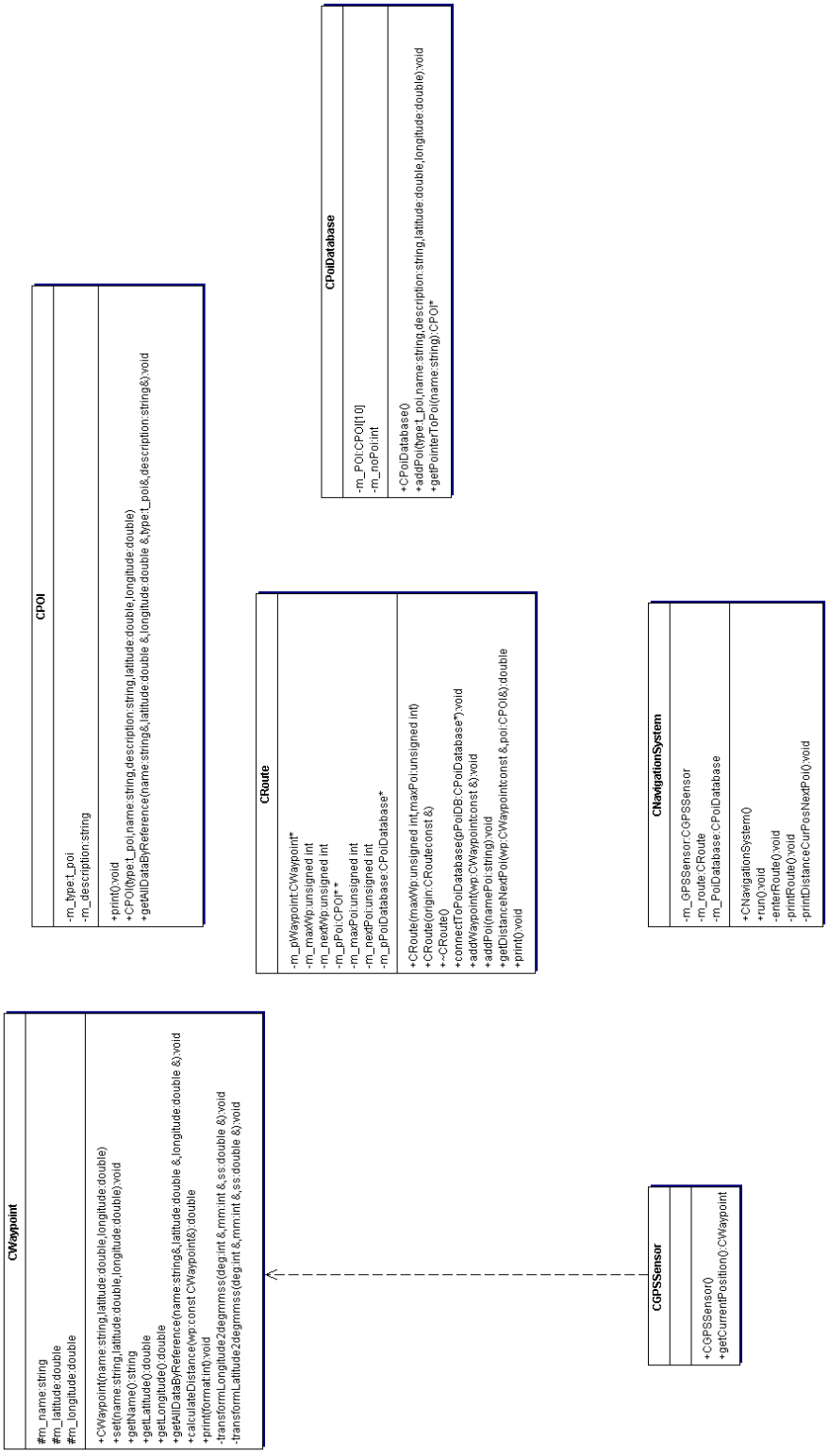
Generated example printout:

```
Constructor
===========
print method
============
Berlin  on latitude = 52.5166 and longitude = 13.4
Berlin on latitude = 52deg 30min 59.76s  and longitude = 13deg 24min 1.19904e-012s
Name as return value: Amsterdam

and a first real method
=======================
Distance between Amsterdam and Berlin: 577 (approx.)
```

1.2 Design

Add all class relations to the design. Do not forget to provide dependencies. Check also the description provided in exercise 1.3.



a) In the class CRoute, you find the following attribute declarations:
```
CWaypoint* m_pWaypoint;
CPOI** m_pPoi ;
```
Explain the different level of pointer indirection.

b) What is the idea of the method `CRoute::connectPoiDatabase (CPoiDatabase* pPoiDB = NULL)`? Who will provide the value for the parameter `pPoiDB`?

c) Why are some of the methods in `CNavigationSystem` declared private?

d) Draw an Activity diagram for the method `CNavigationSystem::printDistanceCurPosNextPoi()`. This method takes the current position from the GPS sensor, received the the shortest distance and the content of the POI from the route and, in case of a valid distance, prints the distance and poi..

e) Draw an Activity diagram for the method `addPoi(string namePoi)`. Make sure that all methods called from other classes are visible as activities.

1.3 Implementation of the complete navigation system

The functionality of the Navigation system is as follows

(1) The Navigation System contains the 3 major components GPS sensor, Route and a Point of Interest Database.

(2) A Route consists of Waypoints and Point of Interests, which are stored in the corresponding dynamic arrays `m_pWaypoint` and `m_pPoi` stored on the heap. The variables unsigned `int m_maxPoi` contains the maximum number of POIs which can be added to the route, `m_nextPoi` contains the index to the next free position in the route (same for the waypoints). Waypoints and Poi's can be added to the route using the appropriate `add` methods in CRoute. Chose the following values for `CPOI::m_type` enumerator, RESTAURANT, TOURISTIC, GASSTATION, UNIVERSITY

(3) The GPS Sensor functionality `getCurrentPosition()` is simulated by manually entering a position via the keyboard.

(4) `CNavigationSystem::enterRoute()`: Add some POI's and waypoints to your route. You may create your own trip.

(5) `CNavigationSystem::printRoute()`: Print all POI's and Waypoints of your route.

(6) `CNavigationSystem::printDistanceCurPosNextPoi()`: Get the current position from the GPS sensor and search for the closest Point of Interest.

(7) `CPoiDatabase::getPointerToPoi(string name)`: Searches for a POI with the provided name and returns a pointer to this POI.

(8) `CRoute::CRoute(unsigned int maxWp, unsigned int maxPoi)` : Creates the dynamic arrays for the waypoints and pois. The copy constructor creates a deep copy.

(9) `void CRoute::addWaypoint(CWaypoint const& wp)` : Add a waypoint to the next free position in the route

(10) `double CRoute::getDistanceNextPoi(CWaypoint const& wp, CPOI& poi)` : Calculates the distance between the current position (passed as wp) and the closest poi in the route. The poi found will be returned by the reference.

(11) The behavior of the other methods should be pretty obvious ;-)

Complete the design and answer all questions before you start coding.

f) Determine a sensible order of implementation. Which classes are you going to implement first?

g) Implement the code for all classes.

h) Provide testcases for all methods, including testcases for the exception behavior (e.g. not finding a POI in the database).

- Add the testcases to the class CNavigationSystem.
- The testcases itself shall be implemented as private functions which are called from the `CNavigationSystem::run()` Method (see class diagram)
- Add the following testcases

i. Add POI's to the POI database

ii. Create a route

iii. Print a route

iv. Calculate the closest POI for the current GPS position.

Sample output:

```
ERROR: Could not find POI Something not in the pool
Our Route has 3 Waypoints and 2 Points of Interest
Amsterdam on latitude = 52deg 22min 23.16s  and longitude = 4deg 53min 31.92s
Darmstadt on latitude = 49deg 51min 5.11583e-012s  and longitude = 8deg 39min 9.72s
Berlin on latitude = 52deg 30min 59.76s  and longitude = 13deg 24min 1.27896e-012s
Point of Interest
=================
 of type Restaurant : The best Mensa in the world
Mensa HDA on latitude = 10deg 0min 0s  and longitude = 20deg 0min 0s
Point of Interest
=================
 of type Restaurant : More expensive but also good
Sitte on latitude = 11deg 0min 0s  and longitude = 22deg 0min 0s
GPS Sensor
  Enter latitude: 15
  Enter longitude: 10
Distance to next POI = 1220.61
Point of Interest
=================
 of type Restaurant : The best Mensa in the world

Mensa HDA on latitude = 10deg 0min 0s  and longitude = 20deg 0min 0s
```