Assignment 1:Karttaretki (mapping hike)
Max Gratschew 283272

Data structures used in the assignment
My implementation has two main data structures.
1. For storing the **places**, I've used an unordered map, as place ids as key, and a Place struct as value, with members representing the data of one place's name, type and coordinates.

2. For storing the **areas**. I've used an unordered map, as area ids as key, and an Area struct as value, with members representing the data of one area's name, coordinates, truth value if a sub area or not, parent's id, and a vector of sub areas.

I decided to go with **unordered map** on both of the datastructures because not only is it logical to have a map style datastructure, it's also the best option time complexity wise in my opinion. **unordered map** offers mostly constant Big O notation for basic operations like insert, access by key and remove by key.

The assignment also had many functions that required vector of place ids or vector of area ids. I also kept up vectors of ids for both unordered maps, containing the keys of previously mentioned. There was two vectors for placeids, other one sorted alphabetically and other one by the coordinates.

For sorting I used **std::sort**, with lambda function to compare the vectors' elements in asked way. This was the slowest algorithm I've used in this assignment. I tried to bring down the performance time by using two boolean variables for both vectors sorted alphabetically or by the coordinates. after each sort the boolean values were set to true, so no unnecessary sorting would happen. When these vectors were modified, or some information of these places/areas had changed, the boolean value would've been set to false.

One extra mention about the implementation of the function **common_area_of_subareas**, it's time complexity was brought down by looping the other id's parents from a **vector** to an **unordered map**, after that the other vector was looped through and checked wether any element of the vector is equal to any key of the previously created unordered map. This way the timecomplexity was brought down nicely, since sorting of the vectors wasn't an option here, the id's must've been in the same order as the hierarchy.

Many of the functions had just to do with **unordered map's** or **vector's** own constant member functions. I think you will get a better understanding of the choices I made when you see the actual code and the short rationales for estimate in the headerfile.