Samuel Schmidt Harjoitustyö 1. Karttaretki

Basic functions that used only the main data structure

In my program I decided to use as my main data structure an unordered_map, with PlaceID as its keys and pointers to Place structs as its members. I decided this implementation because Places were often searched for using their PlaceID:s and the pointers allowed me to change the data in a certain Place without having to modify multiple data structures that I ended up using later.

Places_alphabetically and places_coord_order

When I started implementing places_alphabetically, I didn't like the fact that the data would be reorganized even though it hasn't been changed. So I created a flag for name_changed and a new vector that would store the ordered names and it would only reorganize itself if the data is changed. I did basically the same thing with places_coord_order using coord_changed flag and coord_ordered_places vector. This allows the best case performance aka. (The data hasn't been changed) to be a constant. For the implementation of places_coord_order and places_alphabetically I also decided that creating a temporary multimap would be efficient to order the keys as it is probably implemented as efficiently as possible.

Find_places_type and Find_places_name

For these functions I decided that creating a new data structure would be the best way to go. So I added unordered_multimaps with keys to PlaceType and Name and values as pointers to Places as before. This allows me to use equal_range that is an efficient algorithm to find Places with the same Name or Type. I just needed to take into account these new data structures in my functions that added or changed data in the Places

Areas

For my Area data structure I decided that a similar data structure to the Places would work. So I created a new unordered_map with keys as AreaID:s and values as pointers to Areas. The only difference to Places is that Area structs needed a parent pointer and a vector with weak pointers to children. Creating a treelike data structure that was easy to iterate through

Other stuff worth mentioning

For places_closest_to I ended up using a lot of if statements. I could have also looped through the data structure 3 times but I decided that it might be unefficient if the data structure grew very large. The code ended up a bit messier then I'd like it though.

For common_area_of_subareas I found a very handy algorithm std::mismatch that compares 2 containers and returns the iterator to the first difference. Then I only needed to create the 2 containers and reduce the iterator by 1.

For change_place_name I had to figure out how to change the name also in my unordered_multimap with Names as its keys which ended up working with std::extract.