National Research University Higher School of Economics
Faculty of Computer Science
Bachelor's Program in Data Science and Business Analytics (DSBA)

Introduction to Programming, workshops 27-28

Use the provided template that reads the Titanic data set and fills a vector with **`Passenger`** objects.

GitHub: link– problem 2.

Task 1. Custom multisort.

Part 1.

Implement an **`enum class`** data type **`PassengerField`** that represents fields of a **`Passenger`** object.

Part 2.

Implement a custom comparator **`PassengerComparator`** for **`Passenger`** objects.

This object represents one of the ways of comparing passengers. Specifically, by which field they should be compared. For example, you create an object with value Age of **`PassengerField`**, then use it in a sort, and passengers are compared by age. Then you use method **`setMode`** to change the comparison to Fare, and if you sort passengers again, they will be compared by Fare.

It should have the following features:

1. A private field **`compareField`** of the type **`PassengerField`**.
2. A constructor that takes **`PassengerField`** as input and assigns it to **`compareField`**.
3. A method **`setMode`** that changes **`compareField`** to a new field. It should take a single input variable of the type **`PassengerField`** and return nothing.
4. An overloaded **`operator()`** that implements comparison of two **`Passenger`** objects by comparing the fields corresponding to **`compareField`**.

Part 3.

Sort the vector of passengers using **`std::stable_sort`** and **`PassengerComparator`** in the following ways:

1. First by age. If age is the same, by PClass. If PClass is the same, by number of parents/children (field “Parch”).
2. First by PClass, if it’s the same – by whether a passenger survived or not. If both fields are the same – by name.

Optional task 1.

Implement a parameter of **`PassengerComparator`** allowing to sort the vector of passengers in descending order. Change the constructor and methods of **`PassengerComparator`** appropriately. Your code implementing tasks 1-3 should still work – use default parameters where needed.

Sort the vector using the first order from part 3 of the task, but this time use descending order for each field.

Task 2. Filtering.

Part 1.

Create a vector that has only passengers with Fare less than 10 who embarked at port S. Use **`std::copy_if`** and a custom function.

Passengers should be sorted by their ID.

Part 2. Other containers.

Create a set of **`Passenger`** objects using **`PassengerComparator`**. Repeat task 2 using a set instead of a vector.

Ideally, all your code should be exactly the same, with the only difference being the type of the object you use.

Part 3. Templates.

Implement a template function that does the filtering from tasks 3.2 3.3 to a container. Test it with vectors and sets.