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 23-24.

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

GitHub: https://github.com/dsba-z/week11cpp2021 – problem 3.

Replit: https://replit.com/@l8doku/Workshop23FunctionObjects

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 passengers survived or not. If both fields are the same – by name.

Additional 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.

Additional task 2.

Change field "Embarked" in the structure **`Passenger`** from **`std::string`** to **`enum class`**.

Task 2. Applying functions to collections.

1.

Round all Fare values to integer numbers using **`std::for_each`** or **`std::transform`**.

2 (additional).

Use **`std::accumulate`** to compute the total Fare of all passengers.

This function is intended to be used with values that could be summed, so you need to create a new **`Passenger`** object to store the sum and overload corresponding operators/functions treating **`Passenger`** objects as mathematical objects.

Task 3. Filtering.

1.

Output all different values for fields Parch and Sibsp. Use **`std::unique`** and a custom comparison function.

2.

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

Create a vector that has only passengers with surnames starting with a letter from A to L. Use **`std::copy_if`** and a custom function.

Passengers should be sorted by their id.

Try using **`std::inserter`** to insert elements into an empty vector.

3.

Create a vector that has only passengers with Fare less than 10 who embarked at port S, but does not contain any passengers with names starting with letters A to L. Use the previous two vectors and the function **`std::set_difference`**.

4 (additional).

Round down and sum up all Fares for passengers of PClass 3 with no siblings or spouses.

Task 4. Other containters.

Create a set of **`Passenger`** objects using **`PassengerComparator`**. Repeat task 3 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. This isn't strictly possible because **`std::remove_if`** doesn't work with sets. Change it to **`std::copy_if`** for both functions.

The problem of repeated code can be solved using templates later.

Task 5 (additional). Exceptions.

Throw an exception when data is missing during input inside type conversion functions. Handle the exception and set default values explicitly outside of **`toInt`** and similar functions.

Solve this task for Age before solving it for other fields.

Task 6. Templates.

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