

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 25-26.

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

<https://github.com/l8doku/workshops25-26>

Task 1. Custom multisort.

1.

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

2.

Implement a custom comparator **PassengerComparator** for **Passenger** objects. It should have the following features:

1. A private field **compareField** of the type **PassengerField**.
2. A constructor that takes **PassengerField** as input.
3. A method **setMode** that changes **compareField** to a new field.
4. An overloaded **operator()** that implements comparison of two **Passenger** objects by comparing the fields corresponding to **compareField**.

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.

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

Task 3. Filtering.

1.

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

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.

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.

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

Task 4. Other containers.

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 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.