Middle East Technical University Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring 2021-2022

Programming Exam 5

Due date: 20 May 2022, Friday, 23:59

1 Objectives

In the scope of this assignment you will get familiar with the constructor and copy constructor methods, and operator overloading in C++.

2 Problem Definition

The first High Speed Rail service in Italy was provided in the route of Florence to Rome in Italy with the ElettroTreno Rapido[1] (ETR) 450 in the year of 1988 (see Figure 1a). The technology behind this service has been developing since then, and an example for the state-of-art train, namely FrecciaRossa [2] 1000 is given in Figure 1b. However, like any transportation system they need a robust IT system to respond the needs of customers and to schedule the trains properly. That is why they need our help to implement such a system with basic functionalities.

(a) ETR 450

(b) FrecciaRossa

Figure 1: High Speed Trains in Italy [Wikipedia]

[1]en. Rapid Electric Train
[2]en. Red Arrow

2.1 General Specifications

- The signatures of the functions, their explanations and specifications are given in the following section. Read them carefully.
- Make sure that your implementations comply with the function signatures.
- You may define helper function(s) as you needed.
- You are not allowed to use any library beside the ones that are already provided.
- You should NOT make any change in header file. The one given to you will not be used in evaluation process.

3 HighSpeedTrain

HighSpeedTrain is the class that we are going to use in our implementation of the system. Here are the attributes of this class:

```cpp
class HighSpeedTrain
{
    private:
        std::string from;       // source of the travel
        std::string to;         // destination of the travel
        int eta;                // estimated time of arrival in minutes.

    ...
};
```

There are 2 constructors you have to implement. The first constructor is the default constructor for the default route which is the travel from Florence to Rome in 90 minutes. Other constructor will be used to initialize train with the given arguments.

```cpp
class HighSpeedTrain
{
    ...
    public:
        /**
         * Empty Constructor
         * Initializes attributes according to the default route, namely "Florence -> Rome in 90
            mins.".
         **/
        HighSpeedTrain();

        /*
         * Constructor
         * Fills the attributes according to given values.
         *
         * @param from std::string source of the travel
         * @param to std::string destination of the travel
         * @param eta int estimated time of arrival in minutes.
         */
        HighSpeedTrain(std::string from, std::string to, int eta);

    ...
};
```

Other than these constructors, there will be a copy constructor to replace a train in case of a breakdown. Since there will be need for time to get the new train there, the arrival of the train will be delayed 60 minutes.

```
class HighSpeedTrain
{
    ...

        /*
         * Copy Constructor
         * Fills the attributes according to attributes of the given object with one exception.
         * Since it requires an extra time to replace a train, the eta of the train will be
             delayed 60 mins.
         *
         * @param h HighSpeedTrain train that is going to be replaced in the route.
         * @return this new train with updated eta.
         */
        HighSpeedTrain(const HighSpeedTrain &h);

    ...
};
```

Now that we have the basic methods for the class, we can continue with the operators. The first overload will be done to summarize the route of the train. The format will be as:
`"<source> -> <destination> in # mins."` without a newline at the end.

```
class HighSpeedTrain
{
    ...

        /*
         * Stream Extraction Overload
         * Formats the output of a given train as "<source> -> <destination> in # mins."
         *
         * @param output std::ostream output stream
         * @param h HighSpeedTrain the train whose route is going to be summarized.
         */
        friend std::ostream &operator<<(std::ostream &output, const HighSpeedTrain &h);

    ...
};
```

The next overload is need to implement a possible transfer. This overload will produce a string in the form of `"<source1> -> <destination1> -> <destination2> in # mins."`. If the transfer cannot be done, namely the source of the second train and the destination of the first one differs, then this function will produce the string of `"Transfer is not possible!"`. None of these strings includes a newline at the end.

```
class HighSpeedTrain
{
    ...

        /*
         * Addition Overload
         * This operator will be used to represent transfers.
         * If the destination of second train does not match with the source of the first one,
         * then it produces the string of "Transfer is not possible!" with NO NEW LINE AT THE END.
         * Otherwise it combines the routes and calculate the total eta and produces the string in
         * the form of "<source1> -> <destination1> -> <destination2> in # mins."
         *
         * @param h1 HighSpeedTrain the first train
         * @param h2 HighSpeedTrain the second train
         * @return one of the strings as explained above.
         */
        friend std::string operator+(const HighSpeedTrain &h1, const HighSpeedTrain &h2);

    ...
};
```

The final overloads will be done for the comparison of the trains to summarize the current status of the overall system in an order. The order will be alphabetical according to sources. In case of a equality of the sources, the next thing we will look for is the alphabetical order of the destination. Finally, the train with less estimated time of arrival will be prior in the increasing order, if the destinations are also the same.

```
class HighSpeedTrain
{
    ...

        /*
         * Comparison Overloads
         * This operator will be used to order the summaries of train.
         * The priority for comparison is here:
         * 1. Compare the source of the trains alphabetically (ex. Florence < Milano)
         * 2. Compare the destination of the trains alphabetically (the same example above)
         * 3. Compare the eta's of the trains, the train with less eta will be smaller.
         *
         * @param h1, h2 HighSpeedTrain the trains to be compared.
         */
        friend bool operator<(const HighSpeedTrain &h1, const HighSpeedTrain &h2);
        friend bool operator>(const HighSpeedTrain &h1, const HighSpeedTrain &h2);
};
```

4 Regulations

1. **Implementation and Submission:** The template file named “highSpeedTrain.cpp”, the header file named “highSpeedTrain.h” and sample test files are available in the Virtual Programming Lab (VPL) activity called “PE5” on `OdtuClass`. At this point, you have two options:

   - You can download the template file, complete the implementation and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
   - You can directly use the editor of VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submit a file.

   The second one is recommended. However, if you're more comfortable with working on your local machine, feel free to do it. Just make sure that your implementation can be compiled and tested in the VPL environment after you submit it.

   There is no limitation on online trials or submitted files through OdtuClass. The last one you submitted will be graded.

2. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.

3. **Evaluation:** Your program will be evaluated automatically using “black-box” technique so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to consider the invalid expressions.

   Important Note: The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official testcases to determine your ***actual*** grade after the deadline.