Mittuniversitetet
MID SWEDEN UNIVERSITY

PROJECT
Trains

Object-oriented programming in C++ (DT060G) VT term 2019

General goals:

In this project, you are supposed to show that you can

- structure and solve a more complex problem
- apply basic object oriented analysis to identify classes and their operations
- apply principals of object-oriented programming in a relatively big program
- learn and implement a new concept independently
- use dynamic binding
- use dynamic memory and smart pointers from C++11
- use generic algorithms from the standard library
- create and use various types of callable objects
- use exceptions
- avoid unnecessary duplication of code
- Document and present your work in a report.

Background

The international railway company *Ironbend Train and Brain Railway Inc*. has presented a consulting assignment to solve an urgent IT problem. The word around the company is that there is a total chaos when it comes to their rail vehicles. It is pure luck if some train happens to leave on time and for the right destination, let alone some vehicles to assemble at train are at all available. Nobody has a clear view of the logistics of the vehicles and the whole company is at jeopardy. In the long term, the company will need tools for administration of their vehicle fleet and trains but the goal for our assignment is to construct a prototype for an application to simulate the operations during a 24-hours period. Given the initial data for stations, trains, vehicle and the timetable, you are to simulate the assembly of trains at the departure stations, the running of the trains and their disassembly at the destination stations.
The expected outcome will be information on which trains could reach their destinations on time, which trains were delayed etc.

The task – description and specification

You are to construct the simulation prototype that the *Ironbend Train and Brain Railway Inc* so badly needs.

Phase 1 – The vehicle hierarchy

The first phase in the project covers the creation of an object-oriented model that encompasses trains and vehicles. The model is based on a set of classes and relations between these classes. These classes should then be implemented and tested.

The basis of the model is the following facts and descriptions:

1. There are two kinds of engines: diesel engines and electrical engines.

2. There are two kinds of passenger-carrying car: coach and sleeping car.

3. There are two kinds of freight cars: open freight car and covered freight car.

4. Each train has at least one engine and a number of cars. A train can have more than one type of engines and cars.

5. Every vehicle has a unique id-number.

6. Every train connection has its own *train number*. The train number is a logical concept that relates to a communication between two stations at a certain point of time. In this concept lie
   - type(s) of engine(s),
   - number, types and order of cars
   - departure station
   - destination station
   - departure time
   - time of arrival at the destination

   E.g. train 859 departs from South Bend central at 16.13 and arrives in Tahoma City 19.43 every day and it always consists of a diesel engine followed by 3 coaches and 2 covered freight cars. On the other hand, *the train number does not say anything about exactly which specific vehicles (identified by their unique id) are assembled to make up the train. This can vary depending on which vehicles are available at the station on the occasion at hand.*

1. Every train passes through a series of states throughout its life cycle. These states are

   - *NOT ASSEMBLED*
     The train exists only as a logical concept (see point 6 above). No vehicles are connected.
   - *INCOMPLETE*

The train is partly assembled but one or more vehicles are missing on the station.

- *ASSEMBLED*
  *T*he train is successfully assembled
- *READY*
  The train is complete and ready to leave.
- *RUNNING*
  The train has left the departure station heading for its destination.
- *ARRIVED*
  The train has arrived to its destination station.
- *FINISHED*
  The train has been disassembled at its destination station and put in the pool of available vehicles at that station. The vehicles are available for further use in other trains.

For grades A and B, all these states will be handled. For lower grades, delays due to the INCOMPLETE state need not be handled. More about this in the Phase 2 - Simulation Application and Assessment section.

8. A train is assembled from available vehicles at the departure station. For each train the user should always be able to get information about
   - train number
   - departure station and time
   - destination station and time
   - current state
   - type of every vehicle in the right order (engine first)
   - if the train is fully or partially assembled the information for each vehicle should be extended with its id and other attributes (see 9. below)
   - average speed in km/h calculated from departure time, destination time and distance

9. The following attributes of the different vehicle types are part of the model:
   - coach - number of chairs, Internet yes/no
   - sleeping car - number of beds
   - open freight car – cargo capacity in tons and floor area in square meters
   - covered freight car – cargo capacity in cubic meters
   - electrical engine - max speed in km/h and max power i kW.
   - diesel engine - max speed in km/h and fuel consumption in liters/h

Start by creating the vehicle hierarchy. Use virtual functions to get and write the required information to the screen.

Phase 2 – The simulation

The second phase in the project is the development of the simulation application. The function of this prototype is described as follows. NB. that variations and extensions for

different grades are discussed in the paragraph Grading.

- The running of a number of trains are simulated from one point in time to another point in time during the same day starting at 00:00 and ending at 23:59.
- In the simplest way the simulation time is advanced in 10-minutes intervals by the user striking a key.
- For each time interval the current simulation time is output as hh:mm to the screen. All changes in the states of the trains that have occurred during the last interval are also output to allow the user to monitor the chain of events. If no state changes have occurred, only the current time is output.
- All events (state transitions) according to the above paragraph must also be logged to a file named Trainsim.log.
- As an alternative to advancing the time the user should always be able to get information about
  1. *the time table*: trains with train numbers, stations and times
  2. *a chosen train*: train number, state, stations, times, vehicles
  3. *a chosen station*: trains at the stations with states and vehicles, available vehicle in the pool

  Information about a vehicle should always include id and type.
- Statistics is shown after completed simulation. This includes
  - which trains were successfully run without delays
  - which trains were delayed and how much
  - which trains never left their departure stations and why
  - the accumulated delays in minutes

Each station has a pool of vehicles available to assemble trains. The content of the pool will change in the course of the simulation as departing trains are assembled and arriving trains are disassembled.

The life cycle of a train can be described by the states and the conditions for state changes.

- A train starts in the state NOT ASSEMBLED

- An attempt to assemble the train is made 30 minutes before its departure time. If there are more than one vehicles of the requested type available, the one with the *lowest id* should be chosen. If it is successfully assembled according to the specification its state is changed to ASSEMBLED. If, on the other hand, at least one vehicle is missing its state is changed to INCOMPLETE. New attempts to assemble the train shall then be made every 10 minutes until it succeeds and the state is changed to ASSEMBLED, or the simulation is ended. For every failure to assemble the train, 10 minutes shall be added to its departure time and time for arrival.

- The state of a train is changed from ASSEMBLED to READY 10 minute before its current departure time.

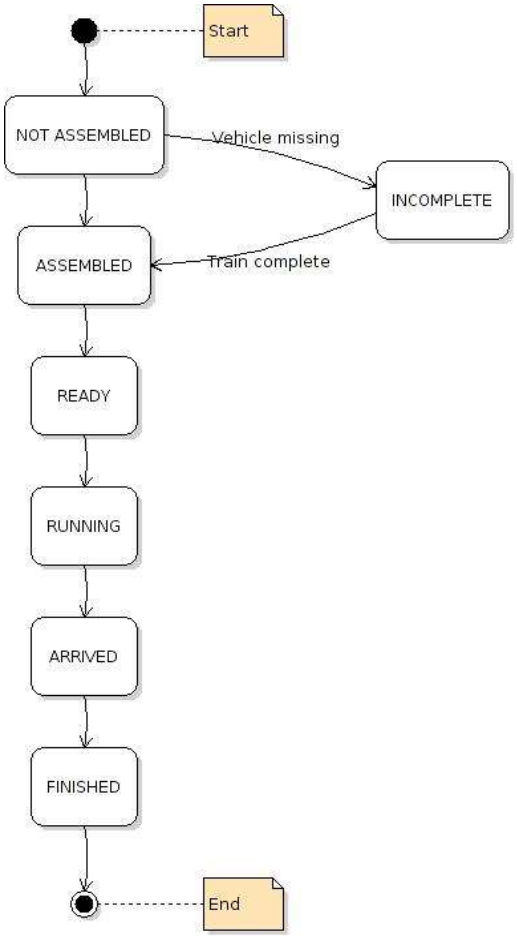
- On departure the state is changed to RUNNING.

- On arrival to its destination the state of the train is changed to ARRIVED.

- 20 minutes after arrival the train is disassembled and its vehicles are placed in the vehicle pool of the station. The train has now reached its final state which is

FINISHED.

The possible state changes can be depicted in an UML state diagram:



If a train comes into the state INCOMPLETE, this means that it will be delayed.
For grades A and B, new attempts to complete the train must be made every 10 minutes until the train is complete and can enter to state ASSEMBLED, or the simulation is complete. For each failure to make the train complete, the times of departure and arrival to be delayed by 10 minutes. The consequences of this is will be, some trains will arrive late to their destination and that some trains will never leave their destination.
For grades C, D and E do not need any new attempt to complete the train . The consequence is that a train, which ended up in the INCOMPLETE state, will never leave the departure station.

The simulation data, ie. All data concerning trains, stations, vehicles are read from the three provided data files at program start-up time. These files are described in the Data files section.

Information to be read is

- the vehicles in the system, their id, type and attributes
- the stations in the system and which vehicles there are in their respective vehicle pool

- the trains with train number, departure station / time, destination station / time, number and types of vehicles, max speed and distance

The program must only know the names of these data files, no other simulation data must be hard-coded. On reading these files, the corresponding objects should be created dynamically.

The simulation spans one day and start at 00:00. In an ideal world without delays all trains have reached their destinations no later than 23:59. If a train due to a delay has started but not reached its destination at 23:59 the simulation should continue until the train has reached the FINISHED state. Put another way round: the simulation stops when

- all trains have reached the FINISHED state
  OR
- time has passed 23:59 and no train is in the RUNNING or ARRIVED state.

NB. if the simulation continues after 23:59 no trains are allowed to be assembled or started as we are only interested in trains that are departed during the day in question.

The obtained results are presented when the simulation is finished. This includes information on:

- which trains eached their destinations on time
- which trains were delayed (and why)
- which trains could never leave their departure stations (and why)
- total delay

Data files

Simulation data is read from the files: TrainMap.txt, TrainStations.txt och Trains.txt.

TrainMap.txt

This file names the stations and the distance between them in km. Each row has three fields:
`stationName stationName2 distance`
The distances are symmetrical and are mentioned only once, i.e. the distance from a to b is the same as the distance from b to a.

TrainStations.txt

This file tells which vehicles are at which station at start-up time. Each row holds data for one station in the format

```
StationName (id type param0 param1)(id type param0 param1)...
```

Each vehicle is coded as `(id type param0 param1)` where `id` is the unique id, `type` is the type coded as an integer 0 – 5 with attribute parameters as follows:

caoch = 0;

`param0` = number of chairs : int
`param1` = Internet yes/no : int (1 = true / 0 = false)

sleeping car = 1;
`param0` = number of beds : int

open freight car = 2;
`param0` = cargo capacity in tons : int
`param1` = floor area in square meters: int

covered fright car = 3;
`param0` = cargo capacity in cubic meters : int

electrical engine = 4;
`param0` = max speed in km/h : int
`param1` = power kW : int

diesel engine = 5;
`param0` = max speed in km/h : int
`param1` = fuel consumption in liters/h : int

Trains.txt

This file describes the trains. Each row contains data for one train as follows:

```
trainId from to departTime arrivTime maxSpeed type type ...
```

`trainId` is the train number followed by departure station, destination, departure time, arrival time and max speed. Thereafter follows an enumeration of the types of the vehicles that make up the train, coded as integers 0 – 5. The max speed is given as an integer but should be treated as a `double` in calculations.

Two sets of datafiles are provided, one for Windows and one for UNIX/Linux/OSX

Requirements for the solution

- The solution shall meet the description and specification as stated above and thereby produce a correct simulation result. Using the id for a chosen vehicle it should be possible for the user to track a chosen vehicle from one station via a train to another station.

- Favor C++11 features over old ones if you have a choice. Do not use deprecated language elements.

- You must not use platform specific code or code that depends on some specific compiler, e.g `#pragma once` or `#include<conio.h>`. However, you may use conditional compilation with `#ifdefined .. #endif`.

- No memory leaks!

- Return values should be checked for error conditions and exception should be caught. No runtime crashes due to erroneous input.

- Your code should be commented in a way that contributes to an understanding of what goes on. Example of a non-comment: `++i  // increase i by one`

Let classes and functions begin with a short description.
Functions should begin with an explanation of used parameters and return value (if any).

- A project report that describes your solution should be provided

Assessment

Your solution will be assessed with one of the grades A, B, C, D, E, Fx or F.
Fx means that you are given a chance to complete the solution to be passed with grade E.

Grounds for assessment are

- *Goal achievement*
  Does your solution comply with the description and with the requirements?

- *Structure*
  Division of the problem domain into classes and their functions. Assignment of responsibilities among classes and functions..

- *Usage of the language*
  To what extent does your solution benefit from language features like smart pointers, generic algorithms, callable objects, exception, dynamic binding etc?

- *Source code*
  Consequent typography and indentation. Descriptive identifier names and meaningful commenting.

- *User friendliness*
  Informative and easy-to-use interface.

- *Project report*
  Structure, layout and content.

In every respect that your solution clearly deviates from the given description and requirements it will render a notation. The number of notations will affect the grading of your work, see below.

Grading

A requirement to be passed (grade E) is a working program. The program should produce the correct results from the given data. Moreover, it must comply with the description given under Phase 2 – The simulation and must have rendered no more than 5 notations.

Extensions for higher grades

The specification can be extended with

a) User option for choosing start and end time of the simulation.
b) User option for changing the interval length.
c) User option for bypassing the fixed interval length and let the simulation run until the next state change has occurred.
d) User option for choosing different levels of details about the vehicle. Minimum is id + type
e) User option for getting the current location of a vehicle given its id.
f) User option for getting a history of the movements of a chosen vehicle so far at any point in the simulation.
g) Implementation of calculated average speed of the trains. Let the average speed be affected by delays. Implement that a train increases its speed in proportion to its delay with the restrictions that it can never exceed its maximum speed and it may never reach its destination before the original arrival time. This means that a delayed train will get 10 minutes added to its departure time for each failed assembly attempt but the arrival time is recalculated.

For grade D: implementation of a) and b) and a maximum of 4 notations.
For grade C: implementation of a) .. e) and a maximum of 3 notations.
For grades A and B, new attempts to complete the train must be made according to the previous deception of incomplete state.
For grade B: implementation of a) .. f) and a maximum of 2 notations.
For grade A: implementation of a) .. g) and a maximum of 1 notation.

A delayed delivery of the solution will lower the grade by one step.

Dicrete Event-Driven Simulation
This project is an example of a discrete event-driven simulation where all events occur at discrete points in time and where the time in between the events can be skipped. A simple framework for this kind of simulation is provided for you to adapt and use if you so wish (Exempel-DEDS.zip).

Report

Write a project report containing...

- a first side which names of the course, the project and yourself
- a summary of the task

- an account of the used platform and your development tools
- a description of your solution. You don't have to go into details about the vehicle hierarchy. Focus on the second phase of the solution and how you implemented the simulation. What classes have you made, their responsibilities and co-operations. You can use UML-diagrams to illustrate this.
- Your own conclusions and comments. I welcome your opinion of the course in general and this project in particular in order to make future improvements.
- Please clearly mention that how many tasks have you **successfully** completed and which grade are you expecting, you can motivate your statement by describing your test program using screenshots.

Submission

The compressed file you hand in should contain
- source code (and build instructions if needed)
- the data files used
- the report in pdf format.

When submitting your solutions you implicitly assure that they are entirely your own and that nothing has been copied from other students. If inappropriate co-operation is at hand it is reported as a disciplinary offense which may lead to a suspension from your studies at this university.

Good luck!
Awais