CS 161 Assignment #6 (Optional) – Train Journey

Assignment due: Saturday, 3/14/2020, 11:59 p.m. (TEACH)

Important Notes:

This optional assignment is worth 80 points. The lowest score of all six assignments will be dropped. Therefore, this assignment can make up for a missed previous assignment (to a maximum of 80 points).

Goals:

- Gain experience working with recursive data structures
- Manage dynamic memory: no memory leaks or dangling pointers

---

Part 0. No Demonstration for Assignment 6

Instead of a demo, you will write a README.txt file (see Part 3 for details). Your assignment will be graded offline, using the README as a guide for how to compile, run, and interact with the program.

Submissions that do not compile on the ENGR servers will receive a 0. Please test your code before submitting! Comment out parts that do not work if necessary.

Part 1. Design Your Updates to an Existing Train Structure

In this assignment, you will start with an implementation of trains using the `train_car` struct:

```
/* Structure defining a train car */
struct train_car {
  string kind; /* Engine, regular car (_***_), or Caboose */
  train_car* next_car; /* pointer to the next train car */
};
```

A train is a recursively linked set of `train_car` structs such that the first one is an Engine and the last one is a Caboose.

Your first step is to familiarize yourself with this code. Download and run it a few times to gain familiarity.
https://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/assign6_trains.cpp

Your goal is to add these extensions to the program:

- (5 pts) Write a **recursive** function that takes in a `train_car*` and returns the **length of the train** (number of cars), including the Engine and Caboose.
- (5 pts) Extend the struct definition to have member variables to track the **number of seats** in each car and the **number of passengers seated** (which will be <= the number of seats). Include comments to justify your choice of data types for these members.
- (5 pts) Update `add_cars()` to initialize these new member variables along with `kind` and `next_car`. The number of seats should be randomly chosen from 10 to 20 (inclusive), except that passengers are not allowed in the Engine or Caboose, so their number of seats is always 0. The number of passengers in all cars should start at 0.
- (5 pts) Update `print_train()` to display the current status of occupied seats in each car. For example, if a car has 15 seats and 2 are occupied, printing this car should

include "(2/15)" in the output. The Engine and Caboose should always show "(0/0)" since passengers are not allowed to sit in those cars.

- (15 pts) Write a **recursive** function that takes in a `train_car*` and number of passengers who want to board. Attempt to put them in the current `train_car` by filling any empty seats. If there are not enough seats, send those passengers to the next car. **Return the number of leftover passengers who cannot be seated in the entire train** (not just the number that cannot be seated in the current car). This number will be 0 if they all fit somewhere in the train.
- (5 pts) In `main()`, create a dynamically allocated array of 5 trains (*note that this means creating an array of `train_car*` items, not an array of `train_car` items*). Start each train with an Engine car (*be sure to initialize its member variables*) and then add a random number (from 1-5, inclusive) of cars, including the Caboose. **Print each train and its length** (think: what is the max length possible? Use this to check your implementation)**.**
- (10 pts) Simulate a train journey in your `main()` function using the first train in your array.
  - Print the train before the journey begins.
  - The journey will have 5 stops.
  - At each stop, ask the user how many passengers want to board. Call your boarding function and report to the user how many passengers could not be seated.
  - Print the train after boarding is complete, then go to the next station.
- Free the memory used by all of the trains in your array - no memory leaks. (**10 point penalty for any memory leaks. Use valgrind to check before submitting.)**

Example Run (User inputs are highlighted):

```
5 trains and their lengths:
   0) Engine(0/0)_***_(0/17)_***_(0/14)Caboose(0/0)
      Length 4
   1) Engine(0/0)_***_(0/15)Caboose(0/0)
      Length 2
   2) Engine(0/0)_***_(0/19)_***_(0/20)_***_(0/13)Caboose(0/0)
      Length 5
   3) Engine(0/0)_***_(0/18)Caboose(0/0)
      Length 3
   4) Engine(0/0)_***_(0/11)_***_(0/11)Caboose(0/0)
      Length 4

The journey begins, using train 0!

Engine(0/0)_***_(0/17)_***_(0/14)Caboose(0/0)

Station 1: How many new passengers? 5
Engine(0/0)_***_(5/17)_***_(0/14)Caboose(0/0)
 Seated all but 0 passengers from station 1.

Station 2: How many new passengers? 8
Engine(0/0)_***_(13/17)_***_(0/14)Caboose(0/0)
 Seated all but 0 passengers from station 2.

Station 3: How many new passengers? 10
Engine(0/0)_***_(17/17)_***_(6/14)Caboose(0/0)
```

```
 Seated all but 0 passengers from station 3.

Station 4: How many new passengers? 10
Engine(0/0)_***_(17/17)_***_(14/14)Caboose(0/0)
 Seated all but 2 passengers from station 4.

Station 5: How many new passengers? 3
Engine(0/0)_***_(17/17)_***_(14/14)Caboose(0/0)
 Seated all but 3 passengers from station 5.
```

Even though you will not submit a design document for this assignment, it is highly recommended that you create one for yourself, since it will significantly reduce your programming and debugging time.

1. **Describe the problem** in your own words and any assumptions you are making to help make the problem more concrete. What updates do you need to make to the program?
2. **Devise a plan**: design your solution and show how it will work with either (1) pseudocode (not C++) and/or (2) a flowchart diagram.
    a. Draw a diagram of your proposed new `train_car` struct and its members.
    b. You should have a separate section of pseudocode, or a flowchart, for each function. Start with how they work in the code you download, then modify to show the changes you want to make. You may like to make your changes in a different color to guide you later when implementing.
3. **Identify as many test cases in different scenarios as you can**. The more tests you think of now, the better your program will be later.

Part 3. Implement Your Train Journey Extensions

Implement your extensions and new functions following the design you have developed. **Note: It is normal (in fact, expected) that your design will evolve as you write the program and figure out new details.** The design provides a starting point, but you can deviate from it.

Implementation requirements (in addition to list above):

- Name your file `assign6_trains.cpp`
- (5 pts) Handle bad user inputs:
    - You can assume that the user will enter an integer, but you should check and re-prompt the user if it does not make sense (e.g., negative number of passengers).
- (15 pts) Employ good programming style and follow the course style guidelines: http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/cs161-style-guidelines.pdf
    - No function (except main()) should be more than 25 lines long (excluding lines that consist only of comments and whitespace). If you find your functions are getting too long, break them into multiple smaller functions.
    - Do not put more than one statement on each line.

Implementation don'ts:

- No use of global variables or goto.
- No use of classes, only structs.
- Don't try to solve the whole program at once. Start with implementing one function to get it working (and tested), then move on to the next one. Use your design as a guide.

Part 3. (10 pts) README instructions

Instead of a demo, you will write a README.txt file that includes the following information:

- Your name, ONID, section (CS161-020), assignment number, and due date

1. **Description**: One paragraph advertising what your program does (for a user who knows nothing about this assignment, does not know C++, and is not going to read your code). Highlight any special features.
2. **Instructions**: Step-by-step instructions telling the user how to **compile and run** your program. If you expect a certain kind of input at specific steps, inform the user what the requirements are. Include examples to guide the user.
3. **Limitations**: Describe any known limitations for things the user might want or try to do but that program does not do/handle.

Here is an example README.txt file for a hypothetical assignment (not this one):

http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/README_example.txt

Submit Your Assignment Electronically

(A) Ensure that your program (1) compiles, (2) does not generate run-time errors, (3) has no memory leaks (use valgrind).

(B) Submit your C++ program (.cpp file, not your executable) and your **README.txt** file before the assignment due date, using **TEACH**.