Group 2

Jack Collins

Coding Standards

1. File Naming
   1. Each file name (unless it is a class file) must be in this format: What module the file is part of, then the file’s intended purpose.
      1. Ex: CTC\_ScheduleCalculator.cpp
2. Classes
   1. The name of the class files should be representative of what the class is supposed to do.
      1. Ex: “TrainCar” – a class that represents a singular train car, including its speed, destination, etc.
   2. Each class definition should be in its own file.
      1. Ex: Class.h
   3. Each class source file should also be its own file.
      1. Ex: Class.cpp
   4. Public vs Protected vs Private
      1. ONLY the interface of the class should be listed as public, the implementation should be listed as private/ protected.
   5. Always initialize your variables in a class.
   6. Each class should at least have these 4 methods, the default constructor, the destructor, a copy constructor (if applicable), and an assignment operator (if applicable).
   7. Make use of accessor methods inside of classes to view/edit variables instead of directly accessing the variables in the class.
3. Properties or Attributes
   1. For everywhere they apply, the properties/ attributes for any block of code (including functions, methods, and regular code) should be appropriately explained.
   2. When naming variables/ files, the user must use Camel Case.
      1. “TrainSpeed” is better than “Trainspeed” for a variable name
4. Methods
   1. Just like the class name, each method inside a class should be representative of what is it supposed to accomplish.
      1. Ex: TrainVelocityCalculator { /\* code \*/ };
5. Variables
   1. When using variables, make sure they are representative of what the value is supposed to be.
      1. Ex: label a speed value as “speed” instead of “x”
   2. Make sure the variable names cannot be confused with any other variable name.
      1. Don’t name something “speed2” when something else is already called “speed”, especially since it isn’t obvious what the difference is.
   3. Make use of **structs** when possible.
      1. Making a struct called “Point” with variables “x” and “y” is easier to comprehend than simply using the “x” and “y” variables by themselves.
   4. All variables should be initialized **before** any blocks of code in which they are used.
6. Constants
   1. Constants must be defined at the beginning of the file/ function.
   2. The name of the constants should reflect what the constant is supposed to represent.
7. Comments and Comment Style
   1. There must be comments before every block of code to explain what it does, since others will have to understand the code you’ve written.
   2. For a comment before a function/ method, it must contain the name, a description of the inputs and outputs, and a description of the function.
   3. At the beginning of each file, there should be a comment that includes the author, lasted revision date, and the description of the file.
   4. Any style of comment is ok, as long as there is visible text.
8. Indentation and brackets
   1. Any new code following an opening curly bracket must be indented by one additional tab.
   2. Methods and blocks of code to complete a task must be separated by at least one white space, in order to make the code more readable.
9. Exception Handling
   1. All functions/ methods/ blocks of code should check if the data they received is valid. If not, they must throw an error.
   2. Each error message must be clear about what part of the program the error is coming from and what caused the error to occur.