Assignment 3

C++ has several primitive types (`int`, `double`, etc) and several core language objects such as `string` and `vector`. However, programs often require new objects to achieve elegant designs. One such object may be a `TimeCode` type, which represents an amount of time in hours, minutes, and seconds.

`4:15:32`

The above is an instance of a `TimeCode` object representing 4 hours, 15 minutes, and 32 seconds. An elegant way of implementing this object is to store only one instance variable: `t`. `t` is a number representing only the total number of seconds. 4 hours, 15 minutes, and 32 seconds is `t = 15,332` seconds.

Your task is to implement a `TimeCode` object in `TimeCode.cpp` following this elegant design.

Provided to you is a *screenshot* of the `TimeCode.h` file from the solution. It is a screenshot intentionally so that you have to mentally process its contents (by re-typing it). You probably are thinking to yourself, "Oh great! More work! I believe that the minor tedium of re-typing this code will have untold benefits to my academic and professional development for years to come." and you're right. It will.

You should implement all of the methods in that file. You may optionally implement more methods (public or private) if it seems appropriate to do so.

In addition to the `TimeCode.h` screenshot, `TimeCodeTests.cpp` is provided which contains a small sampling of *some of the tests* you should perform. When grading this assignment I will use a version of `TimeCodeTests.cpp` that contains tests for *every method of the class*. You should significantly expand `TimeCodeTests.cpp` by coming up with and writing many tests for yourself. Your goal is to test as many cases as you can come up with to ensure that your `TimeCode.cpp` code can pass all of *my* tests.

Details

- `TimeCode`s cannot be negative (for the purposes of this assignment). Their individual components (hour, minute, second) also cannot be negative. Most (all?) of your variables should therefore by `unsigned`. If a negative value is identified at any point you can throw an exception, which looks like this:

  ```
  throw invalid_argument("Negative arguments not allowed: " + to_string(min));
  ```

  We haven't learned fully about exceptions yet. You are encouraged to skip ahead to chapter 12 in the ZyBook to explore them further if you're interested.

- `TimeCode`s also cannot have invalid states. For example 00:61:00 should not exist. There are two possible ways to handle this sort of issue depending on the circumstances. I will leave it to you to figure out when to choose them.
  - 1) Convert the invalid state (0:61:0) to a valid state (01:01:0).
  - 2) Throw an exception.

- Do not import the math module. Inside `TimeCode.cpp` the only import is `TimeCode.h`

- The core logic of `TimeCode` should be `GetComponents()`, and `ComponentsToSeconds()`, using these two methods, along with `GetTimeCodeAsSeconds()` it should be relatively easy to implement all other methods (even the constructors).

- Addition and subtraction are intuitive and straightforward conceptually (1:15:22 + 2:9:5 = 3:24:27). They can be confusing when considering roll-over (1:15:55 + 0:1:25). Be careful implementing them!

- There is (virtually) no limit on the number of hours.

- You should not store hours or minutes or seconds directly. You should only store the total number of seconds: `t`.

- Multiplication and division do not make sense between two `TimeCode` objects (1:0:0 * 0:15:0 = ?). So, they are implemented using a `TimeCode` and a `double` (1:0:0 * 0.5 = 0:30:0).

  Actually, if you think about it this is always the case for all units of everything. What is 1 apple divided by ¼ of an apple? Is it "4 chunks of apple" ? No, that is nonsense. But, 1 apple divided by 4 is ¼ of an apple.

There is no sample output for this program. You are done when your `TimeCode.cpp` implements all of the methods in `TimeCode.h` and passes all of your tests in `TimeCodeTests.cpp`

Submission: Your program will be submitted using canvas and git + github.

1) You should create a private `github.com` repository (repo) with your code in it. You may need to create a github user account and a github repository. To achieve this. The name you choose for the repo and your username are arbitrary but I suggest "HW3" as the repo name.

2) Your repository on github.com should be **private** and you should add my account as a collaborator by going to "settings" → "manage access" → "invite collaborator" and entering my username: `fmresearchnovak`

3) On canvas you should upload a link to this github repository
`https://github.com/fmresearchnovak/HW3.git`

I will grade the final commit made before the due-date. This means that you can actually submit on canvas at any time before the deadline, and still make changes (new commits) to the code.