Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen

2018-09-08

Time

Aim

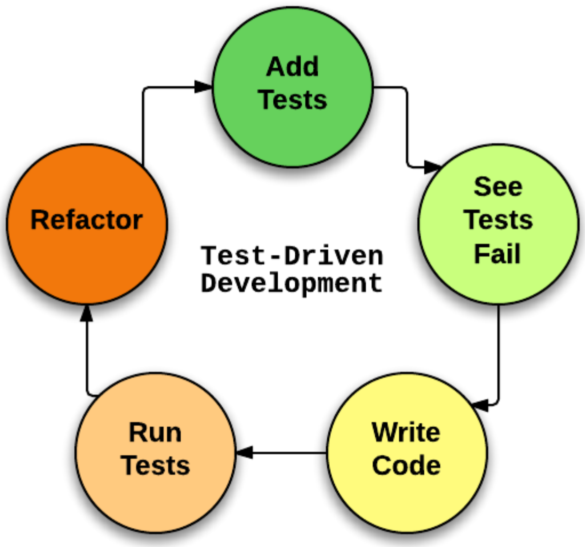
In this assignment you will practice general problem solving, how to work with complex data types (`struct`), functions and operator overloading. You will use the practice of TDD (Test Driven Development).

Litterateur

- `struct`
- Operator overloading `https://en.cppreference.com/w/cpp/language/operators`
- Documentation for Catch: `https://github.com/philsquared/Catch`

Way of working

When you work with this assignment you shall use a methodology called TDD, Test Driven Development. In TDD, when you want to add a certain functionality, you start by thinking how to test this. Compare to the first assignment where you start by implementing the functionality and test afterwards if the functionality is correct. In TDD you start by writing the test first as if the feature is already in place (you have in other words not implemented that function yet). Only after that when you have seen the test fail, should you implement the code as minimally as possible to make it pass. Afterwards you clean up and write better code, this phase is called re-factoring, you rewrite the code so it has better design. This process is repeated until everything is implemented. For each feature, you run all the tests to see if you have broken anything previously implemented.



Figur 1: TDD Way of working

In this assignment you will use the C++ testing framework Catch. Catch is a simple testing framework to get started, the documentation is great (you can find it in the Literature). TDD will also be discussed in the lesson.

Assignment: Time

You will create a data structure and all the functions that are needed to handle time. We defined time as three integers to represent time in the format `HH:MM:SS`. The minimal requirements that must be implemented for time are:

- Must be able to check if the time is valid or not. In other word if the values of hours are between the interval $[0, 23]$, minutes and seconds are between the interval $[0, 59]$. This should be done with a function called `is_valid()`.
- Possibility to get a time as a string. There must be a way to format the time as a 24-hour clock `"14:21:23"`, but also with am or pm `"02:21:23 am"`. The result must be a string in the format `HH:MM:SS[ p]`, where `p` is either am or pm. There must be a function `to_string()` that return a string in the format given above. There must be an extra parameter to `to_string()` that determines whether it is printed in a 24-hour or 12-hour clock format.
- Must be able to check if the time is before or after noon (am or pm). This function must be called `is_am()`.
- Addition with a positive integer `N` to generate a new time `N` seconds into the future with `operator+`. Eg.

```
Time t{};
t + 5; // The result will have the time [00:00:05]. t is still unchanged
```

- Subtraction with a positive integer to give an earlier time with `operator-`.
- `operator++`, `operator--`

```
 Time t{12, 40, 50};
 t++; // t will have the time [12:40:51]
 --t; // t will have the time [12:40:50]
```

  Both operator must have the prefix- and postfix versions.
- Comparison between two `Time` objects with the usual comparison operators (there are six of them). Eg.

```
 Time t1{00, 00, 01};
 Time t2{12, 30, 40};
 t1 > t2; // This is false;
 t1 != t2; // This is true;
```

  It is important to reuse code. Is it possible to implement some comparison operators by calling the others?
- Formatted output with `operator<<`. `Time` must be printed in the format `HH:MM:SS`.
- Formatted input with `operator>>`. You can always assume that a correct input will be in the format `HH:SS:MM`. The values must check for correct values. Your implementation must check that the read values are correct. Eg. Second and minute must be between 0-59, hour must be between 0-23. The `fail`-flag must be set if an incorrect value are read. `operator>>` normally only read until an error occur and the stop. If you choose to implement this in a different way, it is OK as long as you document it.