Programmering for computerteknologi
Hand-in Assignment Exercises

Week 12: Bundling Data and their Functions Together

Please make sure to submit your solutions by next **Monday** (27-11-2023).

In the beginning of each question, it is described what kind of answer that you are expected to submit. If Text answer AND Code answer is stated, then you need to submit BOTH some argumentation/description and some code; if just Text answer OR Code answer then just some argumentation/description OR code. The final answer to the answers requiring text should be one pdf document with one answer for each text question (or text and code question). Make sure that you have committed your code solutions to your GitHub Repository.

Note: the **Challenge** exercises are *optional*, the others mandatory (i.e. you **have** to hand them in).

Exercises

1)

Code answer In this exercise you will create a `Duration` class in `C++`.

(a) Start by creating a basic `Duration` class with a private attribute `time` that represents a number of seconds that have elapsed. Create a public method `get_duration()` that returns the value of the `time` attribute for an object instance.

In `duration.cpp`

```cpp
int Duration::get_duration() {
  // complete this method
}
```

Remember to add the method definition to the header file.

> 🔥 **Tip**
>
> Review the lecture slides for how to define a `C++` class. Don't forget that you need to create both a header file (`.h`) and an implementation file (`.cpp`). You will need to compile your Duration class as a library, as shown in the lecture slides.

> 🔥 **Tip**
>
> Do not include a `main` function in your class definition. You will create a `main` function later in your separate test program.

(b) Next create a public constructor method for your `Duration` class that sets the `time` attribute to 0. This constructor has no arguments.

In `duration.cpp`

```
Duration::Duration() {
  // complete this method
}
```

(c) Create another public constructor method that takes one integer argument `t`, which is used to set the time attribute. You must ensure that the initial time value set in the constructor is always greater than or equal to 0 using `assert()`.

In `duration.cpp`

```
Duration::Duration(int t) {
  // complete this method
}
```

(d) Write a separate `C++` test program with some tests that create instances of the `Duration` class with a variety of different initial start times.

> 🔥 **Tip**
>
> For testing your `Duration` class you can create a separate file e.g. `duration-test.cpp` similar to previous weeks.
>
> You can also use the test framework https://github.com/catchorg/Catch2 that we have been using throughout the assignments to automatically test your code. If you want to do this simply put your tests in the file `./tests/tests.cpp`. **Note** you need to use the functions from the `Catch2` library for this to work properly.
>
> This way you can use
>
> ```
> make -s test
> ```
>
> to run your tests.

(e) Create a public method `tick()` in the `Duration` class that increments the `time` attribute by one (i.e. adds 1 to the value). The `tick()` method does not take any arguments and does not return a value. Update your test program with new tests that demonstrate this feature.

In `duration.cpp`

```
void Duration::tick() {
  // complete this method
}
```

(f) Create another method `tick()` in the `Duration` class that takes one argument amount and adds this value to the `time` attribute value. The value of amount must be greater than 0, which should be ensured using `assert()`. Update your test program with new tests that demonstrate this feature.

In `duration.cpp`

```
void Duration::tick(int dt) {
  // complete this method
}
```

(g) Create a private `alarm` attribute in your `Duration` class that has the same data type as `time`. Also create a boolean attribute `alarm_has_been_set`. The idea is that a user can set a value for the alarm. Create a method `set_alarm()` that allows a user to set the alarm. Think carefully about the purpose of `alarm_has_been_set`. Also update your constructors to give these attributes some useful default values. Do not change the constructor signatures, the user cannot set the alarm through a constructor. If the time value exceeds the alarm value when the user calls `tick()` then it must return true; if not it shold return false. Update the signature of `tick()` so that it returns an boolean. Modify `tick()` so it return either false or true. If true the alarm value should be reset. Modify your test method to check the updated `tick()`.

In `duration.cpp`

```
void Duration::set_alarm(int t) {
  // complete this method
}
```

(h) The user must not be allowed to set the alarm value to a value in the past, i.e. the alarm value must always be greater than the current time. Update your `set_alarm()` method accordingly. Update your test program with new tests that demonstrate this feature.

(i) In general, when writing code for functions and classes, copying code is a bad idea. For example, if two methods M_1, M_2 have almost exactly the same code then a better approach is to create a third more general method M_3 that takes some input arguments that both M_1 and M_2 can use by providing different argument values.

Notice that you have two different `tick()` methods; you may have copy-pasted the alarm update code. Instead, create a new private method `check_and_update_alarm()`. This method should return whether the alarm value has been exceeded by `time`, and resetting the alarm. Call this method from your `tick()` methods. Update your test program with new tests that demonstrate this feature.

Challenge *Self-describing Sequence*

Solomon Golomb's self-describing sequence $\langle f(1), f(2), f(3), ... \rangle$ is the only non-decreasing sequence of positive integers with the propertly that it contains exactly $f(k)$ occurences of k for each k. A few moment's thought reveals that the sequence must begin as follows

n	1	2	3	4	5	6	7	8	9	10	11	12
$f(n)$	1	2	2	3	3	4	4	4	5	5	5	6

In this problem you are expected to write a program that calculates the value of $f(n)$ given the value of n.

Input
The input specifies an integer n, where $1 \leq n$ 2147483648.

Output
The value of $f(n)$ on a separate line.

Example

```
100
21

9999
356

123456
1684

1000000000
438744
```

> 🔥 **Tip**
>
> The Golomb sequence f can also be described by the recurrence relation (given by Colin Mallows)
>
> $$f(1) = 1$$
> $$f(n+1) = 1 + f(n + 1 - f(f(n)))$$