NSWI170 – **Computer Systems** – 2023/24 Summer – Martin Svoboda

Assignment **A5: Stopwatch**

Binding Instructions | Well-Meant Advice | Ideas for Thought | Common Mistakes

501. [**Numeric display class**] We will extend the current display driver with the newly needed functions for displaying whole numbers using the concept of inheritance. We will therefore propose a new class for a numeric display by deriving it from the basic one. If need be, we can of course modify the existing functions of the basic display or also add new ones.

502. [**Division of display responsibilities**] We will, however, adhere to the idea that the basic display will only be responsible for low-level instructions for displaying various individual glyphs themselves, while more complex mechanisms for displaying whole numbers will be solved with its help by the numeric display.

503. [**Domain of supported numbers**] Only the numeric display as such thus gains the ability of displaying whole numbers using all available display positions at once. In particular, we will support non-negative integers and non-negative decimal numbers.

504. [**Displaying digits with a dot**] For displaying digits, we will still use the already existing function of the basic display, we just need to modify and extend it to support lighting up the dot segment in case the optional decimal point was requested. We can use a suitable bitwise operation to mutually combine masks of the corresponding glyphs together.

505. [**Hard-wired dot**] It goes without saying that our display implementation needs to be universal, and so the position of the decimal dot cannot be hard-wired in any way, let alone tied to the requirements we specifically pose on the stopwatch.

506. [**Time multiplexing idea**] We obviously need our display to be capable of showing different glyphs at different positions at the same time in order to display whole numbers. However, that is not structurally possible. Fortunately, we can apply a trick where we just remember the number requested to be displayed at a given moment and unfold its actual displaying in time over the individual positions. This means that we will continuously alternate the individual display positions and during each iteration of the `loop` function we will give an instruction to display just one of them. This will allow us to achieve an optical illusion where we get the feeling that all the positions are lit at once, although technically this will not be the case.

507. [**Requested number representation**] Beside the actual number requested for displaying, we can remember other necessary information inside the numeric display class, too. Let us just add that it is not appropriate to come with a different representation of integer and decimal numbers. Especially when it comes to the decimal ones, we certainly cannot use floating point data types like `float` due to their inaccuracy, and it is also not practical to split such numbers into two parts, integer and decimal.

508. [**Setting a new number**] Let us further note that an integer number without a dot is something else to a decimal number with a dot at any of the positions `0` through `3`. We should remember this when designing the interface of a function or functions we will use to set such numbers.

509. [**Prohibition of invalid values**] To distinguish between the two situations, we could perhaps come with some special position like `-1` and use it just for the case we actually do not want the decimal dot to be included. However, we have already explained earlier that we cannot use such invalid values in general. We therefore need to find another solution.

510. [**Function overloading mechanism**] We only need to separate the function for integers from the function for decimal numbers. Moreover, we can even go further and elegantly use the overloading mechanism. It allows us to have multiple functions of the same name under the condition that they have a different number or different types of parameters.

511. [**Atomicity of setting numbers**] No matter how we design the interface of these setting functions, we should be able to set everything needed atomically with a single instruction. I.e., we should not be forced to gradually call several separate functions, for example, to set the intended number, activate the dot, and determine its position. This would worsen the code clarity and reduce user-friendliness. At the same time, we do not even want to leave our display in an internally inconsistent or otherwise invalid state, however temporarily.

512. [**Omission of leading zeros**] We will display the numbers on our display with unnecessary leading zeros before the first valid position omitted. Be careful to correctly detect such a position especially for decimal numbers, as well as avoid unnecessarily repeated calculations of this position.

513. [**Displaying the minus symbol**] If we voluntarily decided to support negative numbers as well, we would place the minus glyph immediately before the first valid position of the number, not at the beginning of the display on the very left.

514. [**Uniform alternating of positions**] When performing the time multiplexing, we must pay equal attention to each individual position on the display, otherwise we would be able to notice uneven lighting intensity. This also applies to positions where we do not want to display anything.

515. [**Uniform duration of loop iterations**] Implementation of our time multiplex also apparently relies on the yet unspoken assumption that each run of the `loop` function will last the same amount of time. Such an expectation is generally not entirely realistic, but it will work well in our case.

516. [**Merging the same positions**] We could also come with an idea that if the same glyph is to be displayed on two or even more positions, we could do it simultaneously, thus optimizing our code in that sense. However, this is actually not a good idea, because it would only make the whole mechanism more confusing from the logical point of view, and we would not achieve any real benefit anyway due to the increased code volume and branching. Therefore, we will not pursue it.

517. [**Achieving efficient code**] On the other hand, this does not mean that we should give up and not optimize the time multiplex code. Quite the opposite, because it is a mechanism that will be executed during each individual iteration of the `loop` function, i.e., perpetually over and over.

518. [**Invoking the multiplex mechanism**] To be complete, the entire multiplex mechanism is inherently a hardware aspect of our display driver, therefore we must invoke it directly from the `loop` function, not indirectly from our stopwatch, which is supposed to be just an application using our display.

519. [**Display turning off**] We would like to have an opportunity to explicitly turn off the entire numeric display. In other words, we want to be capable of deactivating and bypassing the time multiplex mechanism and returning back to the manual control of the basic display through direct instructions for displaying glyphs. Otherwise, we would continuously need to show some number on the display, which may apparently not always suit us. Even though we do not need such a functionality in this assignment, it will become useful in the following ones.

520. [**Single display instance**] Thanks to the inheritance concept we used, our numeric display will automatically be able to do and contain everything the basic one was able to. This means that we will create only one single instance of the display, namely the numerical one. Creating kind of dual instances of the both would be more than just a technical mistake. Physically, of course, we only have one display device, and both logical instances could also influence each other in an uncontrolled way.

521. [**Chaining of initialization functions**] Both basic and numeric display classes understandably need their own initialization methods. In order to provide the most user-friendly interface possible, we intentionally name both these methods the same. Thanks to that, we can invoke the overall display initialization during the `setup` function in a unified way, without needing to guess which variant of the driver was chosen. However, in order to make everything working correctly, we need to explicitly call the initialization function of the basic display within the initialization function of the numeric one, right at the beginning as its first step. But since the name of this ancestor method is overridden, we have to use a fully qualified name `Display::initialize()`.

522. [**Names of display methods**] When finding suitable names of display methods, we should distinguish functions that really perform active displaying of some glyphs and functions for setting the intended numbers. They do not display anything by themselves, we should hence not confuse the users.

523. [**Class for the stopwatch**] As expected, we will encapsulate all the functionality of our stopwatch in a separate class. It will use the services of buttons and display, but the stopwatch logic itself must be strictly separated from these drivers.

524. [**Stopwatch design procedure**] Proposal of the stopwatch class is worth not to be underestimated. If nothing else, we should thoroughly think about the range of the stored information, internal states and transitions between them, detection of events triggered by individual buttons, execution of the corresponding actions, correctness of the time measurement, preparation of values to be displayed, and control of the numerical display itself. Careful consideration and separation of all these aspects will lead to better code and save our time during debugging.

525. [**Global variables for drivers**] Let us remind that we have not changed anything in the way how we work with the global variables for button or display instances, nor we will change it in the future. We therefore continue to access them directly and not pass them through function parameters. Let us also add that we cannot even store them as data members inside the stopwatch class, since no aspect of our hardware handling can depend on specific applications such as our stopwatch.

526. [**Enumeration class for states**] Although even different means exist, it is probably most reasonable to represent the internal states of the stopwatch using values of an enumeration class. We just need to declare it via `enum class MyEnum { VALUE_1, ... }`, individual values are then accessed using `MyEnum::VALUE_1`.

527. [**Repetition of the same code**] As traditionally, we should not forget to appropriately decompose the internal stopwatch functionality so as not to repeat similar or the same code fragments unnecessarily.

528. [**Logical stopwatch actions**] In this sense, we completely separate the implementation of various logical stopwatch actions from the event handling of individual buttons, i.e., invocation of these actions based on the detected events and also the current internal stopwatch state.

529. [**Placement of the button handling**] Due to the non-trivial extent and the need to correctly reflect the aforementioned internal stopwatch state, it is advisable to place the entire button handling code in some stopwatch method and so no longer keep it directly in the `loop` function, where we should not focus on low-level details of individual applications.

530. [**Decimal dot position**] Even though we assume the time is measured with accuracy of tenths of a second in the specification, and therefore placement of the decimal dot at position `1`, we will program the code so that the position of this dot (i.e., let us say accuracy of the stopwatch) is easily configurable within the meaningful limits.

531. [**Granularity of the internal time**] Let us also realize that in order to achieve the expected accuracy and correctness of the measurement, we have to record the accumulated time inside the stopwatch with a significantly more precise granularity than what we expect at the output.

532. [**Time overflow checking**] Let us also remind that we should solve the overflow issue as well when counting the elapsed time, both from the point of view of the displayed value on our display as well as the system time as such.

533. [**Output time calculation**] When converting the internal time into the expected output format, we just perform a simple division operation. That means we should of course introduce a named constant for this purpose, the value of which we also derive. And let us emphasize correctly.

534. [**Gradual time addition**] We should not update the internally stored item with the overall measured time in every single run of the `loop` function in order to avoid possible accumulation errors arising from small inaccuracies. In other words, this means separating the internal measured time from the output values intended for displaying.

535. [**Unnecessary display updates**] We will undoubtedly instruct the display to show the required time value only when necessary, i.e., only when this value truly changed.

536. [**Change detection efficiency**] This is especially important in situations when the stopwatch is currently on. Not only we cannot unnecessarily adjust the display in every single run of the `loop` function, but at the same time we must also ensure that the detection of the right moment to update the display is not too computationally demanding. In this sense, an approach where we would calculate the current output value in each run of the `loop` function and only compare it with the previous one seems not appropriate enough in this sense.

537. [**Scheduling display updates**] A more efficient way will be to use the familiar timing control mechanism we already used several times during the previous assignments. We will just not consider a fixed interval, we will plan the next event appropriately instead, i.e., specifically according to the current situation.

538. [**Validity of the displayed value**] Whatever approach we choose, we must always show the current value of the measured time on the display, not one that would no longer be valid, however briefly.

539. [**Visual correctness checks**] Finally, in relation to the debugging process of the entire application, let us add that the human eye is nowhere near capable of detecting anything at the level of individual milliseconds, so a mere visual verification of the correct operation of our application cannot ensure its true correctness and so the accuracy of the stopwatch measurement.