NSWI170 – **Computer Systems** – 2022/23 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 using the concept of the inheritance. Therefore we will design a new class for a numeric display by deriving it from the basic display. Of course, this does not prevent us from being able to modify or add new functions to the existing basic display if necessary, but we will limit ourselves to low-level aspects only.

502. [**Domain of supported numbers**] Only the numeric display as such gains the ability to display whole integers using all available segment display positions. Specifically, we want to be able to display any integers, including the negative ones, as well as decimal numbers with the decimal dot placed at any position.

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

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

505. [**Time multiplexing idea**] When displaying different glyphs at individual positions of the display, we have to apply the idea of time multiplexing, that is, we cannot display the intended number all at once at the moment it is requested. Instead, we just remember it and unfold its display in time over the individual positions. This means that we will continuously iterate through individual display positions and during each run of the `loop` function we will give a low-level instruction to display just one of them. This allows us to achieve the necessary optical illusion, when we will get the sense that all positions are lit at once, although technically this will not be the case.

506. [**Requested number representation**] Inside the numerical display class, we can also remember other necessary internal information if need be, but the basis at the logical level must be the number requested to be displayed as such.

507. [**Omission of leading zeros**] We will display numbers on our display with unnecessary leading zeros before the first valid position omitted. We have to correctly solve the detection of such a position for whole and decimal numbers, too.

508. [**Displaying the minus symbol**] In the case of negative numbers, we also display the minus glyph, which is to be placed immediately before the first valid position of the number, not at the beginning of the display on the very left.

509. [**Uniform alternating of positions**] When performing the time multiplexing, we must pay equal attention to each individual position of 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.

510. [**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, of course, unrealistic. On the other hand, alternating positions during `loop` iterations on the one to one basis is still probably the best possible solution, since other methods would have to include timing control, and thus would be excessively complicated on their own regarding the expected benefits they would yield.

511. [**Merging the same positions**] We could also come with an idea that if the same glyph is to be displayed on two or 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 thing more confusing from the logical point of view, and we would not achieve a real benefit anyway due to the increase in code volume and branching. Therefore, we will not pursue it.

512. [**Efficient multiplexing code**] On the other hand, it is obvious that the code handling the time multiplexing will be called from every iteration of the `loop` function, and therefore should not contain things that would not be efficient enough by themselves.

513. [**Setting a new number**] Let us further note that an integer without a dot is different from a decimal number with a dot at any of the positions `0` through `3`. We should also remember this when designing the interface of functions that we will use to set such numbers. It is certainly not possible to use, for example, an invalid position such as `-1` for situations where we do not actually want the decimal dot at all. Fortunately, we can elegantly take advantage of the overloading mechanism, where we can have multiple functions of the same name if they have different numbers or different types of parameters.

514. [**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. So we should not be forced to call several separate functions in turn, for example, to set the intended number, activate the dot, and determine its position. This would significantly worsen clarity and reduce user-friendliness.

515. [**Names of display functions**] When searching for suitable names of display functions, it is necessary to distinguish functions that really perform active displaying of some glyphs and the discussed functions for setting the intended numbers. They do not display anything by themselves, so we should not confuse the users.

516. [**Display turning off**] We should program the entire numeric display so that we have an option to disable it explicitly. Otherwise, we would always need to show something on the display thanks to the multiplexing integrated into the `loop` function, which may not suit us in general.

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

518. [**Stopwatch design procedure**] Design of the stopwatch class is worth not to be underestimated. If nothing else, we should thoroughly think about the range of stored information, internal states and transitions between them, detection of events triggered by individual buttons, implementation of the corresponding actions, correctness of time measurement, preparation of values to display, 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.

519. [**Enumeration class for states**] Although it could also be done differently, 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 using the construction `enum class MyEnum { VALUE_1, ... }`, individual values are then accessed using `MyEnum::VALUE_1`.

520. [**Repeating 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.

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

522. [**Granularity of internal time**] In addition, it is necessary to realize that in order to achieve the expected accuracy of the measurement, of course we have to record the accumulated time inside the stopwatch with a more precise granularity than what we expect at the output.

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

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

525. [**Gradual addition of time**] 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.

526. [**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.

527. [**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.

528. [**Scheduling timer events**] Therefore, a more efficient way will be to use the familiar timing control mechanism, which we have already used several times during the previous assignments. We will just not consider a fixed interval, but will plan the next event appropriately, i.e., specifically according to the current situation.

529. [**Displayed value currentness**] Whatever procedure we choose, we must of course ensure that we always show the current value of the measured time on the display, not one that would no longer be valid, however briefly.