NSWI170 – **Computer Systems** – 2022/23 Summer – Martin Svoboda

Assignment **A6: Messages**

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

601. [**Extension of the basic display**] We will preserve the existing functions of the basic display unchanged, and extended them especially by displaying individual letters of the English alphabet, with their size suppressed. Finally, we add a wrapping function that will accept a single arbitrary character and via branching determine correct glyphs to be displayed for all of the following situations: letter, digit, space for any white character, and a special glyph for any unsupported character.

602. [**Interface of member functions**] Interface of a function for displaying particular digits as well as a function for displaying particular letters must be designed in such a way that we can use them independently, and thus be full-fledged and user-friendly.

603. [**Type of character parameter**] Specifically as for the function for displaying letters, we will technically expect a parameter of data type `char` and not a pointer to `char`, whether it would be a standalone character or just a part of some array or string. The function will thus be more universal and, moreover, faster.

604. [**Temporary debugging glyphs**] For debugging purposes, it will be convenient for us to temporarily use other than empty and especially distinct special glyphs for spaces and the already mentioned unsupported characters. Thanks to that, we can correctly distinguish between these two situations, and we will therefore be sure that we have implemented their processing correctly. Only before the assignment submission do we adjust both constants accordingly.

605. [**Preserving position numbering**] Although it makes sense to align letters on our display to the left as opposed to numbers, we will still preserve the convention of numbering display positions from `0` to `3` from right to left.

606. [**Text display representation**] Similarly to the numerical display in the previous task, this time too we will program our text display using the inheritance, i.e., by deriving it from the basic display. It will then be able to display the intended string of characters, naturally using time multiplexing again.

607. [**Text display functions**] Text display driver will remember and thus can only display four specific symbols. In other words, it will be able to display strings with length exactly corresponding to the number of positions we have on the display. Never longer.

608. [**Setting a requested string**] Inside the function for setting a new string to be displayed, we copy all the requested characters into our own internal data member. In other words, it would not be enough to store just a pointer to that string, because we do not want to force the callers to ensure that such a string would be guaranteed to exist for as long as we want to use it.

609. [**String constancy**] Interface of this function will expect a pointer to a constant string, i.e., `const char*`. With the constancy flag, we not only say that we are not interested in changing this input string, which the compiler will then help us to ensure, but, at the same time, it is also necessary to allow calling this function with such constant strings at all. E.g., strings provided directly as literals.

610. [**Input string length**] Even though we expect the length of a provided string to correspond exactly to the size of our display, we still have to implement the entire function in a way that we can correctly handle even strings that are shorter or longer. In the former case, we append them from the right with spaces, in the latter case, we cut them off and simply ignore the rest.

611. [**Array bounds checking**] At the same time, it is necessary to realize that the correct treatment of both the previous situations is necessary even to prevent potential reading or even writing beyond the

array end. In the best case, these errors will lead to an immediate fall of our program when accessing unallocated memory, in the worst case, we will overwrite some of our other data. The problem can then manifest itself at any time later and, above all, it will be very difficult to debug it.

612. [**Usage of pointer arithmetic**] In order to achieve a more efficient implementation and at the same time learn how to work with pointers, we will use them manually when working with our strings instead of the square bracket operator. This means any code where we will iterate over the individual characters of a string. So, instead of a loop over the position numbers and accessing the elements using the `string[i]` construct, we will use a gradually incremented pointer and `*` for dereferencing. Accessing a single particular position outside of a cycle will be an exception, we can still use square brackets there. Simply because nothing would be gained.

613. [**Complicated string modifications**] Since we perceive the text display class as an output device driver, it is not possible for its function for setting the strings to solve any more complex application logic, such as appending some number of spaces before the beginning, etc.

614. [**Filling the internal array from outside**] Finally, let us add that the data member of the internal array for the current characters is owned by the text display, so we really need to fill the content of this array using the discussed function, it is not possible to provide a pointer to this array to someone else in order to fill it from the outside.

615. [**Display turning off**] As with the numeric display, we will also implement the text display so that we have an option to turn it off. I.e., do that explicitly, not using a trick where we would, e.g., request to display a string containing spaces only.

616. [**Class for running messages**] We consider the whole task of scrolling and setting individual text messages to be an application problem. Therefore we have to solve it via a separate class, which will be completely separated from the text display driver.

617. [**Prohibited display extension**] Likewise, this new class cannot be understood as just another derived and extended version of our display driver. To achieve the necessary functionality of scrolling longer strings, we will therefore use the text display and give it instructions, but we cannot inherit from it by ourselves.

618. [**Completion of message scrolling**] Having displayed the entire current message, the display terminates in a state where spaces are displayed on all positions. We will then not turn the display off nor explicitly display empty glyphs on all positions, we will simply refrain from any further actions.

619. [**Function for completion testing**] It is because we expect that the logic of setting new messages will be figured out from the outside. Beside the function for setting a new message, our class must also offer a function with which it will be possible to find out whether scrolling of the last message has already completely finished or, on the contrary, is still in progress.

620. [**Setting up a new message**] Although it would be preferable to make our own copy of a message string requested for displaying, we will be content here with just saving the provided pointer. In other words, this time, on the contrary, we can rely on the fact that this pointer will be valid and the original string available all the time.

621. [**Dynamic allocation mechanism**] The reason is that our message can have an arbitrary and therefore beforehand unknown length. If we really wanted to make a copy of it, it would lead to the need of using the so called dynamic memory allocation. However, we want to avoid that in this course, because its use requires a certain circumspection and discipline. Without it, we could uncontrollably and irreversibly lose available memory in a running program.

622. [**Arbitrary message length**] Length of a message requested to be displayed can be completely arbitrary, including zero. However, we will never get an invalid pointer.

623. [**Redundant display changes**] It goes without saying that we give instructions to the text display only when a change occurs, i.e., at the moment the current message is scrolled to the next position.

624. [**Disallowed system functions**] We avoid dynamic allocation, it is not necessary, and we did not learn it anyway. Likewise, we will not use any resources offered by the `cstring` library.