Technical University of Munich
School of Engineering and Design
Prof. Dr. Martin Werner

Computational Foundations I (Winter Term 2021/22)
Tutorial 4

Tasks marked with a star like **Optional Task*** are optional. Tasks marked like **Hard Task+** are given, but it is not expected that you solve them now. It is great if you learn to solve them during the lecture. Go back to them after a few weeks and see your own progress.

Learning Outcome: We are hitting the first point of the lecture, where the 4CID circle is applied. As explained in the first lecture, we will start repeating things in slightly different contexts such that you can practice and learn to solve tasks yourself.

Task 10: MATLAB Recap

This task parallels Task 1 of this semester. Therefore, it might be a good idea to review first the task on sheet 1 (also as sheets have seen quite significant updateS), before you solve the following tasks.

a. **On Numbers:** Remember, type casting where numbers are enforced to be a certain type (e.g., you can force a floating point number to be an integer, essentially truncating the number). You need to do this often, when you only approximately calculate the location in a matrix or vector and then want to access an element. Using a floating point number in this context, even if it represents an integer, leads to an error.

   Complete the following erroneous example to run without errors by casting to integer and by using one of the tree rounding functions `floor`, `round`,`ceil`.

```
1;
clear;
tbl = [1,2,4,8,16,32,64]
argument = 1.5
argument_integer = % complete this turning argument into an integer that
    can be used to index arrays
conclusion_low = tbl(argument_integer)
conclusion_high = tbl(argument_integer+1)
fprintf("Conclusion of TypeCast Version: %f <= 2^%.2f <= %f\n",
    conclusion_low,argument, conclusion_high)
```

b. **Custom Eye Matrix:** Implement a function `custom_eye` which parallels the eye function. Implement it in two versions: In the first version, use two loops. The outer loop each time adds a new row to the matrix while the inner loop each time adds a single value to a local variable row. In the second version, use the `zeros` function to initialize the matrix in the right size filled with zeros and use a single for loop to set all relevant locations to one. Tip: To extend a vector to the right, use `x = [x,newvalue]`. To extend a matrix down use `x = [x; value]`. Note that both only work if dimensions match (you can only add a row with n entries to a matrix of n columns.

c. **Functions and Strings:** Implement the MIU system as follows: start with an empty MATLAB file and implement one function for each rule. Each of these functions takes the input string as the parameter and returns a matrix with one row containing a string if the rule is applicable. Then the matrix should hold the output of applying the rule. If the rule is not applicable it returns an empty matrix. With this functional decomposition, the main program for creating all strings of the MIU system can be a simple for loop appending the output of every function.

More concretely, complete the following program:

```
1;
clear
M = ["MI"];

% you can use this as a test
%rule1(M(end))
for i=1:10
    input_element = M(i);
    % apply all rules on this
    new = rule1(input_element);
    if strcmp(new,"")  == 0
        M = [M,new]; % add this string for the future
    end
    % repeat this code with rule2..4
    fprintf("== ITERATION %d ===");
    disp(M);
end



function ret = rule1(in)
  ret = ""
  in = char(in);
  if in(end) == 'I'
    ret = [in,'U'];
    ret =convertCharsToStrings(ret);
  end
end
```

There are a few things to note and to learn in this task: First of all, strings are given in two different ways in MATLAB. There is a string type and a character array type. The string type provides matrices in which every entry is itself a string of variable length. A character array is a string decomposed into its characters and, hence, matrices of strings can only store string data of equal numbers of characters along both axes. The MIU rules are based on such characters, however, therefore we need to convert between both worlds quite a few times.

In more detail: We represent all words of the language we discovered as a vector of strings. Such a vector looks similar to `v = ["MI","MIU"]`. Each entry of the matrix holds a full string, for example, for the left vector `v(1)` holds `"MI"`. All rules of the MIU system, however, are formulated and processed on the level of characters. Therefore, in function `rule1`, we convert the string into a character array (char array). For `"MI"` this means that after the conversion, we end up with `in = 'MI' = ['M','I']`. Note the different notation of string constants and char arrays. In this representation, `in(end)` resolves to `'I'`, hence, Rule 1 is applicable and applied on the level of character arrays (appending a character in this case). As we expect the result of rules to be strings, we convert back to a string in the last line using the `convertCharsToArray` function. The last novel aspect is that string comparison needs to be done with the `strcmp` function. This is very common, for example,

the same function exists in the C programming language. For now, it is sufficient to know that `strcmp` returns 0 in case of equality.

Task 11: MATLAB Refresher

Go through the slides, the skript, the tutorial, and your minutes from the tutorial session. Find all MATLAB constructs you used and recall how they work.

- functions used (like `ceil`, `round`, `plot` ...)
- operations or operators used (e.g., matrix multiplication, solution of linear equation)
- constructs (for, while, if, function, ...)

For this task, no submission is expected.