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

Computational Foundations I (Winter Term 2021/22) Tutorial 5

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 train recursion, see that loops can be implemented by recursion, and create our first text file from within a program.

Task 10: Recursion

Recursion is a fundamental programming technique which we train a bit in this task.

a. **Fibonacci Sequence:** The Fibonacci sequence is a traditional sequence of numbers which seems to be known since about 450 BC. It pops up in really many topics in mathematics, geometry, computer science, and natural science. In this task, we will concentrate on the recursive definition and implement it. The definition is very easy: First, we fix $F_0 = 0$ and $F_1 = 1$. All other Fibonacci numbers are then given based on these as

$$F_n = F_{n-1} + F_{n-2} \text{ for } n \geq 2$$

Implement a MATLAB function `function ret=fibonacci(n)` which computes the n-th Fibonacci number. Compute the sequence of Fibonacci numbers in a `for`-loop, maybe the first 25 numbers, and compare to the Online Encyclopedia of Integer Sequences `https://oeis.org/A000045`.

b. **Greatest Common Divisor:** The prime decomposition of integer numbers claims that each integer number can be represented as a product of prime numbers and that this representation is unique. For example, the number 15 is uniquely decomposed as $15 = 3 \cdot 5$ or 8 is decomposed as $8 = 2^3$. With this unique decomposition, is is possible to define the **Greatest Common Divisor (GCD)** of two numbers a and b as the number which is built from all prime factors that both numbers share (with their multiplicites). For example, the GCD of 8 and 15 would be computed from their decompositions $8 = 2 \cdot 2 \cdot 2$ and $15 = 3 \cdot 5$. As they don't share a prime factor, the GCD is given by the empty product, which evaluates to $\mathrm{GCD}(8, 5) = 1$. For $24 = 3 \cdot 2^3$ and $4 = 2^2$, however, the GCD is $\mathrm{GCD}(24, 4) = 2^2 = 4$ as we can pick two times a two from both prime decompositions.

Write a very simple algorithm which just loops through all numbers, uses the MATLAB `rem` function to test whether a a number divides another. You don't need recursion for this one. Compute the GCD of 9214 and 1513. Print the result using `fprintf(...)`.

c. **Tail Recursion:** We have learnt in the lecture that loops are an essential part of programming (cf. structured program theorem). However, there are quite a few programming languages that (at first sight) don't provide loops. They, instead, provide recursion. Given the following program, write a recursive version which runs in the same or the opposite order. Each invocation of the recursive function shall run no more than one loop body and call itself as long as loop bodys should be computed.

```
1;

recursive_loop(1,5, @loop_body);

function loop_body(x)
   fprintf("Executing Loop Body %d\n",x)
end

function iterative_loop(start, endidx, fun)
    for i=start:endidx
       fun(i)
    end
end

function recursive_loop(start, endidx, fun)
    if (...) % Implement a Suitable Termination
        return
    end
    fun(start) % Call the Loop Body
    recursive_loop(...) % Recursively call recursive_loop
end
```

What is interesting to note in this example is that `@loop_body` creates a function handle with which we can give functions as arugments. In this way, we sometimes implement a simple algorithm and let users of our code implement a tiny aspect of it. When reflecting the purpose of this task, one realizes that it shows that recursion is enough to simulate loops. For the converse, that the elements of the structured program theorem (sequence, loop, branch) are sufficient to simulate recursion, we will first introduce the stack data structure. That is, for the converse, we will need to store data. The family of recursive functions which can be implemented in terms of structured programming without an additional data structure is known as tail recursions.

Task 11: Storing Data in Files

Very often, you will want to store the results of experiments, analyses or complicated computations over a long time. Therefore, MATLAB provides convenient and quite efficient variable storage mechanisms. We learn how to use it in this task.

a. **Storing and Loading Variables:** In one of the previous tasks, you were asked to create turtle graphics such as a nice spiral. This might have looked similar to:

```
global L pen heading location

L=[];
pen = true;
heading = 0;
location=[0,0];

for i =  1:99
   forward(5+i)
   turn(15)
end

plot(L(:,1), L(:,2)) % Here we plot the spiral
```

```
function turn(x)
   global heading
   heading = heading + x;
end

function forward(x)
   global heading pen location L
   last_loc = location;
   dir = [cos(deg2rad(heading)), sin(deg2rad(heading))];
   location = last_loc + dir*x;
   if pen
      L = [L; last_loc; location; NaN,NaN];
   end
end
```

You are now supposed to remove the `plot` call from this program such that it does not visualize the spiral directly. Instead, it shall save the needed information (`L`) into a file. Then, implement a second file which just loads this data file and plots the data. Read the documentation of `save` and `load` and have a look at the lecture video.

b. **Generic Text Files$^+$**: While the `save` and `load` mechanism is nice within the MATLAB environment, it is not designed for data exchange, maybe with other researchers or different programming languages. However, basic text file support in MATLAB is not complicated and based on POSIX standards. Therefore, the principle is shared among all but a few high-level languages:

As files are managed by the operating system, you can only create a file by using an Application Programming Interface (API) of your operating system. In this context, a file is just an entry in a table that is managed by the operating system. You identify files by their index in this table. Such an index is often called **handle** and can be used with a set of functions.

First, a file must be opened. Three basic modes are available: open for reading, open for writing (replacing the file), and open for appending. We will only use write in this task. The basic approach is now as follows: Opening the file gives you a handle. Then, `fprintf` can be used to write to the file, finally, `fclose` should be called to signal to the operating system that the file operations have completed. Or in code:

```
1;
f = fopen("test.txt","w")
fprintf(f,"Hello File\r\n")
fclose(f)
```

Write a program that calculates all prime numbers smaller than 100 using the MATLAB built-in function `primes(...)` and store all values into a file (one number per line) called "primes.txt". Open the file in MATLAB (it should appear in the browser on the left).