Programmering for computerteknologi
Hand-in Assignment Exercises

Week 11: Verifying Correctness of Recursive Programs

Please make sure to submit your solutions by next **Monday** (20-11-2023).

In the beginning of each question, it is described what kind of answer that you are expected to submit. If Text answer AND Code answer is stated, then you need to submit BOTH some argumentation/description and some code; if just Text answer OR Code answer then just some argumentation/description OR code. The final answer to the answers requiring text should be one pdf document with one answer for each text question (or text and code question). Make sure that you have committed your code solutions to your GitHub Repository.

Note: the **Challenge** exercises are *optional*, the others mandatory (i.e. you **have** to hand them in).

Exercises

1)

Text answer Write down a proof that the following recursive factorial function is correct using *proof by induction*.

🔥 **Tip**

Review the lecture slides for the two components of a proof by induction, i.e. (a) the *base case* and (b) the *inductive step*.

```c
// Factorial function definition
int fact(int n) {
  // pre-condition
  assert (n >= 1);
  // post-condition
  if(n > 1)
    return n * fact(n - 1);
  else
    return 1;
}
```

2) See `sumn.{h,c}`

Code answer Consider the inductive proof below. It proves that the sum of the first n positive odd numbers equals n^2, that is, $(2 * 1 - 1) + (2 * 2 - 1) + ... + (2 * n - 1) = n^2$.

Your task is to use the content of the inductive proof as inspiration to create a recursive function that calculates the sum of the first n positive odd numbers.

🔥 **Tip**

Try to identify the parts of the inductive proof that correspond to the *base case* and *recursive step* that you need to implement in your recursive function.

Proof by Induction

Base case:

$2 * 1 - 1 = 1$

Inductive step:

Assume $(2 * 1 - 1) + \cdots + (2 * (n - 1) - 1) = (n - 1)^2$

Then

$$\begin{aligned}(2 * 1 - 1) + (2 * 2 - 1) + ... + (2 * n - 1) &= \\ (n - 1)^2 + (2 * n - 1) &= \\ n^2 - 2n + 1 + 2 * n - 1 &= \\ n^2\end{aligned}$$

3) See `sum.{h,c}`

Code answer Convert the following recursive program into:

```c
// Sum integers 1 to n
int sum(int n) {
  assert(1 <= n);

  if (1 < n) {
    return n + sum(n - 1);
  } else {
    return 1;
  }
}
```

1. An equivalent tail recursive program.
2. A program using a while loop.

Add test cases for the two functions in `./tests/tests.cpp` within `TEST_CASE("sumwhile")` and `TEST_CASE("sumtail")`.

4) **See** `fib.{h,c}`

Code answer Convert the following recursive Fibonacci function into an equivalent tail recursive program.

```c
int fib(int n) {
  // pre-condition
  assert (n >= 1);
  // post-condition
  if(n == 1) {
    return 1;
  } else if(n == 2) {
    return 1;
  } else {
    return fib(n - 1) + fib(n - 2);
  }
}
```

Add test cases for the function in `./tests/tests.cpp` within `TEST_CASE("fib")`.

Challenge

Consider the `C` function:

```c
int f(int n) {
  assert(1 <= n);

  if (n == 1) {
    return 1;
  } else {
    return ((2 * n) - 1) + f(n - 1);
  }
}
```

A)
What does this function compute?

🔥 **Tip**

For a given n (e.g. $n = 5$) write down exactly what each term in the function evaluates to, e.g.

$$\begin{aligned} f(5) &= ((2 \cdot 5) - 1) + f(4) \\ &= 9 + f(4) \\ &= 9 + ((2 \cdot 4) - 1) + f(3) \\ &= \ldots \end{aligned}$$

Eventually you will end up with a *series* that can give you insight into what is being computed.

B)
Prove by induction that f computes what you claim that it computes.