Week 11 Programming Assignment

Steffen Petersen — au722120

November 21th 2022

Here is the link for my repository, in which you will find all the edited code files and such.
https://github.com/Aarhus-University-ECE/assignment-11-SirQuacc

1

Write down a proof that the following recursive factorial function is correct using *proof by induction*. Put your inductive proof into a pdf file (`text_answers.pdf`). *Hint:* review the lecture slides for the two components of a proof by induction, i.e. (a) the base case and (b) the inductive step.

```
/* 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;
}
```

We know per definition that the factorial of 1, is 1, and we can confirm that if n=1 in the function, it will pass the assert, escape the if and enter the else, which returns 1. Thus the base-case of the program, for n=1, must be correct.
We also know that other factorials of natural numbers, are the given number n, times all of its previous numbers, until 1, i.e.:
$n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 2 \cdot 1$
The inductive step of the function is to return $n \cdot fact(n-1)$, meaning if we assume $fact(n-1)$ is correct, multiplying this by n, will also be correct. And since the base case is correct, we know that n=2 is also correct, and since n=2 is correct n=3 must also be.
We could continue this, theoretically, for any positive natural number. This, in combination with seeing that the recursion calls the function correctly with $fact(n-1)$, means the whole function is correct. Because the $fact(n-1)$ step ensures we will reach our base-case of n=1 eventually.

2

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

Hint: 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

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

Below is the recursive function, it can also be found in sumn.c

```
int sumn (int n)
{
    assert(n >= 1);
    if(n == 1){ // Base case
        return 1; //2*1-1 = 1
    }
    else{
        return 2*n-1 + sumn(n-1); //Recursive step 2*n-1 + 2*(n-1)-1 and so on.
    }
}
```

3

Convert the following recursive program into (a) an equivalent tail recursive program, and (b) a program using a *while* loop. Add test cases for the two functions in `tests\src\tests.cpp` within `TEST_CASE("sumwhile")` and `TEST_CASE("sumtail")`

```
/* Sum integers 1 to n */
int sum(int n)
{
  /* pre-condition */
  assert (n >= 1);

  /* post-condition */
  if(n > 1)
    return n + sum(n - 1);
  else
    return 1;
}
```

The code for these can be seen below, and can be found in sum.c

a

```c
int sumtail (int n, int total)
{
    assert(n >= 1);
    if(n > 1){
      return sumtail(n-1, n + total); //For n's above 1, update the running total by adding n, and re-call func with n-1
    } else
        return 1 + total; //Base case, when reached, return 1 + running total
}
```

b

```c
int sumwhile (int n)
{
  assert(n >= 1);
  int r = 0; //Sum variable to return
  while(n > 0){ //Run loop n times (subtracting 1 each time)
    r+=n; //Add current n value to result variable
    n--;
  }
  return r;
}
```

4

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

```
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\src\tests.cpp` within `TEST_CASE("fib")`

The tail-recursive function is seen below, using p and pp as running total inputs, starting at the second and first values of the sequence (1 and 0).
The function can be found in fib.c

```c
int fib (int n, int p, int pp)
{
    assert (n >= 1);
    if(n == 1){
        return pp; //If input is simply 1, return pp, should be 0.
    } else if(n == 2){
        return p; //If input is 2 (can happen recursively), return the new p value
    } else
        return fib(n-1, pp + p, p); //If we want a number in the sequence later than 2, call fib again with updated p and pp as running sums.
}
```