CSci 2041: Advanced Programming Principles

Spring 2017

Homework 3, due at noon on Wednesday March 1, 2017, via GitHub.
(Submission instructions are given at the end of this document.)

Question 1: Power function, over natural numbers
Recall the OCaml `power` function over natural numbers, shown below:

```
let rec power n x =
  match n with
  | 0 -> 1.0
  | _ -> x *. power (n-1) x
```

Using induction over natural numbers show that

$$\texttt{power n x} = x^n.$$

Your proof must explicitly and clearly indicate the base case you prove, the inductive case you prove and what the inductive hypothesis provides in the proof.
Each step in your proof must be accompanied by a justification describing why that step could be taken.

Question 2: Power over structured numbers
Recall from lecture the OCaml type `nat`, the function `toInt`, and the `power` function working over this representation of natural numbers:

```
type nat = Zero | Succ of nat

let toInt = function
  | Zero -> 0
  | Succ n -> toInt n + 1

let rec power n x =  match n with
  | Zero -> 1.0
  | Succ n'-> x *. power n' x
```

What is the principle of induction for the type `nat`?

Using induction over `nat` values show that

$$\texttt{power n x} = x^{\texttt{toInt}(n)}$$

Your proof must explicitly and clearly indicate the base case you prove, the inductive case you prove and what the inductive hypothesis provides in the proof.
Each step in your proof must be accompanied by a justification describing why that step could be taken.

Question 3: Length of lists
Consider the following function over lists:

```
let rec length = function
  | [] -> 0
  | x:xs -> 1 + length xs
```

What is the principle of induction for the `list` type?

Using induction show that

```
length (l @ r) = length l + length r
```

In doing so, you may want to use properties of list like the following:

```
(l1 @ l2) @ l3 = l1 @ (l2 @ l3)
x :: xs = [x] @ xs
```

Your proof must explicitly and clearly indicate the base case you prove, the inductive case you prove and what the inductive hypothesis provides in the proof.
Each step in your proof must be accompanied by a justification describing why that step could be taken.

Question 4: List length and reverse
Recall our function for reversing a list:

```
let rec reverse l = match l with
  | [ ] -> [ ]
  | x::xs -> reverse xs @ [x]
```

Using induction show that

```
length (reverse l) = length l
```

Your proof may refer to the definition of `length` in the previous problem. Your proof must explicitly and clearly indicate the base case you prove, the inductive case you prove and what the inductive hypothesis provides in the proof.
Each step in your proof must be accompanied by a justification describing why that step could be taken.

Question 5: List reverse and append
Consider the following definition of `append`:

```
let rec append l1 l2 = match l1 with
  | [] -> l2
  | (h::t) -> h :: (append t l2)
```

Using the definition of `reverse` from the previous question and the definition of `append` above, show, using induction, that

```
reverse (append l1 l2) = append (reverse l2) (reverse l1)
```

Your proof must explicitly and clearly indicate the base case you prove, the inductive case you prove and what the inductive hypothesis provides in the proof.
Each step in your proof must be accompanied by a justification describing why that step could be taken.

Question 6: Sorted lists
Consider the functions for ordered lists from the `ordered_list.ml` file in the code-examples directory of the public class repository:

```
let rec place e l = match l with
  | [ ] -> [e]
  | x::xs -> if e < x then e::x::xs
             else x :: (place e xs)

let rec is_elem e l = match l with
  | [ ] -> false
  | x::xs -> e = x || (e > x && is_elem e xs)

let rec sorted l = match l with
  | [ ] -> true
  | x::[] -> true
  | x1::x2::xs -> x1 <= x2 && sorted (x2::xs)
```

Using induction, show that

```
sorted l => sorted (place e l)
```

Your proof must explicitly and clearly indicate the base case you prove, the inductive case you prove and what the inductive hypothesis provides in the proof.
Each step in your proof must be accompanied by a justification describing why that step could be taken.

Question 7: Sorted lists
Considering the same functions for ordered lists in the previous question, recall our proof that

```
is_elem e (place e l)
```

Explain why we could make this claim even though the argument does not require that the list `l` be sorted but `is_elem` seems to assume that the list is sorted.

Is the premise `sorted l` needed in the proof for question 6? Explain why or why not.

Submission instructions: Writing proofs such as these requires a bit of clear thinking and it is important to check your work.

Checking your work means you need to be able to read it. And to assess it, we need to be able to read it as well.

Thus, we are requiring your solutions be electronically generated. You may turn your work in using any of the following forms.

1. A PDF file - named `hwk_03.pdf`.

   You may use LaTeX, enscript, or even MS Word to generate a PDF file that contains your solutions.

2. A Markdown file - named `hwk_03.md`.

   This is used for the lab and other homework specifications and makes it easy to see your solution in GitHub.

3. A text file - named `hwk_03.txt`.

   We've written proofs in text files in class and examples can be found in the `Notes` directory of the public class repository.

This work is to submitted via GitHub in a folder named `Hwk_03`.
The work is due by noon on Wednesday, March 1.