

# COMP3400

## Assignment 2 Written

Paul Vrbik

April 9, 2022

### Tail Recursion

Consider a function that takes the *average* of a list of numbers:

```
average :: (Fractional a) => [a] -> a
> average [1, 2, 3]
2.0
```

This will be a *par* value to avoid this and define

<https://eduassistpro.github.io/>  
Add WeChat edu\_assist\_pro

Doing so makes this problem easier (though it will irritate t

#### Question 1. [3 MARKS]

Define a *primitive recursive* of average:

```
p_average :: (Fractional a) => [a] -> a
```

#### Question 2. [3 MARKS]

Define a *tail recursive* helper function

```
h_average :: (Fractional a) => a -> a -> [a] -> a
```

that finds the average of the list.

#### Question 3. [1 MARK]

Define average via a single call to h\_average.

**Question 4.** [1 MARK]

Define an iteration invariant for `h_average` that proves the correctness of `average`.

**Question 5.** [5 MARKS]

Prove `h_average` satisfies your iteration invariant.

**Question 6.** [1 MARK]

State the bound value for `h_average`.

**Question 7.** [2 MARKS]

Prove your bound value is always non-negative and decreasing.

**Question 8.** [2 MARKS]

Define *two* distinct quick-checks for `average` that *both* use lists from `Arbitrary [Float]`.

In particular, your quick-checks should be for lists of *arbitrary length*.

**Higher Order**

Consider the higher-order function `foo`

```
foo :: (t -> Bool) -> (t -> a) -> (t -> t) -> t -> (a)
foo p h t x
  | p x      = []
  | otherwise = h x : foo p h t (t x)
```

**Question 9.** [3 MARKS]

Define `takeWhile` by equating it to a single invocation of `foo`.

**Question 10.** [3 MARKS]

Define `map` by equating it to a single invocation of `foo`.

**Question 11.** [3 MARKS]

Define `iterate` by equating it to a single invocation of `foo`.

## Induction

### Question 12. [8 MARKS]

Consider the following definitions for implementing addition on natural numbers.

```
1 data Nat = Zero | Succ Nat deriving Show
2 plus :: Nat -> Nat -> Nat
3 plus m Zero      = m
4 plus m (Succ(n)) = plus (Succ m) n
```

Further consider this *embedding* which maps each Nat to a unique integer.

```
5 emb :: Nat -> Integer
6 emb Zero      = 0
7 emb (Succ n) = 1 + emb n
```

Using induction prove:

where + is the addition

When justifying your steps use the line numbers on this page

*Hint:* Do induction over n while letting m be free.

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro