Vivian Do
CSC 4330
Sep 11 2023

Homework 1

1. Consider the function

```
f (x:xs) = x
```

What is its type? (5 points). Note: that "type" has a formal meaning in Haskell, that is, there is only one correct (and very precise) answer

What does this function "f" do, in plain language? (5 points). For example, this function "multiplies two numbers", or "concatenates two lists", etc.

Note: that function "f" corresponds (roughly) to a standard prelude function. Mentioning this standard prelude function is an alternative way to answer this question about what "f" does

Hint: put the definition of "f" into a Haskell script and check the type with the ":type" command, and then, to understand what it does, run it with GHCi to see how it behaves

```
f :: [a] -> a
```

Function “f” takes the list as the input and returns the first item of the list as the output.

2. Consider the function

```
g [] = 0
g (x:xs) = 1 + g xs
```

What is its type? (5 points). What does this function "g" do, in plain language? (5 points)

Note: again, that function "g" corresponds (roughly) to some standard prelude function. Mentioning this standard prelude function is an alternative way to answer this question about what "g" does

```
g :: Num p => [a] -> p
```

Function “g” calculates the length of the list.

3. Consider the function

```
h x 0 = 1
h x n = x * h x (n-1)
```

What is its type? (5 points). What does this function "h" do, in plain language? (5 points)

Note: again, that function "h" corresponds (roughly) to some standard prelude function --- one could argue that there are three candidates for such function, in fact. Mentioning one of these three standard prelude functions is an alternative way to answer this question about what "h" does

```
h :: (Eq t, Num t, Num p) => p -> t -> p
```

Function “h” calculates the power of the number “x” through multiplication.

4. Consider the function

second xs = head (tail xs)

What is its type? (5 points)

Explain why "second" has the type that it has, i.e., "tail :: [a] -> [a]", that is, the type of "tail" is a mapping from type "[a]" to type "[a]", while "head :: [a] -> a"..., so why does chaining them together (or composing them) like in the definition of "second" give us the type that it has? (5 points)

second :: [a] -> a

“second” takes the list and returns the second item on the list as the output.
“tail” is used to remove the first item of the list and returns the rest back and “head” is used to return the first item from the list.

5. Consider the function

swap (x,y) = (y,x)

What is its type? (5 points). In general, why can "x" and "y" correspond to different types, i.e., why don't they have the same (polymorphic) type? (5 points)

swap :: (b, a) -> (a, b)

“x” and “y” correspond to different types due to polymorphism; Haskell does not need both elements to have the same type.

6. Consider the function

pair x y = (x,y)

What is its type? (5 points). In general, why can "x" and "y" correspond to different types here, as well? (5 points)

pair :: a -> b -> (a, b)

“x” and “y” correspond to different types due to polymorphism.

7. Consider the function

double x = x*2

What is its type? (5 points). Note that there are particular type class constraints on "x" and the return type of "double" --- why is this? (5 points)

double :: Num a => a -> a

“double” has a class constraint due to being an arithmetic operation; it can calculate results through multiplication.

8. Consider the function

palindrome xs = reverse xs == xs

What is its type? (5 points). Note the particular type class constraints on "xs" --- why is this? (5 points)

palindrome :: Eq a => [a] -> Bool

“xs” has a class constraint due to the function “palindrome” using the operator “”==” to check if the lists are the same.

9. Consider the function

twice f x = f (f x)

What is its type? (5 points). What constraints does "twice" place on the type that function "f" can be? (5 points).

Hint: try seeing what happens when we call "twice head [1,2,3]" in GHCi.

twice :: (t -> t) -> t -> t

“twice” has constraints on function “f” due to “f” taking the input of one specific type and returning the results of that same type.

10. What is the type of

[tail, init, reverse]

and why? (10 points)

[[a] -> [a]]

This is a list of functions with different types that when called on would take a list and return another list of the same type.