CS4450 Lecture 11 Notes

Wednesday, September 12, 2018 3:03 PM

Lecture 10 was the TA office hours.

Code will be posted on Canvas today regarding the different data types shown in class.

HW2 will be out tomorrow!!!

HW2 will be a similar flavor to HW1. Around 10 problems, the first 8 of which will be fairly elementary, where the last two will be slightly harder.

The list definitions will be the main focus.

GHCi is basically a fancy calculator.

Type errors are your friends. Inputting 99 + "Hey" will throw an error that the types do not match.

GHCi commands are listed on the slides. You can also use :h for help.

doubleMe :: Num a => a -> a

doubleMe is of type a from the Num class.

The empty list has no items in it (denoted as []). The list of the empty list (denoted as [[]]) has one item, which is an empty list. The list [[], [], []] is a list that contains three empty lists.

<u>List operations</u>

Head of a list: **head** takes a list and returns its head. The head of a list is it's first element.

```
head :: [a] -> a
head [] = error "List is empty."
head (x:xs) = x -- x is the guy on the front of the list, so just
return it. Also, since we don't use xs on this line, we can replace it
with a wildcard pattern. The reason for this is that it's kind of a way
of saying that you're not using the tail of the list. It's more or less
just a comment for your program for others to recognize. Some say that
it's more efficient because under the hood, you're not binding xs to
anything. The GHCi compiler will realize that you're not using xs and
will throw the code away.
tail :: [a] -> [a]
tail [] = error "List is empty"
tail (x:xs) = xs
```

Use !! to get an item from a list at a given index. E.g. "Steve Buscemi" !! 6 will return B (lists start at 0 in Haskell)

Way 1 to write a list:

```
(!!) :: [a] -> Int -> a
[] !! i = error "Empty!"
(x:xs) !! i = if i == 0 then x else xs !! (i - 1)
```

Way 2 to write a list:

```
(!!) :: [a] -> Int -> a
[] !! i = error "Empty!"
(x:xs) !! i | i==0 = x -- This structure is called a "guard", it's like
a case statement and is syntactic sugar.
| i>0 = xs !! (i-1)
| otherwise = error "Empty!"
```

More basic functions on lists: **last** takes a list and returns its last element.

How to write **last**:

```
last :: [a] -> a
last [] = error "Error!"
-- last (x:[]) = x -- We split the first clause into two, this is one
way of doing it.
-- last (x:xs) = last xs
-- Here's another way to do it:
last (x:xs) = case xs of
[] -> x
_ -> last xs
```

null is a built in function that we can use in Haskell.

How do we implement **reverse**?

```
-- This one is correct but inefficient:
reverse :: [a] -> [a]
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
```

take takes a number and a list. It extracts that many elements from the beginning of the list.

What is the type of **take**?

Type Int.

How do we write **take** in Haskell?

```
take :: Int -> [a] -> [a]
take 0 _ = []
take n _ = []
take n (x:xs) = x : take (n-1) xs -- This says take (n-1) of xs BUT
hold on to x.
```