LING 185A: Section 2

Thursday, October 9

Reminders

- **Homework #1** is due at **11:59 PM on Friday, October 10**.
  - Submit as a **PDF** on **BruinLearn**.
- Keep posting questions on Campuswire!

Wisdom

1. Writing a function? Start with a lambda: `\x -> ...`
2. Don't know how to use `x`? Pattern match: `case x of { ... }`
3. Got an `f :: a -> b`? And an `x :: a`? Apply the function: `... f x ...`

These will take you far.

Practice Recursion$^{\text{Recursion}^{\text{Recursion}^{\cdot^{\cdot}}}}$

`numbToInt :: Numb -> Int`

`numbToInt n` evaluates to the nonnegative integer represented by `n`:

`numbToInt Z` $\Longrightarrow^*$ `0`
`numbToInt (S (S (S Z)))` $\Longrightarrow^*$ `3`

```
numbToInt :: Numb -> Int
```

`lessThanOrEq :: Numb -> (Numb -> Bool)`

`lessThanOrEq m n` evaluates to `True` if and only if `numbToInt m` is less than or equal to `numbToInt n`.

`lessThanOrEq (S (S Z)) (S (S (S Z)))` $\Longrightarrow^*$ `True`

```
lessThanOrEq :: Numb -> (Numb -> Bool)
```

`greaterThan :: Numb -> (Numb -> Bool)`

`greaterThan m n` evaluates to `True` if and only if `lessThanOrEq m n` evaluates to `False`.

`greaterThan Eq (S (S Z)) (S (S (S Z)))` $\Longrightarrow^*$ `False`

```
greaterThan :: Numb -> (Numb -> Bool)
```

`bigger :: Numb -> (Numb -> Numb)`

`bigger m n` evaluates to the larger of `m` and `n`.

`bigger (S (S Z)) (S Z)` $\Longrightarrow^*$ `S (S Z)`

```
bigger :: Numb -> (Numb -> Numb)
```

`isSorted :: (a -> (a -> Bool)) -> ([a] -> Bool)`

`isSorted cmp l` evaluates to `True` if and only if the list `l` is sorted according to the ordering `cmp`.

`isSorted lessThanOrEq [Z, Z, S (S Z), S (S (S Z))]` $\Longrightarrow^*$ `True`

`isSorted :: (a -> (a -> Bool)) -> ([a] -> Bool)`

`remove :: (a -> Bool) -> ([a] -> [a])`

`remove pred` removes every `x` from its list argument such that `pred x` $\Longrightarrow^*$ `True`.

`remove (lessThanOrEq (S Z)) [Z, Z, S (S Z), S (S (S Z))]` $\Longrightarrow^*$ `[Z, Z]`

`remove :: (a -> Bool) -> ([a] -> [a])`

`map :: (a -> b) -> ([a] -> [b])`

`map f` applies `f` to every element of its list argument.

`map numbToInt [Z, Z, S (S Z), S (S (S Z))]` $\Longrightarrow^*$ `[0, 0, 2, 3]`

```
map :: (a -> b) -> ([a] -> [b])
```

`(++) :: [a] -> ([a] -> [a])`

`x ++ y` evaluates to the concatenation of the lists `x` and `y`.

`[Z, Z, S (S Z)] ++ [S Z, Z]` $\Longrightarrow^*$ `[Z, Z, S (S Z), S Z, Z]`

```
(++) :: [a] -> ([a] -> [a])
x ++ y =
```