COMP105 Lecture 9

Mutual and Multiple Recursion

Mutual recursion

Mutual recursion is when two functions call each other

```
even' 0 = True
even' n = odd' (n-1)

odd' 0 = False
odd' n = even' (n-1)
```

We have to make sure that

- We terminate in a base case
- We always make progress towards a base case

Mutual recursion

Getting the even indexed elements from a list with mutual recursion

```
evens [] = []
evens (x:xs) = x : odds xs

odds [] = []
odds (x:xs) = evens xs
```

```
ghci> evens "abcdefg"
"aceg"
```

We could have also done this with one function

- Mutual recursion is a stylistic choice

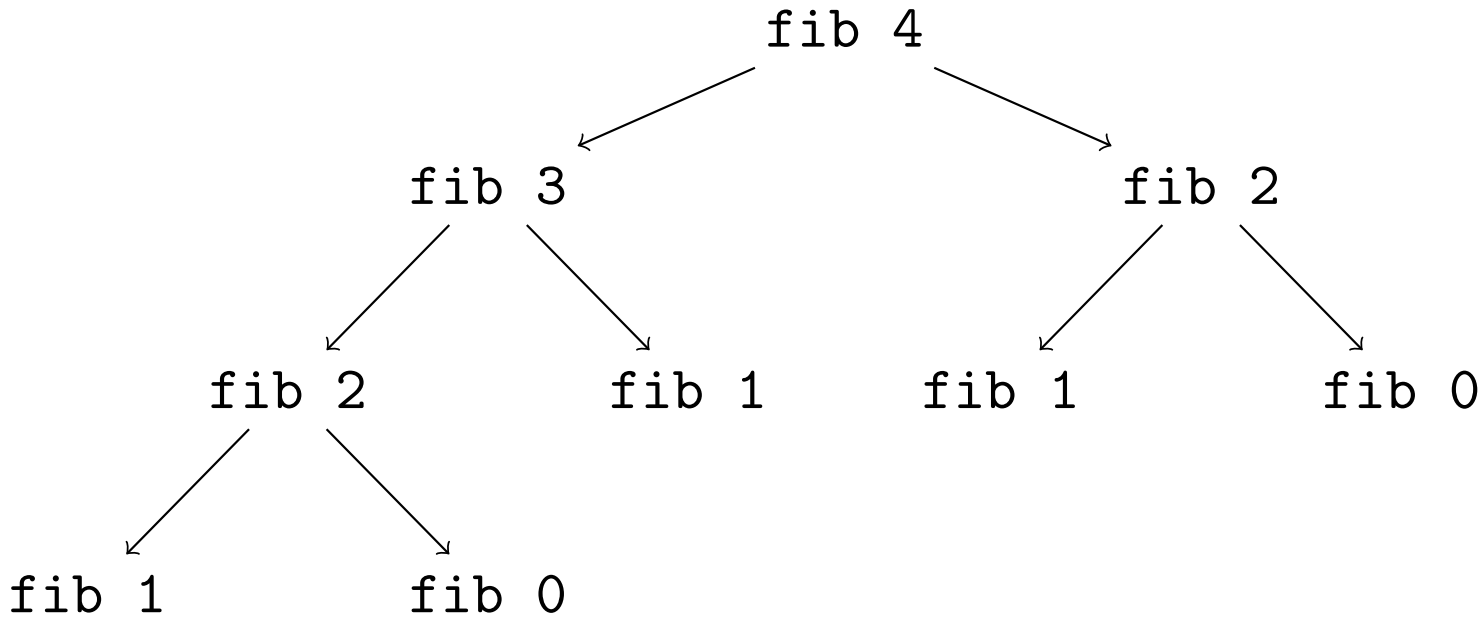
Multiple recursion

Multiple recursion is when a function makes more than one recursive call in the same recursive rule

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Multiple recursion can make your code slow...

Multiple recursion



Each base case is called many times

- So very inefficient
- Already `fib 40` takes a huge amount of time

A faster `fib`

Create a **helper function** that computes the fibonacci list:

```
fast_fib_help 1 = [1, 0]
fast_fib_help n = x + y : (x:y:xs)
    where (x:y:xs) = fast_fib_help (n-1)
```

```
ghci> fast_fib_help 10
[55,34,21,13,8,5,3,2,1,1,0]
```

```
fast_fib n = head (fast_fib_help n)
```

Exercises

1. Use mutual recursion to write a function `multipleThree` that takes one number `x` and returns `True` when `x` is divisible by 3, and `False` otherwise. Hint: you will need to write three different functions.

2. The lucas numbers are defined so that $L_0 = 2$, $L_1 = 1$, and $L_n = L_{n-1} + L_{n-2}$. Use mutiple recursion to write a function `lucas` that takes one argument `n` and returns L_n