CS558 Programming Languages – Fall 2023 – Suggested Study Question Solutions for Lecture 6b

1. If `g` is nested within `f`, then its scope is limited to `f` (including perhaps other nested functions defined in `f`). Then there are only a limited number of ways in which `g` can be called:

- `g` can be called directly by its name from within the body of `f`. In this case, `g` returns before `f` returns.
- `g` can be called directly by its name from within some function `h` defined locally in `f` (where `h` might be `g` itself, if `g` is recursive). But by an inductive argument, we can reason that `h` also cannot be called after `g` returns, so its call to `g` cannot be after `f` returns either.
- Since downward funargs are allowed, `g` can be passed by `f` (or some `h`, as above) as an argument `p` to some other function `r`, and `r` can then call `g` indirectly under its alias `p`. But since `r` must return before `f`, this call to `g` must also occur before `f` returns.
- Since *only* downward funargs are allowed, there is no other way for `g` to escape from `f`: it cannot be returned or stored in a global variable that might be accessed after `f` returns.

2. (a)

```
case class Map[A,B](f: FCF[A,B]) extends FCF[List[A],List[B]]{
  def apply (xs:List[A]) : List [B] = {
     def g(xs:List[A]) : List[B] = xs match {
        case Nil => Nil
        case (y::ys) => f.apply(y)::g(ys)
     }
     g (xs)
   }
}

def pow(n:Int, b:Int) : Int =
  if (n == 0) 1 else b * pow (n-1,b)

case class Pow(n:Int) extends FCF[Int,Int] {
  def apply (b:Int) = pow(n,b)
}

val v = Map.apply(Pow(3)).apply(List(1,2,3))
```

(b)

```
case class Compose[A,B,C] (f: FCF[B,C], g: FCF[A, B]) extends FCF[A,C] {
  def apply (x:A) : C = f.apply(g.apply(x))
}

val h = Compose(MkFCF((x:Int) => x> 3),MkFCF((y:Int) => y * 2))
```

3. (a)

```
def fac (n:Int) : Int = {
    def factcps(n:Int,k:Int => Int) : Int =
      if (n < 2)
        k(n)
      else factcps(n-1, v => k(n*v))
    factcps(n, v => v)
  }
```

(b)

```
  def fib (n:Int) : Int = {
    def fibcps(n:Int,k:Int => Int) : Int =
      if (n < 2)
        k(1)
      else fibcps(n-1, v1 => fibcps (n-2, v2 => k (v1+v2)))
    fibcps(n, v => v)
  }
```