CS558 Programming Languages – Fall 2023 – Study Questions Lecture 4c

These questions are intended for self-study, to help review and deepen your understanding of the lecture. Sample answers are available. There is nothing to hand in.

1. Which of the following Scala functions are tail-recursive? For those that are, give an equivalent function using iteration and no recursion.

```
def s(x:Int) : Int = {
  if (x == 0)
    0
  else
    1 + s(x-1)
}

def t(x:Int) : Int = {
  if (x == 0)
    0
  else
    t(x-1) + 1
}

def even(x:Int) : Boolean = {
  if (x == 0)
    true
  else if (x == 1)
    false
  else even(x-2)
}

def fac(x:Int) : Int = {
  if (x < 2)
    1
  else
    x*fac(x-1)
}

def facn(x:Int,y:Int) : Int = {
  if (x < 2)
    y
  else
    facn(x-1,x*y)
}

def fib(x:Int) : Int = {
  if (x < 2)
```

```
    x
  else
    fib(x-1) + fib(x-2)
}

def g(x:Int) : Int = {
  if (x < 0)
    g(-x) * 2
  else if (x > 0)
    g(x/2)
  else 0
}

def h(x:Int) : Int = {
  if (x == 0)
    0
  else
    h(h(x-1))
```

2. (a) Recall the (non-tail-recursive) definition of the factorial function from a previous lecture:

```
def fac(x:Int) : Int = {
  if (x < 2)
    1
  else
    x*fac(x-1)
}
```

Use the recursion removal techniques described in lecture to convert this to a non-recursive Scala function using an explicit stack. It is easiest to develop a solution by first temporarily pretending that you have labels and gotos available in Scala, and then converting the resulting code to use structured control operators like `while`.

It is convenient to `import scala.collection.mutable.Stack`, ignoring the deprecation warning.

(b) Do the same with the Fibonacci function:

```
def fib(x:Int) : Int = {
  if (x < 2)
    x
  else
    fib(x-1) + fib(x-2)
}
```

Note: This example is much more complicated! It is best to start by making intermediate results and control flow as explicit as possible.

```
def fib (n: Int) : Int = {
  var r = 0
  if (n < 2)
    r = n
  else {
    val t1 = fib(n-1)
    val t2 = fib(n-2)
    r = t1 + t2
  }
  return r
}
```

In general, each entry in the explicit stack needs to encode which recursive call is to be "returned to," and the values of any variables that are defined before the call and used after it. For `fib`, there are two "return points" corresponding to the two recursive calls. We need to remember the value of `n` over the first call, and the value of `t1` over the second call. A suitable Scala data type for stack elements is therefore:

```
sealed abstract class C
case class C1(val n : Int) extends C
case class C2(val t1 : Int) extends C
```