Foundations of Functional Programming 2004
Paper 6 Question 10 (ACN)

```
    val k = ref 0;
    fun c(n, r) = (
        k := !k + r;
        if r=0 orelse r=n then 1
        else c(n-1,r-1) + c(n-1,r));
```

```
I will give every adjusted function 2 extra args. One will
be the continuation it needs, which I will usually call x.
The other will be k and gives the way that the "state" can
be passed around. Note that this means that continuations
will need to be passed k as well as a "return value".

fun c(x,k,n r) = if r = 0 orlse r=n then x(k+r,1)
  else c(XXX, k+1, n-1,r-1)

where XXX(k, v1) =
   c(YYY, k, n-1, r)   (* I need to write this within the
   def of c so I can access n and r... *)

where YYY(k, v2) = x(k, v1+v2) (* must be nested where it
   can access x and v1... *)

If I make XXX and YYY just lambda-expressions written where
they are used the scope issue sort themselves out neatly.


Something like "A handle x=>B" might have "raise x" within
A. With continuations what you do is to represent the
exception x as a continuation that just does B and follows on
with the outer continuation. Raise is just done by calling that
unusual continuation instead of the regualr one within A.
```