Solution notes

Foundations of Functional Programming 2005 – Paper 6 Question 10 (ACN)

(*a*) give reduction rules associated with the standard combinators `S`, `K` and `I`. [4 marks]

```
I x -> x
K x y -> x
S f g x -> f x (g x)
```

(*b*) Explain as simple a procedure as possible that converts lambda expressions into terms that only use these combinators. At this stage you are not expected to be concerned with efficiency, but you should make it clear why your combinator forms relate to the lambda-expressions that you start with. [5 marks]

```
Start with an smallest innermost lambda expression, and perform the
rewrite
    \ x . x -> I
    \ x . y -> K y   (y differs from x)
    \ x . f g -> S (\ x . f) (\ x . g)

This EITHER makes the smallest included lambda smaller (via the
rule that introduces S) OR it reduces the number of lambdas. Thus
it makes progress and will eventually finish with no lambdas left.

Each rewrite above is easy to see as leading to something that is
extensionally equal to where you started.
```

(*c*) Illustrate the scheme you have given above by applying it to the lambda-expression $\lambda x.\lambda y.((y\ x)\ y$ and explain why in practical reduction to combinators at least additional symbols `B` and `C` are normally introduced. [5 marks]

```
   \ x . \ y . ((y x) y)
   \ x . S (\ y . (y x)) (\ y . y)
   \ x . S (S I (K x)) I

Now to deal with \ x.
   S (\ x . (S (S I (K x)) (\ x . I)
It is NICE here to permit the extensional rule \ x . K x -> K. I can
slog through using just S but it becomes jolly tedious! But the good
news is I could do it in fragments
   \ x . (K x)
   S (\ x . K) (\ x . x)
   S (K K) I
```

```
and
   \ x . (S (S I (K x)))
   S (\ x . S) (\ x . (S I (K x)))
   S (K S) ... again concentrate just on what is left

   \ x . (S I (K x))
   S (\ x . (K I)) (\ x . (K x))
   S (K (K I)) (S (K K) I)

So even if all done in full gore it is OK in the time allowed.

But with B and C and a bit of optimision I can go
   \ x . (S (S I (K x)))
   B S (\ x . (S I (K x))
   B S (B (S I) K)
which was a lot quicker and easier.
```

(d) Suppose that you have primitive functions available that can represent integers, booleans, arithmetioc and an "if...then" operation (and so on, as necessary). Show how to convert the following ML-like code first into raw lambda calculus and then into combinator form:

```
fun f n = if (iszero n) 1
             (double (f (subtract1from n)));
```

Note that the code has been presented in a way intended to show the functions used to perform aritrhmetic. You may assume the availability of a `Y` operator if that helps.

[6 marks]

```
fun f n = if (iszero n) 1
             (double (f (subtract1from n)));

f = Y \ f . \ n . if (iszero n) 1 (dbl (f (sub1 n)))
Look at the \n . sub-term first

S (\ n . (if (iszero n) 1))
  (\ n . (dbl (f (sub1 n))))

S (C (B if iszero) 1)
  (B dbl (B f (B sub1)))

So that bit was easy!  Now I just need to stick \ f . back
on the front.
```

```
B (S (C B if iszero) 1)
  (B (B dbl) (C B (B sub1)))
```

and for the final result I just need Y of that.