# G6021: Comparative Programming

Exercise Sheet 7

## 1 Types

What are the types of the following? In each case, try to work out the type first, then check with Haskell. (Note: at least one of these does not have a type!)

```
i x = x
k x y = x
zero f x = x
one f x = f x
two f x = f(f x)
three f x = f(f(f x))
s x y z = x z (y z)
w x y = x y y
d x y = x x y
newi = s k k
fib n x y = if n==1 then x else fib (n-1) (x+y) x
fib2 (n,x,y) = if n==1 then x else fib2 (n-1, x+y, x)
```

**Answer:** d does not have a type. The types of the rest are:

```
i :: t -> t
k :: t -> t1 -> t
zero :: t -> t1 -> t1
one :: (t -> t1) -> t -> t1
two :: (t -> t) -> t -> t
three :: (t -> t) -> t -> t
s :: (t -> t1 -> t2) -> (t -> t1) -> t -> t2
w :: (t -> t -> t1) -> t -> t1
newi :: t -> t
fib :: (Num a, Num a1) => a -> a1 -> a1 -> a1
fib2 :: (Num t, Num t1) => (t, t1, t1) -> t1
```

## 2 Fixpoints

1. Write the function fix from the notes in Haskell.

   **Answer:**

   ```
   fix f = f(fix f)
   ```

2. Write factorial as a functional fact, without recursion (as given in the notes). Test that

   ```
   fix fact 4
   ```

   computes factorial of 4 as expected.

**Answer:**

```
fact = \f -> \x  -> if x==0 then 1 else x*f(x-1)
```

3. Experiment with fix and other functions.

## 3 Comparison with Java

1. Haskell (and functional languages in general) are very good as sophisticated calculators. To demonstrate this, compute the following:

   ```
   67^457
   ```

   **Answer:**

   ```
   3282349590901342216621455638945446839267811092749563948417371881487113
   6533219838605595229721418046192627674009529642292232329555852538522734
   8901773813311026429261717697005894523767169633873076206916377965162487
   0756337072544686031982779280507403608407927134431857885077808925541577
   6376495838844975108723527246344495950054360097058174676501245058907443
   9590872184927795939470004611206950268779866100592825778072407589907879
   2433505576226083995044063427927436380705534039913055415017799606603072
   3145940352545412968484442619252471830641151899091634112663881371160900908
   2310857459960921700540415602048196227506812181190199807271127494532751
   8807952777337620399416424718159173641496047412230503890164590170824421
   3470431598083896993234476142860850375597312442873616290671173839933500
   88294201102260931501295187571854945422189510143875512479118395427
   ```

   How would you go about doing this is Java? (You don't have to write any Java, but think about what you would need to do to be able to write it.)

   **Answer:** Take a look at `java.math.BigInteger`. However it is much more complicated to perform the above calculation.

2. In Haskell we can create a new data-type together with functions that work over the new data-type such as this:

   ```
   data Shape = Square Float | Circle Float
                deriving (Show)
   area (Square x) = x*x
   area (Circle r) = pi*r*r
   ```

   We can test with:

   ```
   area (Square 2)
   area (Circle 4)
   ```

   In Java we can represent this by letting `Square` and `Circle` be subtypes of an abstract type `Shape`. Write Java code to do the above example, and compare the code with Haskell.

   **Answer:**

```
abstract class Shape {}
class Circle extends Shape {
    double r;

    Circle(double r) {this.r = r;}

    double area() {
       return 3.1425*r*r;
    }
}
class Square extends Shape {
    double l;
    Square(double l) {this.l = l;}

    double area() {
       return l*l;
    }
}
class Test {
    public static void main (String[] args) {
        Circle c = new Circle(2);
        Square s = new Square(4);
        System.out.println(c.area());
        System.out.println(s.area());
    }
}
```

3. Suppose in you wanted to write a function/method to return two values (say a pair of integers). Outline how you would do this in both Haskell and Java.

   **Answer:** Haskell has pairs (and triples, etc.) built-in, so there is nothing new needed for Haskell. For Java, we need to create a new type, for example at least this is needed:

```
class Pair {
  int fst, snd;
}
```