5. (a) A *for loop* is an iterative construct which from a given initial state repeatedly applies a given action for each index in a range of integers. In Haskell we can declare the name and type of a *for loop* as follows.

```
for :: Int -> Int -> (a -> a) -> (a -> a)
```

That is, `for n m f s` repeatedly applies a function `f` starting from an initial state `s` and an implicit *counter* initially valued `n`, incrementing the counter by `1` until it exceeds `m`. Give a definition in Haskell for the *for loop* `for`. [6]

(b) *For loops* are found in many programming languages, being very efficient for simple iteration. Using your Haskell `for` loop define the function `mult :: Int -> Int` which multiples $1 \times 2 \times \ldots \times n$ for any given integer $n \geq 1$. For example, `mult 4` evaluates to `24`. [6]

(c) A key role for higher order functions in functional programming is to design language constructs which may not be provided by your favourite language. For example, there are many variations on a *for loop* in different languages. Design a generalisation of `for` called `forge` which adds another Curried argument to specify what function is to be used to increment the counter. For example, `forge` with increment function `(+ 1)` is in effect `for`. [8]