G6021: Comparative Programming

Exercise Sheet 2

1 Function arguments

If a function needs two arguments, it can take them one at a time or both together, as illustrated by the following functions:

```
f x y = x+y
g (x,y) = x+y
```

We say that `f` is a *curried* version of `g`.

1. What are the types of `f` and `g`? Test the above two function definitions, including the type definition of each function.
2. Consider a function that returns the sum of three integer arguments. Try to write four alternative versions that take arguments in different ways (think about currying, and mixing styles). What are the types of your functions?
3. Write a function that takes four arguments. The body of the function can be anything, but must use the 1st argument once, the 2nd argument twice, the 3rd argument once and ignores the 4th argument. What is the type of this function?
4. Two built-in functions provided by the Haskell system are:

   ```
   curry :: ((a, b) -> c) -> a -> b -> c
   uncurry :: (a -> b -> c) -> (a, b) -> c
   ```

   By looking carefully at the types of the functions `f` and `g` above, show how you can convert `f` to take one argument (a pair of numbers) and convert `g` to take 2 arguments (one at a time).
5. Write your own version of `curry` and `uncurry`.

2 Lists in Haskell

1. Enter `[1,2,3,4,5]`, and find out the type of this expression.
2. Find two other ways to write (or generate) the list `[1,2,3,4,5]`.
3. What is generated by `[x | x<-[1..], x<6]`? Think about it before testing with the interpreter.
4. Write a function `inorder` that will test if a list of integers is sorted in ascending order. Hint: The preferred solution would use pattern matching. Some examples of patterns are:
   - `[]` : the empty list.
   - `[x]` : a list with exactly 1 element (alternative notation: `x:[]`).

- `(x:t)` : a list with at least one element.
- `(x:y:t)` : a list with at least two elements, etc.

5. Write a function `insert` that takes an integer `x` and a sorted list, and returns the sorted list with `x` inserted in the correct place.

   Example: `insert 3 [1,2,4,5] = [1,2,3,4,5]`

6. Using the previous function or otherwise, write a sort function.

   Example: `sort [4,3,2,1] = [1,2,3,4]`

3 If you have time...

1. Write a function that returns the first n elements of a list (you can assume that the list is longer than n). Examples:

   ```
   first 3 [9,2,4,7,6,5] = [9,2,4]
   first 0 [1,2,3,4,5,6,7] = []
   first 10 [1..] = [1,2,3,4,5,6,7,8,9,10]
   ```

   (Hint: use the first parameter to countdown.)

2. Write a function `filt` that takes a number n and a list of numbers, and returns the list with all the multiples of n removed. You can make use of a built-in function `mod` that gives the remainder after division (`mod 2 2 = 0`, `mod 3 2 = 1`).[1]

   Examples:


   ```
   filt 2 [1,2,3,4,5,6] = [1,3,5]
   filt 1 [1,2,3,4] = []
   filt 3 [1..10] = [1,2,4,5,7,8,10]
   ```


3. Starting with the list `[2..]`, we can generate a list of prime numbers by repeatedly doing the following two steps:

   (a) Keep the first element of the list (say h) as this is a prime number.

   (b) Delete all multiples of h from the remaining list.

   Write a function `primes` that will generate all the prime numbers. Using your function `first` test your algorithm to produce the first 10, 100, 1000, etc. primes.

[1] `2 `mod` 2` is a way to use this function infix rather than prefix.