F28PL - Programming Languages

Question B (Recursion, chaining functions)

Tommy Lamb

H00217505

The plain text file which accompanies this PDF contains a listing of all functions alongside test function calls, with some formatting to aid readability. It can be opened and executed in Poly/ML from the Terminal by navigating to the save location of the file, opening Poly/ML, and using the command : ' use "F28PL - Tommy Lamb - Question B - Code.txt"; '. This method will cause Poly/ML to ignore the document formatting and only display system output (not input). To maintain formatting, all text can be copied and pasted into a running instance of Poly/ML. For best results it should be copied/pasted in small blocks, around 1 question in size.

Note:
Many functions here are computationally dangerous for certain arguments. Red text highlights these inputs in each answer. No such inputs feature in the provided txt file.

1. Find the integer logarithm to the base 2 of an integer n by repeatedly dividing by 2 and counting:

```
log2 1 = 0
log2 n = 1+log2 (n div 2)
- log2 8;
> 3 : int
```

- `fun log2 1 = 0 | log2 n = 1 + log2 (n div 2);`

  Recursively divides number n by 2 until the result is 1. Returns the number of recursive calls made. Pattern matching provides the base case, which returns 0. Moving up the call stack each function call adds 1 to the return value of the immediately lower recursive call to eventually total the height of the call stack i.e. the number of recursive calls before reaching the base case.

  When $n \leq 0$, this function will run indefinitely and rapidly consume system memory.

2. Find the square root of an integer x by repeatedly decrementing an initial guess s:

```
sqrt x s = s if s*s <= x
sqrt x s = sqrt x (s-1) if s*s > x
- sqrt 10 5;
> 3 : int
```

- `fun sqrt x s = if s*s <= x then s else (sqrt x (s-1));`

  Recursively decrements the value s, evaluates s^2 with respect to x, and ultimately returns the largest value of s for which $s \leq s_0$ and $s \leq \sqrt{x}$, where s_0 is the original value of s.
  If the initial value of the guess s, is lower than the square root of x, then s is just returned, else a value for s is returned as above

3. Find the sum of the squares of the first n integers: $n^2 + (n-1)^2 + (n-2)^2 + \ldots 1^2$
   The sum of the squares of the first 0 integers is 0.
   The sum of the squares of the first n integers is n squared plus the sum of the squares of the first n-1 integers.

```
- sumSq 3;
> 14 : int
```

- `fun sumSq 0 = 0 | sumSq n = (n*n) + (sumSq (n-1));`

  Squares the given value n and then adds the result of recursive calls on n−1, such that the result is n^2 + sumSq (n−1) + sumSq (n−2)...+ SumSq 0. Which equates to n^2 + $(n-1)^2$ + $(n-2)^2$...+ 1^2 + 0

  When $n < 0$, this function will run indefinitely and rapidly consume system memory.

4. Find the sum of the halves of the first n integers $\frac{n}{2} + \frac{(n-1)}{2} + \frac{(n-2)}{2} \ldots + \frac{1}{2}$

```
- sumHalf 6;
> 9 : int
```

- `fun sumHalf 0 = 0 | sumHalf n = (n div 2) + (sumHalf (n - 1));`

  This function, like the previous function, acts recursively to decrement the given value n and returns the sum of the result of each function call. In this function the operation performed on n for each call is integer division by 2. While this function satisfies the given test case it does not truly operate as the question text specifies, as the automatic rounding introduces significant mathematical error. Ignoring the test case and satisfying solely the textual specification yields this function:

  ```
  fun realSumHalf 0 = 0.0 | realSumHalf n = ((real n) / 2.0) + realSumHalf (n-1);
  ```

  By returning a `real` rather than an `int` this function avoids the rounding error and provides a mathematically accurate result, though this does not satisfy the given test case. I include this only so that I have a correct answer regardless of how the question as a whole is interpreted.

  When $n < 0$, these functions will run indefinitely and rapidly consume system memory.

5. Find the sum of applying function f to the first n integers.
   The sum of applying f to the first 0 integers is 0.
   The sum of applying f to the first n integers is f applied to n plus the sum of applying f to the first n-1 integers.

```
- fun inc x = x+1;
> val inc = fn : int -> int
- sumF inc 3;
> 9 : int i.e. inc 3+ inc 2+ inc 1+ 0;
```

- `fun sumF f 0 = 0 | sumF f n = (f n) + (sumF f (n-1));`

  Following the pattern of the previous two functions, this recursively decrements the value n and returns the sum of the repeated application of some function, f, to n. Unlike the earlier functions this one does not 'hard-code' the function f, rather it accepts it as an argument.

  When $n < 0$, this function will run indefinitely and rapidly consume system memory.

6. Write sumSq using sumF.

- `fun sumSq2 n = sumF (fn x => x*x) n;`

  Utilising the previous sumF function for recursion, this function simply calls sumF with the argument `(fn x => x*x)`, which is logically equivalent in result to `(n*n)` when $x = n$. Thus it is functionally identical to the original implementation sumSq. The difference here is that the function is more abstract and has higher cohesion (to steal an object-orientated phrase), as it uses an independent function to handle the addition and recursion.

  When $n < 0$, this function will run indefinitely and rapidly consume system memory.

7. Write `sumHalf` using sumF.

- `fun sumHalf2 n = sumF (fn x => x div 2) n;`

  Exactly as in Q6, this method passes a function to sumF to produce its output. The function passed is again logically equivalent to that hard-coded in the original implementation of `sumHalf`. This method implements the version of `sumHalf` specified by the provided test case, as `realSumHalf` would require a different version of sumF than has been defined.

  When $n < 0$, this function will run indefinitely and rapidly consume system memory.