Lab – 3

Week of Jan. 21, 2018

1 Instructions

Use the OCaml top-level to develop and debug your code. Please note that *input validation* is a must for all problems even if not explicitly stated. In questions that require string outputs, be careful not to include leading or trailing whitespace.

The following submission files are *required*:

1. `leap_year.ml`
2. `middle_child.ml`
3. `p_checker.ml`
4. `square_root.ml`
5. `fizzbuzz.ml`

2 Assignments

1. **Leap Year Checker** (submission file: `leap_year.ml`)

   Write a function `isLeapYear: int -> bool` that accepts an year argument and decides if it is a leap year or not. A year is a leap year if it is divisible by 4 but not divisible by 100 unless it is also divisible by 400. Examples:

   (a) `isLeapYear 2004`, `isLeapYear 2016` and `isLeapYear 2000` return `true`.

   (b) `isLeapYear 2017`, `isLeapYear 2018` and `isLeapYear 1900` return `false`.

2. **Middle Child Syndrome** (submission file: `middle_child.ml`) When a family has three children, a child is born in between is known as the middle child. A stereotype known as "Middle Child Syndrome" says that the middle child has trouble sharing – the oldest is bossy, everyone babies the baby, and the middle child is, well, stuck in the middle.

   (a) Define a function `middleChild : int -> int -> int -> int` that accepts the ages of three children, and returns the age of the middle child. If there isn't a middle child (for example), in the case of twins,

return -1, and in case of triplets, you should return -2. And in case of erroneous inputs (can you think of invalid inputs?), return -3. Examples:

i. `middleChild 17 12 15` returns 15.

ii. `middleChild 3 3 5` returns -1.

iii. `middleChild 12 12 12` returns -2.

(b) Define a function `print_middle_child: int -> int -> int -> string` which takes in the ages of the three children and ~~prints~~ returns the age of the middle child as `"The age of the middle child is: <age>"` if there is a middle child. Otherwise, it ~~prints~~ returns `"There are twins!"` or `"There are triplets"` if the set of children has twins or triplets. In case of erroneous inputs, ~~print~~ return `"Invalid inputs!"`.

3. **Primality Checker** (submission file: `p_checker.ml`)

Write a function `isPrime: int -> bool` to return whether an integer x is a prime or not. (Hint: It is sufficient to test divisibility by numbers upto $\sqrt{x}$). Examples:

(a) `isPrime 25` returns `false`.

(b) `isPrime 97` returns `true`.

(c) `isPrime 1` returns `false`.

4. **Square root by Newton's method** (submission file: `square_root.ml`): In order to compute the square root of a number a, we can start with some approximate value x and refine it as follows:

$$y = \frac{x + \frac{a}{x}}{2}$$

For example, if $a = 4$ and our initial estimate is $x = 3$, then

$$y = \frac{3 + \frac{4}{3}}{2} = 2.1666$$

We can repeat the process with y as our new estimate. Continuing the above example, we get

$$y' = \frac{2.1666 + \frac{4}{2.1666}}{2} = 2.0064$$

Now, repeat again with y' as our estimate until there is no change in the value computed.

Implement a function `square_root: float -> int -> float` that takes in a floating point number a, an integer k corresponding to the number of times to repeat the estimation procedure and returns the square root. If $k \leq 0$, then the function is expected to run until there is no change in the estimated value. Use the initial estimate of 1.0 for all cases.

Examples:

(a) `square_root 4 2` should return `2.05` (as the initial estimate is 1.0)

(b) `square_root 4 0` should return `2.0`.

5. **Fizzbuzz** (submission file: `fizzbuzz.ml`)
   Fizzbuzz is a group word game for children to teach them about division. Players take turns to count incrementally, replacing any number divisible by three with the word "Fizz", and any number divisible by five with the word "Buzz", and any number divisible by both as "Fizzbuzz".

   For example, a typical round of fizz buzz would start as follows: "One... Two... Fizz... Four... Buzz... Fizz... Seven ..... (a long time later) .... Fourteen... Fizzbuzz... Sixteen...".

   (a) Define a function `fizzbuzz : int -> string` that accepts an integer and returns the correct way to call this number - the output should be "Fizz", "Buzz", "Fizzbuzz" or the number itself.

      i. `fizzBuzz 17` returns `"17"`.
      ii. `fizzBuzz 18` returns `"Fizz"`.
      iii. `fizzBuzz 20` returns `"Buzz"`.
      iv. `fizzBuzz 30` returns `"Fizzbuzz"`.

      Note that the checker is case-sensitive to the output.

   (b) Define a function `fizzbuzz_string : int -> string` that accepts an integer and returns a sequence in a Fizzbuzz game (you should use the function `fizzbuzz` defined above).

   (c) Examples:

      i. `fizzbuzz_string 4` returns `"1 2 Fizz 4"`
      ii. `fizzbuzz_string 10` returns `"1 2 Fizz 4 Buzz 6 7 8 Fizz Buzz"`
      iii. `fizzbuzz_string 15` returns `"1 2 Fizz 4 Buzz 6 7 8 Fizz Buzz 11 Fizz 13 14 Fizzbuzz"`