Assignment 1
COMP 302 Winter 2020
Programming Languages and Paradigms

Prakash Panangaden
McGill University: School of Computer Science

Due Date: 17th January 2020

Please do all four questions using the LearnOCaml system as explained in class. In addition to the solution you must provide test cases so that we can see how thoroughly you have tested your program. We will evaluate the quality of your test cases by running mutated programs on your test cases and see if it catches the errors. Test cases are to be written as a list of `(input,output)` pairs. The lists of test cases will be declared for you in the LearnOCaml template; your task is to populate the lists with the necessary test cases.

There is a fifth question which you don't have to do or even look at. This is a so-called "spiritual growth" question. It will not earn you any extra credit for the course.

Q1. [14 points] On the LearnO'Caml system we have installed *incorrect* code for the following functions:

```
val double : int -> int = <fun>
val fact : int -> float = <fun>
```

The function "double" is supposed to take a non-negative integer $n \geq 0$ and return its value doubled. Its type is as shown above. The function "fact" is supposed to take a non-negative integer $n \geq 0$ and return $n!$ as a floating-point number. We did this to give you some experience with changing types; there is no good mathematical reason for doing this[1]. The implementations we have provided may have logical, mathematical or syntactic errors. Please fix them.

Q2. [28 points] Here is a way to compute the square root of a positive real number say x. We make a wild guess that the square root is some number g. Then we check if g^2 is close to x. If it is we return g as our answer; if not we refine our guess using the formula

$$g' = (g + x/g)/2.0$$

where g' is the new guess. We keep repeating until we are close enough. Write an OCaml program to implement this idea. Since we are working with floating point numbers we cannot check for equality; we need to check if the numbers are close enough. I have written a function called `close`

[1] Unless you are extending the factorial to non-integer values through the Gamma function.

for you as well as a function called `square`. In OCaml floating point operations need special symbols: you need to put a dot after the operation. Thus you need to write, for example, $2.0 +. 3.0$. All the basic arithmetic operations have dotted versions: $+., *., /.$ and must be used with floating point numbers. You cannot write expressions like $2 + .3.5$ or $2.0 + 3.0$. We will not test your program on negative inputs so we are not requiring you to deal with the possibility that you may get a negative answer. There are of course built-in functions to compute square roots. We have written the tester program to check if you used this; **you will get a zero if you use a built-in function for square roots.**

Q3. [28 points] This is similar to the question above except that we are computing cube roots. The formula to refine the guess is

$$g' = (2.0 * g + x/g^2)/3.0$$

where I am using mathematical notation with arithmetic operation symbols overloaded. In your code you must use the floating-point versions of the arithmetic operations.

Q4. [30 points] In lecture 1, week 2, I quickly explained the Russian peasant algorithm for fast exponentiation. In this exercise you have to implement a tail-recursive version of the algorithm. Everything is with integers in this question so please do not use floating point operations. Our tester will detect whether you used built-in functions, so please don't try to use them.

Q5. [0 points] This question is for your spiritual growth only. Do not think it will give you extra credit or help you learn the material better. It will however stretch your brain in other directions. Do not attempt it if you have not yet finished the required homework. Do not submit a solution; please talk to me if you have solved it. Do not worry about it if you don't understand the question.

Why do the algorithms for square root and cube root above converge? Will this kind of idea work for *any* function? What is the correct update formula for fifth roots?