COMP 302 Programming Languages and Paradigms
Assignment 4

Prakash Panangaden
McGill University: School of Computer Science

Due Date: 7^{th} February 2020 3pm

There are 4 questions to be solved on the LearnOCaml system. The assignment is due at 3pm. Extensions are not possible for any reason.

[Question 1: **23 points**] In this question we work with polymorphic trees

```
type 'a tree = Empty | Node of 'a tree * 'a * 'a tree;;
```

Write a function `mapTree` that takes a function and a tree and applies the function to every node of the tree. Here is the type

```
val mapTree : ('a -> 'b) * 'a tree -> 'b tree = <fun>
```

Here is an example

```
# let t1 = Node(Empty, 0, (Node(Empty, 1, Empty)));;
let t2 = Node(Node(Empty,5,Empty),6,Empty);;
let t3 = Node(Node(t2,2,Node(Empty,3,Empty)),4,t1);;
# t3;;
- : int tree =
Node
 (Node (Node (Node (Empty, 5, Empty), 6, Empty), 2, Node (Empty, 3, Empty)),
 4, Node (Empty, 0, Node (Empty, 1, Empty)))
# mapTree((fun n -> n + 1), t3);;
- : int tree =
Node
 (Node (Node (Node (Empty, 6, Empty), 7, Empty), 3, Node (Empty, 4, Empty)),
 5, Node (Empty, 1, Node (Empty, 2, Empty)))
```

[Question 2: **34 points**] Suppose that $f : \mathbf{R} \to \mathbf{R}$ is a *continuous* function from the real numbers to the real numbers. Suppose further that there is an interval $[a, b]$ in which the function is known to have *exactly one root*[1]. We will assume that[2] $f(a) < 0$ and $f(b) > 0$ and the root r is somewhere in between. There is an algorithm that is reminiscent of binary search that finds (a

[1]The root of a function f is a value r such that $f(r) = 0$.

[2]When I am writing mathematical statements I do not bother to write 0.0; when I am writing code I am careful to distinguish between 0 and 0.0.

good approximation of) the root. The *half-interval method* for finding the root of a real-valued function works as follows. The idea is to search for the root by first guessing that the root is the midpoint of the interval. If the midpoint is the root we are done. If not, we check whether the value of f at the midpoint is positive or negative. If it is positive then the root must be between a and the midpoint; if it is negative, the root must be between the midpoint and b. In either case we recursively call the search algorithm. Code this algorithm in OCaml. The following shows the names and types that we expect.

```
# let rec halfint((f: float -> float),
                  (posValue:float),(negValue:float),(epsilon:float)) = ....
  val halfint : (float -> float) * float * float * float -> float = <fun>
```

The parameter `epsilon` is used to determine the accuracy with which you are making comparisons. Recall that you cannot *reliably* test floating-point numbers for equality. You may need to use `Float.abs` which is the floating point version of the absolute value function.

[Question 3: **33 points**] This is a classic example of a higher-order function in action. Newton's method can be used to generate approximate solutions to the equation $f(x) = 0$ where f is a differentiable real-valued function of the real variable x. The idea can be summarized as follows: Suppose that x_0 is some point which we suspect is near a solution. We can form the linear approximation l at x_0 and solve the linear equation for a new approximate solution. Let x_1 be the solution to the linear approximation $l(x) = f(x_0) + f'(x_0)(x - x_0) = 0$. In other words,

$$\begin{aligned} f(x_0) + f'(x_0)(x_1 - x_0) &= 0 \\ x_1 - x_0 &= -\frac{f(x_0)}{f'(x_0)} \\ x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} \end{aligned}$$

If our first guess x_0 was a good one, then the approximate solution x_1 should be an even better approximation to the solution of $f(x) = 0$. Once we have x_1, we can repeat the process to obtain x_2, etc. In fact, we can repeat this process indefinitely: if, after n steps, we have an approximate solution x_n, then the next step is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

This will produce approximate solutions to any degree of accuracy provided we have started with a good guess x_0. If we have reached a value x_n such that $|f(x_n)| < \varepsilon$, where ε is some real number representing the tolerance, we will stop.

Implement a function called `newton` with the type shown below. The code outline is for your guidance.

```
# let rec newton(f, (guess:float), (epsilon:float), (dx:float)) =
  let close((x:float), (y:float), (epsilon:float)) = abs_float(x-.y) < epsilon in
  let improve((guess:float),f,(dx:float)) = .... in
  if close((f guess), 0.0, epsilon)
  then
```

```
    guess
  else
    ....
val newton : (float -> float) * float * float * float -> float = <fun>
```

which when given a function f, a guess x_0, a tolerance ε and an interval dx, will compute a value x' such that $|f(x')| < \varepsilon$. You can test this on built-in real-valued functions like `sin` or other functions in the mathematics library. Please note that this method does not always work: one needs stronger conditions on f to guarantee convergence of the approximation scheme. Never test it on *tan*! Here are two examples:

```
let make_cubic((a:float),(b:float),(c:float)) =
fun x -> (x*.x*.x +. a *. x*.x +. b*.x +. c)
let test1 = newton(make_cubic(2.0,-3.0,1.0),0.0,0.0001,0.0001)
(* Should get -3.079599812; don't worry if your last couple of digits are off. *)
let test2 = newton(sin,5.0,0.0001,0.0001)
(* Should get 9.42477.... *)
```

[Question 4: **10 points**] In class we say how to define a higher-order function that computes *definite* integrals of the form

$$\int_b^a f(x)\mathrm{d}x.$$

To remind you

```
let integral((f: float -> float),(lo:float),(hi:float),(dx:float)) =
  let delta (x:float) = x +. dx in
  dx *. iterSum(f,(lo +. (dx/.2.0)), hi, delta);;
```

where `iterSum` was also defined in class. In this question I want you to define a function that computes the *indefinite integral*:

$$indIntegral(f) = \int_0^x f(y)\mathrm{d}y.$$

This means that it returns a function not a number. Here is the type that I expect

```
# let indIntegral(f,(dx:float)) = ...
val indIntegral : (float -> float) * float -> float -> float = <fun>
```

Use any of the functions defined in class. We will preload `iterSum` and `integral` so you won't have to recode them. **You do not have to do test cases for the last question.**