CIS 120 Quiz 3

February 4–5, 2015

Name: 

PennKey (e.g. mkizner): ___________________________ Section: ___________________________

Indicate the section you’re registered for, even if you’re attending a different section.

type ‘a tree =
   | Empty
   | Node of ‘a tree * ‘a * ‘a tree

let rec tree_max (t: ‘a tree) : ‘a =
   begin match t with
   | Empty -> failwith "tree_max: no maximum in empty tree"
   | Node (_, x, Empty) -> x
   | Node (_, _, rt) -> tree_max rt
   end

let rec delete (n: ‘a) (t: ‘a tree) : ‘a tree =
   begin match t with
   | Empty -> Empty
   | Node (lt, x, rt) ->
     if x = n then
       begin match lt, rt with
       | Empty, Empty -> Empty
       | Node _, Empty -> lt
       | Empty, Node _ -> rt
       | _ ->
         let m = tree_max lt in
         Node (delete m lt, m, rt)
       end
     else if n < x then Node (delete n lt, x, rt)
     else Node (lt, x, delete n rt)
   end
1. The following code segment uses the definition of generic binary search trees and of the binary search tree delete function, reproduced on the front side of the page.

```ocaml
let t = Node (Node (Empty, 7, Empty), 8, Node (Node (Empty, 16, Empty), 15, Node (Empty, 17, Empty))) in
let r = delete 16 t
```

Does \( r \) satisfy the binary search tree invariant?  

☐ Yes  ☐ No

2. Write the type of each of the following OCaml expressions in the blank provided, or ill-typed if the expression does not type check.

(a) let a : ________________________________ = [[[]]; [[]]]
(b) let b : ________________________________ = Node (Empty, 3, Empty)
(c) let c : ________________________________ = Node (Empty, b, Empty)
(d) let d : ________________________________ = delete b 3
(e) let e : ________________________________ = delete false
(f) let f : ________________________________ = (fun x l -> x :: l)
(g) let g : ________________________________ = (fun f -> f 42)

3. As you learned in lecture, OCaml functions are “first-class” values, so they can be treated like any other values. In particular, it’s possible to construct a list containing function values. Here’s an example:

```ocaml
let l : (int -> int) list = [fun x -> x + 2; fun x -> 3 * x]
```

Implement a function `apply_all` which takes an integer \( x \) and a list of \( \text{int -> int} \) functions, like \( l \) above, and returns the result of applying the integer to each function in order. For example, `apply_all 1 l` should return 9, while `apply_all 4 l` should return 18. If the list is empty, `apply_all` should just return \( x \).

```ocaml
let rec apply_all : int -> (int -> int) list -> int =
  fun x l ->
```