CS 496: Homework Assignment 2
Due: 7 March, 11:55pm

1 Assignment Policies

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

2 Assignment

This assignment involves implementing a series of operations on general trees. A *general tree* is a non-empty tree in which each node can have any number of children. An example of a general tree is given in Fig. 1. This general tree is a general tree of integers since the nodes hold integers. Note that not all nodes have the same number of children. For example, node 33 has two children, but node 7 has three children.

General trees may be modeled in OCaml by means of the following algebraic data type:

```
type 'a gt = Node of 'a*('a gt) list
```

Thus a general tree is a node that has a data value of type `'a` and a list of children (each of which are general trees). The simplest example of a general tree is a leaf. For example,

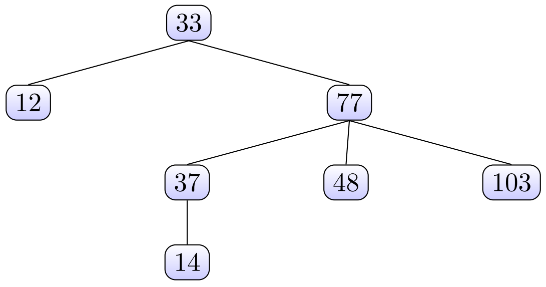

Figure 1: Sample value of type `int gt`

the expression `Node(12,[])`, of type `int gt`, is a leaf that holds the integer 12. The following expression `t`, of type `int gt`, represents the tree depicted in Fig. 1:

```
let t : int gt =
 Node (33,
       [Node (12,[]);
        Node (77,
              [Node (37,
                     [Node (14, [])]);
               Node (48, []);
               Node (103, [])])
       ])
```

Here is a sample function that given `n` builds a general tree that is a leaf holding `n` as data.

```
let mk_leaf (n:'a) : 'a gt =
  Node(n,[])
```

3 Operations to be Implemented

Please implement the following operations:

1. `height`: that given a general tree returns its height. The height of a tree is the length of the longest (in terms of number of nodes) path from the root to a leaf. Eg.

```
# height t;;
- : int = 4
```

2. `size`: that given a general tree returns its size. The size of a general tree consists of the number of nodes.

```
# size t;;
- : int = 7
```

3. `paths_to_leaves t`: returns a list with all the paths from the root to the leaves of the general tree `t`. Let n be the largest number of children of any node in `t`. A path is a list of numbers in the set $\{0, 1, \ldots, n-1\}$ such that if we follow it on the tree, it leads to a leaf. The order in which the paths are listed is irrelevant. Eg.

```
# paths_to_leaves t;;
- : int list list = [[0]; [1; 0; 0]; [1; 1]; [1; 2]]
```

4. `is_perfect`: that determines whether a general tree is *perfect*. A general tree is said to be *perfect* if all leaves have the same depth. Eg.

```
# is_perfect t;;
- : bool = false
```

5. `preorder`: that returns the pre-order traversal of a general tree. The pre-order traversal of a general tree is the same as that of a binary tree. Eg.

```
# preorder t;;
- : int list = [33; 12; 77; 37; 14; 48; 103]
```

6. `mirror`: that returns the mirror image of a general tree. Eg.

```
# mirror t;;
- : int gt =
Node (33,
 [Node (77, [Node (103, []); Node (48, []); Node (37, [Node (14, [])])]);
  Node (12, [])])
```

7. `mapt f t`: that produces a general tree resulting from `t` by mapping function `f` to each data item in `d`. Eg.

```
mapt (fun i -> i>20) t;;
- : bool gt =
Node (true,
 [Node (false, []);
  Node (true,
   [Node (true, [Node (false, [])]); Node (true, []); Node (true,
[])])])
```

8. `foldt f t`: that encodes the recursion scheme over general trees. Its type is

```
foldt: ('a -> 'b list -> 'b) -> 'a gt -> 'b
```

For example, here is how one may define `sumt` and `memt` using `foldt`:

```
let sumt t =
  foldt (fun i rs -> i + List.fold_left (fun i j -> i+j) 0 rs) t

let memt t e =
  foldt (fun i rs -> i=e || List.exists (fun i -> i) rs) t
```

For example,

```
# sumt t ;;
- : int = 324
# memt t 12;;
- : bool = true
# memt t 33;;
- : bool = true
# memt t 35;;
- : bool = false
```

9. Implement `mirror'` using `foldt`. It should behave just like Exercise 6.

4 Submission instructions

Submit a file `hw2.ml` through Canvas. Only one member of a the group submits via Canvas (on behalf of both). The name of the other member must be placed as a Canvas comment. Also, the name of both members of the group have to be placed in the source code.

Each exercise is worth ten points, except the last two which are worth 15 points each.