ACCESSING MODULES AND SUBMODULES (15/15 points)

1. Use fully-qualified names to fix the compilation of [bfs]. (The `open` directive is forbidden here.)

THE GIVEN PRELUDE

```
module Tree = struct

  type 'a t = Leaf of 'a | Node of 'a t * 'a * 'a t

  module Iterator = struct

    type 'a path =
      | Top
      | Left of 'a path * 'a * 'a t
      | Right of 'a t * 'a * 'a path

    type 'a iterator = Loc of 'a t * 'a path

    exception Fail

    let go_left (Loc (t, p)) =
      match p with
        | Top -> raise Fail
        | Left (father, x, right) -> raise Fail
        | Right (left, x, father) -> Loc (left, Left (father, x, t))

    let go_right (Loc (t, p)) =
      match p with
      | Top -> raise Fail
      | Left (father, x, right) -> Loc (right, Right (t, x, father))
      | Right (left, x, father) -> raise Fail

    let go_up (Loc (t, p)) =
      match p with
        | Top -> raise Fail
        | Left(father, x, right) -> Loc (Node (t, x, right), father)
        | Right(left, x, father) -> Loc (Node (left, x, t), father)

    let go_first (Loc (t, p)) =
      match t with
        | Leaf _ -> raise Fail
        | Node (left, x, right) -> Loc (left, Left (p, x, right))

    let go_second (Loc (t, p)) =
      match t with
        | Leaf _ -> raise Fail
        | Node (left, x, right) -> Loc (right, Right (left, x, p))

    let focus (Loc ((Leaf x | Node (_, x, _)), _)) = x

  end

end
```

YOUR OCAML ENVIRONMENT

```
6           let results = (Tree.Iterator.focus l) :: results in
7           try
8             aux results (ls @ [Tree.Iterator.go_first l; Tree.Iterator.go_second l])
9           with Tree.Iterator.Fail ->
10            aux results ls
11    in
12    aux [] [Loc (t, Top)]
13  ;;
```

Switch >>

Typecheck

Reset Templ

Full-screen |

Check & Sa

Exercise complete (click for details) — 15 pts

v Exercise 1: no open — Completed, 15 pts

Bravo, I did not find any open — 5 pts

Now, I will check that your function actually works

Found `bfs` with compatible type.

Computing `bfs (Tree.Node (Tree.Leaf 'i', 'k', Tree.Leaf 'q'))`

Correct value `['k'; 'i'; 'q']` — 1 pt

Computing

```
bfs
  (Tree.Node (Tree.Leaf 'n', 'h',
    Tree.Node (Tree.Node (Tree.Leaf 'b', 'i', Tree.Leaf 'q'), 'b',
     Tree.Leaf 'y')))
```

Correct value `['h'; 'n'; 'b'; 'i'; 'y'; 'b'; 'q']` — 1 pt

Computing

```
bfs
  (Tree.Node (Tree.Leaf 'v', 'x',
    Tree.Node (Tree.Leaf 'c', 'n',
     Tree.Node (Tree.Leaf 'x', 'm', Tree.Leaf 'x'))))
```

Correct value `['x'; 'v'; 'n'; 'c'; 'm'; 'x'; 'x']` — 1 pt

Computing `bfs (Tree.Leaf 'd')`

Correct value `['d']` — 1 pt

Computing `bfs (Tree.Leaf 'r')`

Correct value `['r']` — 1 pt

Computing `bfs (Tree.Node (Tree.Leaf 'k', 'j', Tree.Leaf 'r'))`

Correct value `['j'; 'k'; 'r']` — 1 pt

Computing `bfs (Tree.Leaf 'b')`

Correct value `['b']` — 1 pt

Computing `bfs (Tree.Node (Tree.Leaf 'p', 'g', Tree.Leaf 'w'))`

Correct value `['g'; 'p'; 'w']` — 1 pt

Computing `bfs (Tree.Leaf 'g')`

Correct value `['g']` — 1 pt

Computing

```
bfs
  (Tree.Node (Tree.Node (Tree.Leaf 't', 'x', Tree.Leaf 'o'), 'w',
    Tree.Leaf 'r'))
```

Correct value `['w'; 'x'; 'r'; 't'; 'o']` — 1 pt