Three F# Exercises

This assignment consists of three F# exercises. The code for all exercises are included in the single file, `hw4.fsx`.

For each of the exercises, you are only required to write the provided functions. There is no additional global functionality. Your functions should produce no output. To test your functions, either:

- Test the functions using the interactive simulator
- Add your own tests to the script (but be sure to remove them or comment them out before submission)

Implementation Rules

- You may define and use additional functions if your feel it is appropriate.
- Without prior approval from me, you may only use the subset of F# described in class.
  - In particular, you may NOT use F# system functions (such as min) or methods
  - Even though they were mentioned in class, you are NOT allowed to use the methods in the List module such as `List.map`.

First Exercise, `maxCylinderVolume`

Write a function maxCylinderVolume that takes a list of floating-point tuples that represent dimensions of a cylinder and returns the volume of the cylinder that has the largest volume. Each tuple has two floating point values that are both greater than zero. The first value is the radius *r* and the second value is the height *h*. The volume of the cylinder is computed using $\pi r^2 h$. The value π is represented in F# with `System.Math.PI`. If the list is empty, return 0.0.

Examples:

```
> maxCylinderVolume [(2.1, 3.4); (4.7, 2.8); (0.9, 6.1); (3.2, 5.4)];;
val it : float = 194.3137888
> maxCylinderVolume [(0.33, 0.66)];;
val it : float = 0.2257988304
```

Second Exercise, `elimDuplicates`

Write a function `elimDuplicates` that takes a list of integers and eliminates consecutive duplicates; replacing them with a single instance of the value. Order is preserved and non- consecutive duplicates are unaffected.

Examples:

```
> elimDuplicates [1; 2; 2; 3; 3; 3; 4; 4; 4; 4; 5; 5; 5; 5; 5];;
val it : int list = [1; 2; 3; 4; 5]
> elimDuplicates [1; 2; 2; 1; 3; 3; 1; 4; 4; 1; 5; 5; 1];;
val it : int list = [1; 2; 1; 3; 1; 4; 1; 5; 1]
```

Third Exercise, `BST`

Write the following binary search tree functions for a binary search tree of integers. Use the following type definition for a BST (copy this into your solution):

```
// Tree definition for problem 3
type BST =
    | Empty
    | TreeNode of int * BST * BST
```

A. `insert value tree`: Inserts the value into the tree and returns the resulting tree. The resulting tree does NOT need to be balanced. If the value already exists in the tree, return the tree without inserting the value.

B. `search value tree`: Returns true if the value is in the tree and false otherwise.

C. `count func tree`: The parameter `func` is a Boolean function that takes a single parameter and returns true or false. The function tests the value of each node with func and returns the number of nodes that evaluate to true.

D. `evenCount tree`: Returns the number of nodes that contain even integers. REQUIREMENT: This function must be a single call to `count` (part 3C) using a lambda function.

Examples:

```
> let bt1 = insert 10 Empty;;
val bt1 : BST = TreeNode (10, Empty, Empty)
> let bt2 = insert 5 bt1;;
val bt2 : BST = TreeNode (10, TreeNode (5, Empty, Empty), Empty)
> let bt3 = insert 3 bt2;;
val bt3 : BST =
  TreeNode (10, TreeNode (5, TreeNode (3, Empty, Empty), Empty), Empty)
> let bt4 = insert 17 bt3;;
val bt4 : BST =
  TreeNode
    (10, TreeNode (5, TreeNode (3, Empty, Empty), Empty), TreeNode (17, Empty, Empty))
> let bt5 = insert 12 bt4;;
val bt5 : BST =
  TreeNode
    (10, TreeNode (5, TreeNode (3, Empty, Empty), Empty),
     TreeNode (17, TreeNode (12, Empty, Empty), Empty))
> search 17 bt5;;
val it : bool = true
> search 4 bt5;;
val it : bool = false
> evenCount bt5;;
val it : int = 2
```

Guidance Note:

Wanted to clarify that your program will not be penalized if the compiler emits warnings and your program works. F# will warn you anytime a run-time error is possible and, due to limitations in its analysis, will emit warnings in cases where a run-time error is not possible.

One situation occurs is when using this statement:

```
let hd::tl = list
```

You'll get a warning indicating that incomplete pattern matching occurs on this expression and mentions the [] (empty list) case. It is true that you will get a runtime error if this statement executes when list is empty. You should check your code to make sure it is handled. However, even if you guarantee that list is not empty, the compiler will still emit the warning. The only way to avoid the warning is to use pattern matching and extract hd and tl using one of the patterns.

> **HW4 Rubric**

Criteria	Ratings							Pts
maxCylinderVolume	**12 pts** **All tests pass**	**10 pts** **1 test fails**	**8 pts** **2 tests fail**	**6 pts** **3 tests fail**	**4 pts** **4 tests fail**	**2 pts** **5 tests fail**	**0 pts** **6 or more tests fail**	12 pts
elimDuplicates	**12 pts** **All tests pass**	**10 pts** **1 test fails**	**8 pts** **2 tests fail**	**6 pts** **3 tests fail**	**4 pts** **4 tests fail**	**2 pts** **5 tests fail**	**0 pts** **6 or more tests fail**	12 pts
insert	**8 pts** **All tests pass**	**6 pts** **1 test fails**	**4 pts** **2 tests fail**	**2 pts** **3 tests fail**	**0 pts** **4 or more tests fail**			8 pts
search	**8 pts** **All tests pass**	**6 pts** **1 test fails**	**4 pts** **2 tests fail**	**2 pts** **3 tests fail**	**0 pts** **4 or more tests fail**			8 pts
count	**5 pts** **All tests pass**	**3 pts** **1 test fails**	**1 pts** **2 tests fail**	**0 pts** **3 or more tests fail**				5 pts
evenCount	**5 pts** **All tests pass**	**3 pts** **1 test fails**	**1 pts** **2 tests fail**	**0 pts** **3 or more tests fail**				5 pts
Additional Deductions	**0 pts** **Additional deductions** Will be represented using negative points.				**0 pts** **No additional deductions**			0 pts
Total Points: 50								