Comp Sci 321: Programming Languages — Homework 1

1 Set up DrRacket & Create Handin Account

Be sure you have Racket version 8.0 installed. You can download it from:
`https://download.racket-lang.org/`

Open DrRacket, set the language to "The Racket Language" (via the **Language** | **Choose Language** menu item), put the following program into the upper window (replacing whatever is there), and click "Run":

```
#lang plai
(+ 2 2)
```

You should see the prompt in the interactions window (a > character) and the number `4`.

The first line tells Racket that we'll be using the `plai` language, which is what we'll use throughout the quarter. You can find the documentation for the `plai` language here. The second line just does some computation to make sure everything is working.

2 Honor Code

Every homework assignment you hand in must begin with the following definition (taken from the Provost's website; see that for a more detailed explanation of these points):

```
(define eight-principles
  (list
   "Know your rights."
   "Acknowledge your sources."
   "Protect your work."
   "Avoid suspicion."
   "Do your own work."
   "Never falsify a record or permit another person to do so."
   "Never fabricate data, citations, or experimental results."
   "Always tell the truth when discussing your work with your instructor."))
```

If the definition is not present, you receive no credit for the assignment.

3 Trees

The following `Tree` datatype describes binary trees with interior nodes (that carry no value) as well as two kinds of leaves, one to represent positive integers, and another to represent negative integers:

```
(define-type Tree
  [positive-leaf (val natural?)]
  [negative-leaf (val natural?)]
  [interior-node (left Tree?) (right Tree?)])
```

Zero can be represented as either a positive or a negative leaf. Note that either kind of leaf can only hold `natural?` numbers, that is, integers greater than or equal to zero. Thus, a leaf holding the integer `-5` would be expressed using `(negative-leaf 5)`.

Implement a `contains?` function that takes a `Tree` as its first argument, and an integer (positive or negative) as its second, and that returns `true` if the given integer is present anywhere in the tree. It should return `false` otherwise.

Every problem must come with an appropriate set of test cases. At a minimum the test cases must cover every branch in each function you write. Here's one to get you started:

```
(test (contains? (interior-node (interior-node (positive-leaf 5)
                                               (negative-leaf 4))
                                (positive-leaf 3))
                 -4)
      true)
```

Hint: `type-case` may be useful here.

4 Smallest

Implement a `smallest` function, which takes a `Tree` as argument and returns the integer (i.e., not the node) that has the smallest (i.e., closest to negative infinity, **not** closest to zero) value.

5 Balance

We will consider a `Tree` to be balanced if the sum of the values of its leaves is zero.

Implement a `balanced?` function, which takes a `Tree` as argument and returns `true` if the tree is balanced, and returns `false` otherwise.

6 More Balance

We will consider a `Tree` to be *deeply balanced* if and only if **all** the ***interior nodes*** in the tree are balanced.

Implement a `deep-balanced?` function, which takes a `Tree` as argument and returns `true` if it is deeply balanced, and returns `false` otherwise.

7 Negation

Implement a `negate` function, which takes a `Tree` as argument, and returns a new `Tree` of the same shape, but where the value of each leaf is negated.

8 Addition

Implement an `add` function, which takes a `Tree` as its first argument and an integer as its second, and adds the integer to the value of all the leaves in the tree.

9 Transmutation

Implement a `positive-thinking` function, which takes a `Tree` as argument and produces a new tree which removes all negative leaves from the original tree. If the resulting tree would have no nodes, return `false`.

Hint: the total number of nodes (leaf and interior) in the resulting tree will be *less than or equal* to the total number of nodes in the original, minus the number of negative leaves in the original.

10 Handin

Submit your Racket source file via Canvas.

Your submission must include your test cases; submissions without test cases will get a grade of 0.