Assignment 1
COMP 302 Programming Languages and Paradigms

Brigitte Pientka
MCGILL UNIVERSITY: School of Computer Science

Due Date: 25 January 2012

Your homework is due at the beginning of class on Jan 25, 2012. All code files must be submitted electronically, and your program must compile.

Q1 20 points Warm-up.

Q1.1 10 points The average can be computed follows: $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{(x_1+\ldots+x_n)}{n}$

Write a function `average: int list -> real` which accepts a list of integers and returns their average as a real number. The average of the empty list is zero.

```
- average [1, 5, 2, 7];      - average [13, 25, 22, 27];
val it = 3.75 : real         val it = 21.75 : real
```

Q1.2 10 points The standard deviation is computed as follows: $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2}$

In other words, the standard deviation can be calculated as follows:

1. For each value x_i calculate the difference between x_i and the average value $\overline{x}$.
2. Calculate the squares of these differences.
3. Find the average of the squared differences. This quantity is the variance σ^2.
4. Take the square root of the variance.

Write a function `stDev: int list -> real` which given a list of integers, computes the standard deviation as a real. For the empty list, `stDev` returns zero.

```
- stDev [1, 5, 2, 7];             - stDev [13, 25, 22, 27];
val it = 2.38484800354 : real     val it = 5.35607132141 : real
```

Note: Pay careful attention to conversions!

Q2 10 points Implement a function `psum:int list -> int list` that takes a list and computes the partial sums, i.e. the sums of all the prefixes.

Here are some cases illustrating the behavior.

```
- psum [1,1,1,1,1];              - psum [];
val it = [1,2,3,4,5] : int list  val it = [] : int list

- psum [1,2,3,4];                - psum [9];
val it = [1,3,6,10] : int list   val it = [9] : int list
```

Q3 70 points Consider binary search trees as we discussed in class. The data-type for binary trees can be defined as follows.

```
datatype 'a bstree = Empty | Node of (int * 'a) * 'a bstree * 'a bstree
```

The nodes in a tree consist of a tuple: the key and an element. While the key is an integer, the element may be a string or anything else. This is a binary search tree, so every tree should satisfy the order invariant: For any `Node((key,el), left, right)`, all keys in the `left` subtree are smaller than `key` and all keys in the `right` subtree are greater than `key`.

In this question, we explore balanced binary search trees, more precisely AVL trees. An AVL tree (after Adelson-Velskii and Landis) is a binary search tree satisfying the following balancing condition:

for every node in the AVL tree the absolute value of the difference between height of the left subtree and the height of the right subtree should not be greater than 1, i.e. $|$`height(left) - height(right)`$| \leq 1$.

We call the difference between the height of the left subtree and the height of the right subtree the *balance factor*. In an AVL tree all subtrees have balance factor 1, 1 or 0.

Q3.1 10 points Implement a function `balanced:'a bstree -> bool` that takes a binary search tree as input and checks that it is balanced.

```
val t2 = Node ((7,"a"),Node ((3,"b"), Empty, Empty),
                       Node ((9,"c"), Empty, Empty));

val t3 = Node ((2, "f"), Empty, t2);

- balanced t3;
val it = false : bool
 - balanced t2;
val it = true : bool
```

Q3.2 20 points When we insert or delete an element in a tree, we may need to re-balance the tree. There are four possible rotations shown in Figure 1 which transform an unbalanced tree into a balanced one. Implement four functions each corresponding to one rotation.

```
rotLeft        :'a bstree -> 'a bstree
rotLeftRight   :'a bstree -> 'a bstree
rotRight       :'a bstree -> 'a bstree
rotRightLeft   :'a bstree -> 'a bstree
```

Finally, implement a function `rebalance:'a bstree -> 'a bstree` which accepts a binary tree search which has a subtree whose balance factor is 2, i.e. the tree is ill balanced, and re-balances the tree. You can assume that we do not give binary search trees who are completely degenerate, i.e. they contain subtrees whose balance factor is greater than 2.

Hint: Use the height of the subtrees to decide how to rebalance the tree.

Q3.3 15 points Implement `insert : (int * 'a) * 'a bstree -> 'a bstree`. Given a key `k`, an element `el`, and an AVL tree, this function will insert a new entry `(k, el)` into the AVL tree. The end result must be again an AVL tree, i.e. it is balanced.

Use the previously implemented function `rebalance` to ensure the tree is balanced after inserting the entry.

Q3.4 25 points Implement a function `del: 'a bstree * int -> 'a bstree` which takes an AVL tree `t` together with a key `k` and deletes the entry with key `k` from `t`. The resulting tree must still be an AVL tree, i.e. the key's of elements in the left subtree must be smaller than keys of elements in the right subtree and the balance invariant must hold. You can again use the previously implemented function `rebalance` to ensure that the tree you return is balanced.

To implement `del`, you will need an auxiliary function `delMin:'a bstree -> 'a bstree * 'a` . Given a tree `t`, the function will return the entry `e` which has the smallest key in `t` and a new tree `t'` which is the tree `t` where `e` has been removed. `t'` must be balanced.

Using the function `delMin`, implement the function `delete`.

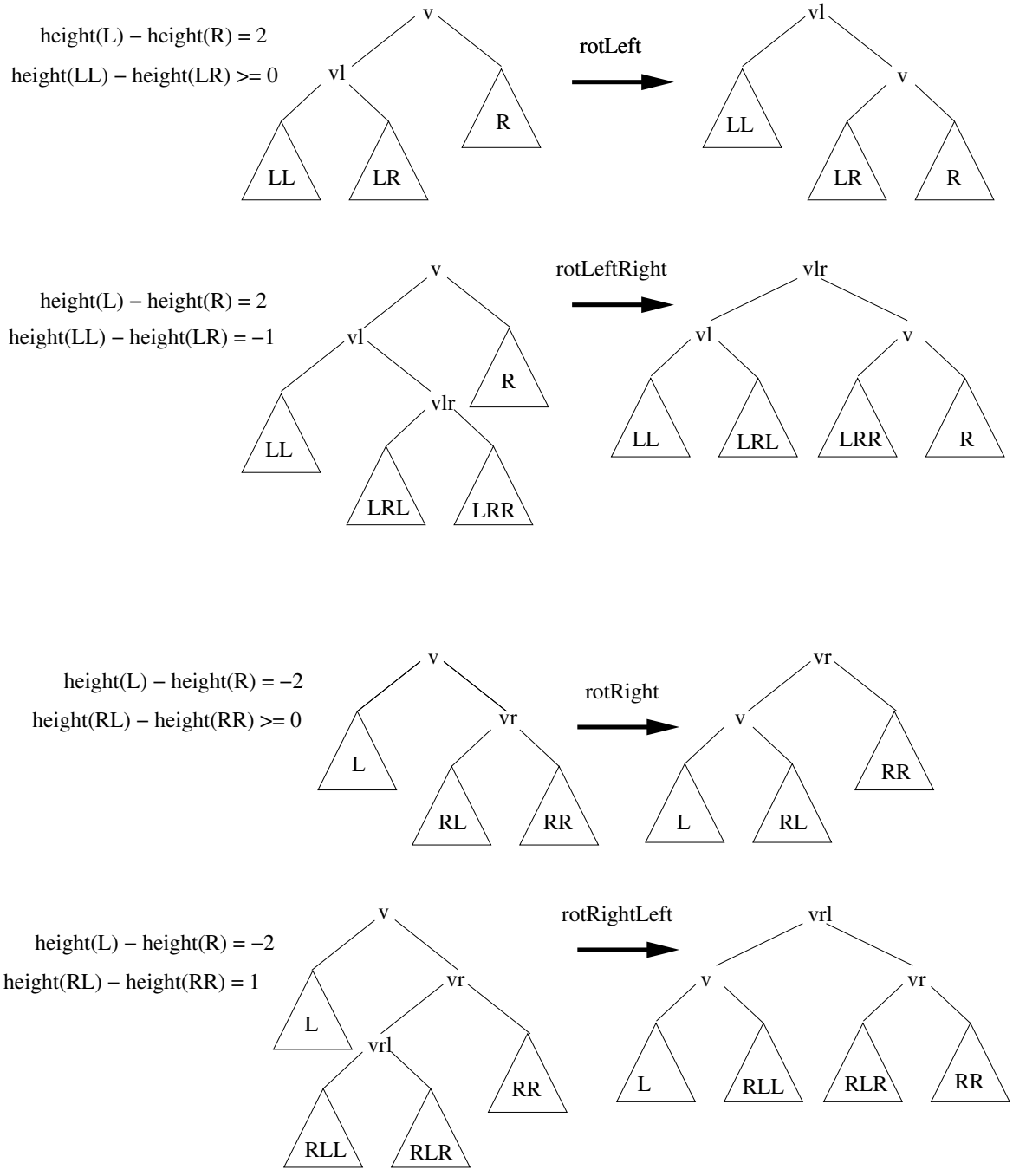

Figure 1: Rotations