

CS320 Programming Languages

Homework #1

Due: 11 September 2019

1 Primitives (20 pts)

1. Write the function `volumeOfCuboid`, which consumes three non-negative integer numbers `a`, `b`, and `c` denoting lengths of three sides and produces the volume of the cuboid. (Note: $0 \leq a, b, c \leq 1,000$)
2. Write the function `concat`, which consumes two strings `x` and `y`, and it returns their concatenation. For example, `test(concat("abc", "def"), "abcdef")`

2 Function Values (30 pts)

1. Write the function `addN`, which consumes an integer number `n` and produces a function that adds `n` to a given integer number. For example,

```
val f: Int => Int = addN(5)
test(f(3), 8)
test(f(42), 47)
```

2. Write the function `twice`, which consumes a function `f` whose type is `Int => Int` and returns another function that applies the function `f` twice. For example,

```
val g: Int => Int = twice(addN(3))
test(g(2), 8)
test(g(7), 13)
```

3. Write the function `compose`, which consumes two `Int => Int` functions `f` and `g` and returns their composition `f ∘ g`. For example,

```
val h: Int => Int = compose(addN(3), addN(4))
test(h(5), 12)
test(h(11), 18)
```

3 Data Structures (50 pts)

3.1 Lists (20 pts)

1. Define the function `double`, which consumes a list `l` of integers and returns another list whose elements are doubles of elements of `l`. For example,

```
val l: List[Int] = List(1, 2, 3)
test(double(l), List(2, 4, 6))
test(double(double(l)), List(4, 8, 12))
```

2. Define the function **sum**, which consumes a list **l** of integers and returns the sum of elements of the list **l**. For example,

```
test(sum(List(1,2,3)), 6)
test(sum(List(4,2,3,7,5)), 21)
```

3.2 Maps (10 pts)

1. Define the function **getKey**, which consumes a map **m** from strings to integers and a string **s**. If there exists a mapping for the string **s** in the map **m**, it returns the corresponding integer number. Otherwise, it throws an error with a message containing the string **s** via the helper function **error**. For example,

```
val m: Map[String, Int] = Map("Ryu" -> 42, "PL" -> 37)
test(getKey(m, "Ryu"), 42)
test(getKey(m, "PL"), 37)
testExc(getKey(m, "CS320"), "CS320")
```

3.3 User-defined Structures (20 pts)

We provide the **Tree** type to represent binary trees. It is either **Branch** for a non-leaf node, or a **Leaf** for a leaf node. A **Branch** consists of three members; **left** and **right** denote the left and right sub-trees, and **value** denotes its value. A **Leaf** has unique member **value** to represent its value. (Note: DO NOT re-define **Tree** in **package.scala** because it is already defined in **Homework01.scala**)

```
trait Tree
case class Branch(left: Tree, value: Int, right: Tree) extends Tree
case class Leaf(value: Int) extends Tree
```

1. Define the function **countLeaves**, which consumes a tree **t** and returns the number of its leaf nodes. For example,

```
val tree: Tree = Branch(Leaf(1), 2, Branch(Leaf(3), 4, Leaf(5)))
test(countLeaves(tree), 3)
```

2. Define the function **flatten**, which consumes a tree **t** and returns a list containing the values of nodes inside the tree **t** with in-order tree traversals.

```
val tree: Tree = Branch(Leaf(1), 2, Branch(Leaf(3), 4, Leaf(5)))
test(flatten(tree), List(1, 2, 3, 4, 5))
```