CS320 Programming Languages
Scala Tutorial

1 Getting Started

1. Install JDK 8. You can find installation files on `https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html`.
2. Check the JDK version by executing `java -version` in your terminal.
3. Install Scala 2.13.3. You can find installation files on `https://www.scala-lang.org/download/`. (See the bottom of the webpage.)
4. Check the Scala version by executing `scala -version` in your terminal.
5. Install SBT 1.3.13. You can find installation files on `https://www.scala-sbt.org/download.html`.
6. Download the SBT project by executing `sbt new kaist-plrg-cs320/scala-tutorial.g8` in your terminal. The command will create the `scala-tutorial` directory (though you may change the name during the creation).

```
$ sbt new kaist-plrg-cs320/scala-tutorial.g8
[info] welcome to sbt 1.3.13
[info] loading global plugins from
[info] set current project to (in build)
[info] set current project to (in build)
name [Scala Tutorial]:

Template applied in ./scala-tutorial
```

2 Directory Structure

```
scala-tutorial
├── build.sbt
├── core/src
│   ├── main/scala/cs320
│   │   ├── Assignment.scala
│   │   ├── Implementation.scala
│   │   └── Template.scala
│   └── test/scala/cs320
│       ├── Spec.scala
│       └── SpecBase.scala
├── macros
└── project
```

- `core/src/main/scala/cs320/Assignment.scala`
  This file contains the following utility functions:

- `error`

  ```
  def error(): Nothing
  ```

  The function throws an exception with the empty string as an error message. Use this function to throw an exception without any information.

- `error`

  ```
  def error(msg: String): Nothing
  ```

  The function throws an exception with a given string as an error message. Use this function to throw an exception with some additional information.

- `cast`

  ```
  def cast[T](v: Any, msg: String): T
  ```

  The function check whether `v` is a value of a type `T`. If so, it returns `v`, and the return value can be used as a value of `T`. Otherwise, it throws an exception with a given string as an error message.

DO NOT use the `throw` keyword to throw an exception. You must use the provided `error` functions.

- `core/src/main/scala/cs320/Template.scala`
  This file contains the definitions of functions that you must implement to complete the exercise. In addition, it defines some types used in the exercise. **DO NOT** edit this file.

- `core/src/main/scala/cs320/Implementation.scala`
  You must fill this file to implement the required functions. To finish the exercise, it is enough to edit **only** this file.

- `core/src/test/scala/cs320/Spec.scala`
  This file contains some test cases. You can test your implementation with this file. Passing all the provided tests does not guarantee that your implementation is correct. We highly recommend you to add your own tests in this file.

In summary, fill the `core/src/main/scala/cs320/Implementation.scala` file to complete the exercise. You may add your own tests in the `core/src/test/scala/cs320/Spec.scala` file. In any cases, you must not edit files other than the above two files.

3 Instructions

Under the `scala-tutorial` directory, execute `sbt` to launch an SBT server. On the SBT console, you can test your implementation with the `test` command. Without changing `Implementation.scala`, you will see that every test fails.

```
$ sbt
[info] welcome to sbt 1.3.13
[info] loading global plugins from
[info] loading project definition from
[info] loading settings for project scala-tutorial from build.sbt ...
[info] set current project to scala-tutorial (in build file:
[info] sbt server started at local
sbt:scala-tutorial> test
[info] Compiling 1 Scala source to ...
[info] Compiling 3 Scala sources to ...
[info] Compiling 2 Scala sources to ...
[info] Spec:
```

```
[info] volumeOfCuboid(1, 3, 5)
[info] - should be 15 *** FAILED ***
[info]   scala.NotImplementedError: an implementation is missing
...
[info] Run completed in 1 second, 310 milliseconds.
[info] Total number of tests run: 20
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 0, failed 20, canceled 0, ignored 0, pending 0
[info] *** 20 TESTS FAILED ***
[error] Failed tests:
[error]         cs320.Spec
[error] (core / Test / test) sbt.TestsFailedException: Tests unsuccessful
[error] Total time: 10 s
```

After implementing all the functions correctly, you will see that every test succeeds.

```
sbt:scala-tutorial> test
[info] Compiling 1 Scala source to ...
[info] Spec:
[info] volumeOfCuboid(1, 3, 5)
[info] - should be 15
[info] volumeOfCuboid(2, 3, 4)
[info] - should be 24
[info] concat("x", "y")
[info] - should be "xy"
[info] concat("abc", "def")
[info] - should be "abcdef"
...
[info] countLeaves(t1)
[info] - should be 2
[info] countLeaves(t2)
[info] - should be 3
[info] flatten(t1)
[info] - should be List(1, 2, 3)
[info] flatten(t2)
[info] - should be List(1, 2, 3, 4, 5)
[info] Run completed in 232 milliseconds.
[info] Total number of tests run: 20
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 20, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 5 s
```

You may use ~ to re-execute a certain command automatically each time you update the code.

```
sbt:scala-tutorial> ~test
...
```

You must follow the next rules while implementing the functions:

- **DO NOT** use mutable variables, such as `var x = ...`.
- **DO NOT** use mutable collections, such as `scala.collection.mutable.Map`.
- You can use immutable collections. More precisely, you can use
  - every method of `scala.collection.immutable.List` except `addString`, `combinations`, `copyToArray`, `copyToBuffer`, `grouped`, `inits`, `iterator`, `permutations`, `reverseIterator`, `sliding`, `stepper`, `tails`, `to`, `toArray`, `toBuffer`, and `toIterator`.

  - every method of `scala.collection.immutable.Map` except `addString`, `copyToArray`, `copyToBuffer`, `grouped`, `inits`, `iterator`, `keyStepper`, `keysIterator`, `sliding`, `stepper`, `tails`, `to`, `toArray`, `toBuffer`, `toIterator`, `valueStepper`, and `valuesIterator`.
  - every method of `scala.collection.immutable.Set` except `addString`, `copyToArray`, `copyToBuffer`, `grouped`, `inits`, `iterator`, `sliding`, `stepper`, `subsets`, `tails`, `to`, `toArray`, `toBuffer`, and `toIterator`.
  - every method of `scala.Option` except `addString`, `copyToArray`, `copyToBuffer`, `grouped`, `inits`, `iterator`, `productElementNames`, `productIterator`, `sliding`, `stepper`, `tails`, `to`, `toArray`, `toBuffer`, and `toIterator`.

- **DO NOT** use `while` loops and `break` statements. However, you may use `for` loops.
- **DO NOT** use the `import` keyword.

4 Description of Functions

4.1 Primitives

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: $0 \leq \mathtt{a}, \mathtt{b}, \mathtt{c} \leq 1,000$)

```
test(volumeOfCuboid(1, 3, 5), 15)
test(volumeOfCuboid(2, 3, 4), 24)
```

2. Write the function `concat`, which consumes two strings `x` and `y`, and it returns their concatenation. For example,

```
test(concat("x", "y"), "xy")
test(concat("abc", "def"), "abcdef")
```

4.2 Function Values

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,

```
test(addN(5)(3), 8)
test(addN(5)(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,

```
test(twice(addN(3))(2), 8)
test(twice(addN(3))(7), 13)
```

3. Write the function `compose`, which consumes two `Int => Int` functions `f` and `g` and returns their composition $\mathtt{f} \circ \mathtt{g}$. For example,

```
test(compose(addN(3), addN(4))(5), 12)
test(compose(addN(3), addN(4))(11), 18)
```

4.3 Lists

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,

```
test(double(List(1, 2, 3)), List(2, 4, 6))
test(double(double(List(1, 2, 3, 4, 5))), List(4, 8, 12, 16, 20))
```

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)
```

4.4 Maps

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)
testExc(getKey(m, "CS320"), "CS320")
```

4.5 Algebraic Data Types

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 `Implementation.scala` because it is already defined in `Template.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 t1: Tree = Branch(Leaf(1), 2, Leaf(3))
val t2: Tree = Branch(Leaf(1), 2, Branch(Leaf(3), 4, Leaf(5)))
test(countLeaves(t1), 2)
test(countLeaves(t2), 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 t1: Tree = Branch(Leaf(1), 2, Leaf(3))
val t2: Tree = Branch(Leaf(1), 2, Branch(Leaf(3), 4, Leaf(5)))
test(flatten(t1), List(1, 2, 3))
test(flatten(t2), List(1, 2, 3, 4, 5))
```