YSC4231: Parallel, Concurrent and Distributed Programming

Programming Assignment 1

The assignments for the corresponding problems are located in the packages `homework.problemX` under the `src/main/scala` source root of the Scala Primer project.[1] The test templates are located in the test suite files in the same packages, but under the `test/main/scala` folder. Use IntelliJ navigation to locate the corresponding files and run the tests. Note that tests reveal a number of library functions that you might want to use in your implementations, for instance, for generating random numbers.

The score for this programming assignment depends on how many of the automated tests pass (initially almost all of them fail). Ideally, all of them should eventually turn green. Some of the tasks require you to modify the provided tests or add new ones.[2]

Problem 1. The templates for the subtasks of this problem are available in the object `Problem1` of the Scala primer project.

- Implement a function with the following signature and tests for it:

  ```
  def compose[A, B, C](g: B => C, f: A => B): A => C = ???
  ```

- Implement a `fuse` function with the following signature:

  ```
  def fuse[A, B](a: Option[A], b: Option[B]): Option[(A, B)] = ???
  ```

  The resulting `Option` instance should contain a tuple of values from the `Options` `a` and `b` if they are both non-empty (e.g., they are not `None`). If possible, use `for`-comprehensions to implement this function.[3]
- Implement the function `check` with the following signature:

  ```
  def check[T](xs: Seq[T], pred: T => Boolean): Boolean = ???
  ```

  It should return `true` if and only if the `pred` function returns `true` for all the values in `xs`. As a bonus you can implement it in a way that catches exceptions that can be possibly thrown by `pred`, via Scala's `try`/`catch`/`finally` statement.[4] In this case, please, add a test for this scenario.
- Modify the `Pair` class from the lecture notes so it can be used in a pattern matching statement.

Problem 2. Implement your versions of the following functions on Scala lists:

- `def myMap[T, S](l: List[T], f: T => S): List[S] = ???`
- `def myFilter[T](l: List[T], f: T => Boolean): List[T] = ???`
- `def myFlatten[T](l: List[List[T]]): List[T] = ???`
- `def myFoldLeft[A, B](l: List[A], z: B, f: B => A => B): B = ???`
- `def myFoldRight[A, B](l: List[A], z: B, f: A => B => B): B = ???`

The test suite `Problem2Tests` provides some basic tests, relating those functions to the ones from the standard Scala library. Feel free to extend it with your own tests.

Problem 3. Implement the following two functions:

- `def listToArray[T : ClassTag](l : List[T]): Array[T] = ???`
- `def arrayToList[T](a: Array[T]): List[T] = ???`

Please, do not attempt to rely on similar conversions from the standar library (but it's okay to use other functions on lists and arrays). Add the corresponding checks to the test suite `Problem3Tests`, as per the title of the tests, and make sure that the tests pass.

[1] Link avaialbe at Canvas.

[2] Indeed, the modifications are expected to be "sensible" e.g., replacing `assert(false)` by some problem-specific property to check or adding more "preparatory" code.

[3] You can find online instructions on what does it mean to use Scala `for`-expressions for `Options`.

[4] https://www.scala-lang.org/old/node/255.html

Problem 4. Implement the following two classes of mutable (imperative) data structures:[5]

- `class SequentialQueue[T: ClassTag]`
- `class SequentialStack[T: ClassTag]`

Feel free to use Scala's lists and/or arrays in any combinations with mutable fields (`var`) to implement the underlying logic for a stack and a queue. You might want to check the lecture notes from YSC2229 for inspiration.[6] Provide meaningful implementations for the corresponding tests.

Problem 5. Change your implementations of `SequentialQueue` and `SequentialStack` to obtain the following classes, which are suitable to use in the presence of concurrent modifications:

- `class ConcurrentQueue[T: ClassTag]`
- `class ConcurrentStack[T: ClassTag]`

Use any mechanisms for synchronisation/mutual exclusion in Scala that you are aware of (e.g., from the lectures). It is fine to delegate some logic to your sequential implementations. The crux of this exercise is in designing the tests (in `Problem5Tests`) showing that certain *desirable* properties are preserved by the concurrent implementation. Follow the hints in the tests.[7]

[5] Never mind the `ClassTag` annotation: it is only required by the compiler if you decide to base your queue/stack on an array.

[6] https://ilyasergey.net/YSC2229/

[7] Once done with those tests and having made sure that they pass, try to use sequential collections instead of concurrent ones and see if the tests *sometimes* might fail.