Homework 9

Please email your solutions to Rongdi Huang (rh1424@nyu.edu). Solutions to programming exercises must be submitted electronically as plain text files. No exotic formats, please!

The deadline for Homework 9 is December 12.

For the following problems, make sure your code runs under Scala. The Scala language distribution can be downloaded from [http://www.scala-lang.org](http://www.scala-lang.org).

While implementing this exercise, please make sure that you don’t change the definitions given in the templates. In particular:

- Don’t change the package declaration or the object name.
- Don’t change the function names.
- Don’t change the given function signatures.

Respecting these guidelines allows us to be more efficient in correcting the submissions. Thank you for your collaboration.

**Problem 1  Calendar (20 Points)**

In this part we are interested in printing a calendar using Scala. More specifically, we want to print an overview of a given month that shows which date falls on which day of the week.

For example, in 2012, the First of August was a Wednesday. The month of August 2012 should be printed as follows:

<table>
<thead>
<tr>
<th>Su</th>
<th>Mo</th>
<th>Tu</th>
<th>We</th>
<th>Th</th>
<th>Fr</th>
<th>Sa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Before you start solving this exercise, make yourself familiar with the methods provided by the **List** class in the Scala standard API. All parts of this exercise have very short solutions if you use the appropriate functions provided by class **List**.

**Warm-Up (2 Points)**

Define a function `unlines` that turns a list of lists of characters into a list of characters inserting a `\n` character between each two lists. The following example illustrates the function `unlines`:

```
unlines(List(List('f','e','i','s','t','y'),List('f','a','w','n'))) should yield
```

1
Leap years, the First of January and all that (2 Points)

To be able to print a monthly overview, we first have to determine on which weekday falls
the first day of the given month. We provide you with the following function definitions to
simplify this task:

```scala
/** The weekday of January 1st in year y, represented
   * as an Int. 0 is Sunday, 1 is Monday etc. */
def firstOfJan(y: Int): Int = {
  val x = y - 1
  (365*x + x/4 - x/100 + x/400 + 1) % 7
}

def isLeapYear(y: Int) =
  if (y % 100 == 0) (y % 400 == 0) else (y % 4 == 0)

def mlengths(y: Int): List[Int] = {
  val feb = if (isLeapYear(y)) 29 else 28
  List(31, feb, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
}

With the help of these functions, define a function firstDay that calculates the weekday of
the first day of a given month:

def firstDay(month: Int, year: Int): Int = ...

How to picture that? (16 Points)

Picturing data with a non-trivial layout such as a calendar can be tricky. Therefore, we
want to use a compositional approach where larger, more complex pictures are composed of
smaller, simpler pictures.

In our design, pictures are represented as instances of the Picture case class:

```scala
case class Picture(height: Int, width: Int, pxx: List[List[Char]]) {
  def showIt: String = unlines(pxx).mkString(
```

As we can see, a picture has a height and width, and contents pxx which is character data
represented as a list of rows, where each row is a list of characters. The showIt method
turns the picture into a list of characters using the unlines function defined in the first part.
The following function pixel creates a simple picture of height and width 1 that contains a
given character:

def pixel(c: Char) = Picture(1, 1, List(List(c)))

From pictures as simple as that, we want to compose larger ones using composition operators.
(a) Define a method `above` for class `Picture` that returns a new picture where the argument picture is put below this:

```scala
case class Picture(...) {
  def above(q: Picture): Picture = ...
}
```

For instance, the following code

```scala```
println((pixel('a') above pixel('b')).showIt)
```

should print

```
a
b```

Give an error message (using the predefined function `sys.error`) when the pictures do not have the same width.

(b) Define a method `beside` for class `Picture` that returns a new picture where the argument picture is put on the right side of this:

```scala
case class Picture(...) {
  def beside(q: Picture): Picture = ...
}
```

Give an error message (using the predefined function `sys.error`) when the pictures do not have the same height.

(c) Define functions `stack` and `spread` that arrange a list of pictures above and beside each other, respectively, producing a single resulting picture. For `stack`, the picture at the head of the argument list should be the topmost picture in the result. Similarly for `spread`, the head of the list should be the leftmost picture in the result.

```scala```
def stack(pics: List[Picture]): Picture = ...
def spread(pics: List[Picture]): Picture = ...
```

(d) Define a function `tile` that arranges a list of rows of pictures in a rectangular way using the `stack` and `spread` functions:

```scala```
def tile(pxx: List[List[Picture]]): Picture = ...
```

(e) Define a function that takes a width `w` and a list of characters, and produces a picture of height 1 and width `w` where the given characters are justified on the right border:

```scala```
def rightJustify(w: Int)(chars: List[Char]): Picture = ...
```

Give an error message if `chars.length > w`. 

3
(f) Define a function `group` that splits a list into sublists. The function takes an integer as argument that indicates the split indices (e.g. split every 7 elements). We intend to use this function to split a list representing a whole month into a list of weeks. Note that this function is parameterized which means that it can be used with lists of any element type.

```scala
def group[T](n: Int, xs: List[T]): List[List[T]] = ...
```

(g) Define a function `dayPics` that takes the number of the first day and the number of days of a month and produces a list of 42 pictures. In this list, the first d pictures are empty (i.e., the character data is a list of spaces) if the number of the first day is d (d==0: Sunday, d==1: Monday, etc.). The trailing pictures that correspond to days of the next month are empty, too. Using this function, a picture of a calendar can be produced by grouping and tiling the result of `dayPics`.

```scala
def dayPics(d: Int, s: Int): List[Picture] = ...
```

*Hint:* A Scala string can be converted to a list of characters by calling its `toList` method. This might come in handy when converting days to lists of characters.

(h) Using the functions defined in the previous steps, define a function `calendar` that produces a picture of a calendar that corresponds to the given year and month.

```scala
def calendar(year: Int, month: Int): Picture = ...
```

Problem 2  User-Defined Control Constructs (10 Points)

Scala (deliberately) does not provide `break` and `continue` statements for loops. Extend the `whileLoop` control constructs from the lecture slides with `break` and `continue` constructs that implement the appropriate behavior of `break` and `continue` statements in languages such as C. The following two examples demonstrate how the constructs should work:

```scala
var x = 0
whileLoop (x < 5) {
  x += 1
  if (x == 3) continue
  println(x)
}

1
2
4
5
```

```scala
var x = 0
whileLoop (x < 5) {
  x += 1
  if (x == 3) break
  println(x)
}

1
2
```

*Hint:* One way of implementing `break` and `continue` is using exceptions.