The SDP Primer

Exercise sheet 02

Session: 2016-17

1 Purposes of this assignment

- To get you started programming in Scala.
- To convey the idea of "Write a little, test a little."

2 General idea of the assignment

Hammurabi is a very old computer game, in which you are the ruler of ancient Samaria, trying to increase the wealth of your country. Your job is to bring this program into the 21st century by writing it in Scala.

3 Step by step instructions

Since some of you have never programmed before, this assignment is in the form of a step-by-step tutorial. Later assignments will leave more up to you.

1. Using a text editor, write a `println` statement to print out the following introductory message:

```
Congratulations, you are the newest ruler of ancient Samaria, elected
for a ten year term of office. Your duties are to dispense food, direct
farming, and buy and sell land as needed to support your people. Watch
out for rat infestations and the plague! Grain is the general currency,
measured in bushels. The following will help you in your decisions:

  * Each person needs at least 20 bushels of grain per year to survive.

  * Each person can farm at most 10 acres of land.

  * It takes 2 bushels of grain to farm an acre of land.

  * The market price for land fluctuates yearly.

Rule wisely and you will be showered with appreciation at the end of
your term. Rule poorly and you will be kicked out of office!
```

Hint #1: You can put this all into a single `println` statement by using a so-called "triply-quoted string"; that is, a string that begins and ends with three double quote marks. Such a string can hold more than one line.

Hint #2: You can save some typing by copying code from the assignment and pasting it into your Scala file.

How to run this:

(a) Save the above on a file named `Hammurabi.scala`.

(b) Open a command-line ("Terminal" or "DOS") window.

(c) Use the `cd path_to_directory` ("change directory") command to move to the directory containing the `Hammurabi.scala` file.

(d) Start the REPL by entering the command: `scala`

(e) In the REPL, enter the command `:load Hammurabi.scala`
This should display the above introductory message.

Stay in the REPL as you write the rest of the program.

2. Put the `println` statement from above into a function called `printIntroductoryMessage`. Test your function by calling it from the REPL.

   Save and load the modified program. Since the `println` expression is now inside a function, it is not automatically executed when you load the program. To run it, enter `printIntroductoryMessage`

3. Write a function called `hammurabi`. Inside this function put a call to `printIntroductoryMessage`.

   Once we have more than one function, we have to arrange them in a certain order. This is a nuisance. To avoid this problem, "wrap" all your code into an object, like this:

```
object Hammurabi {
  All your code goes in here
}
```

   Test your `hammurabi` function. Since your function is now inside the Hammurabi object, call it by entering `Hammurabi.hammurabi`. That is, the name of the object, a dot, and the name of the function you want to call. Other functions can be tested similarly.

4. Declare the following variables, which you will need in the rest of the program. Put them as the first thing inside your `hammurabi` function, before the call to `printIntroductoryMessage`.

```
var starved = 0            // how many people starved
var immigrants = 5         // how many people came to the city
var population = 100
var harvest = 3000          // total bushels harvested
var bushelsPerAcre = 3      // amount harvested for each acre planted
var rats_ate = 200          // bushels destroyed by rats
var bushelsInStorage = 2800
```

```
var acresOwned = 1000
var pricePerAcre = 19       // each acre costs this many bushels
var plagueDeaths = 0
```

5. Next inside `hammurabi`, write a loop to print the following report ten times, with the variable `year` set to 1, 2, 3, ..., 10. Each of the numbers in the report (shown in red) should be the value of the corresponding variable (remember that you can "add" strings and numbers).

```
O great Hammurabi!
You are in year 1 of your ten year rule.
In the previous year 0 people starved to death.
In the previous year 5 people entered the kingdom.
The population is now 100.
We harvested 3000 bushels at 3 bushels per acre.
Rats destroyed 200 bushels, leaving 2800 bushels in storage.
The city owns 1000 acres of land.
Land is currently worth 19 bushels per acre.
There were 0 deaths from the plague.
```

   Remember that expressions inside parentheses can extend across more than one line. It is good style to keep your lines short (less than about 80 characters) in a program. A good place to break some of the above `println` statements is just after a `+`.

   The above summary represents the initial state, at the *beginning* of the first year– that is, when you first take office, and before you do any of the computations below). So, for example, the previous year (under a different ruler) must have started with 95 people; none starved, and 5 entered the kingdom, so as you enter office you rule 100 people.

   Since you are not yet changing the values of any of the variables, when you test the `hammurabi` function you should get the *same* report (except for the `year` number) printed out ten times.

That's it for most of the printing.

6. You will be asking the user to enter integers, which you *could* do with code like this:

```
var answer = readLine(prompt).toInt
```

   but if the user enters something that isn't an integer, the program will crash. That's frustrating. Instead, copy *and test* the following function, which I have written for you:

```
def readInt(message: String): Int = {
  try {
    readLine(message).toInt
  } catch {
    case _ : Throwable =>
      println("That's not an integer. Please enter an integer.")
      readInt(message)
```

```
    }
  }
```

7. Next, inside the "year loop," you will use functions to ask the Great Hammurabi (the user) to make some decisions. Check each decision to see whether it is possible (give enough parameters to the function to be able to decide this). If it is not possible, ask the Great Hammurabi very, very politely (as befits his high status) for a different answer, and repeat if necessary until a legal answer is obtained.

   At the beginning of each year, ask the player ("Hammurabi") for:

   - **How many acres of land to buy**
   - **How many acres of land to sell**
     - Do not ask this question if the player is buying land.
   - **How much grain to feed to the people**
   - **How many acres to plant with seed**

   Questions should be asked in this order, and the player does not get a chance to go back and change an earlier answer. A player who is buying land is not allowed to sell land during the same turn; so if the user asks to buy land, do not call the function that asks how much land to sell.

   As an example, the first function has been written for you.

```
def askHowMuchLandToBuy(bushels: Int, price: Int) = {
  var acresToBuy = readInt("How many acres will you buy? ")
  while (acresToBuy < 0 || acresToBuy * price > bushels) {
    println("O Great Hammurabi, we have but " + bushels + " bushels of grain!")
    acresToBuy = readInt("How many acres will you buy? ")
  }
  acresToBuy
}
```

   And here is a sample call to the function (which you would put inside the `hammurabi` function).

```
var acresToBuy = askHowMuchLandToBuy(bushelsInStorage, pricePerAcre)
acresOwned = acresOwned + acresToBuy
```

   Each function should be given just enough information to test whether the user's answer is legal, and to return that legal answer. That answer should then be used (within the `hammurabi` function) to adjust the values of the variables declared in 4 above. Don't forget to test each function as you write it!

8. After you collect the answers to the above questions, you need to determine the consequences, in the order specified below. For each of the following, call an appropriately-named function, and use the number returned by the function to update the variables in the `hammurabi` function. Do the update immediately, before going on to the next part (for example, if people die of the plague, there are fewer people when you check for starvation). Test each function as you go.

Many of the functions use "random" numbers; see below for information on how to get these numbers.

a. **If there is a plague**

- Each year, there is a 15% chance of a horrible plague. When this happens, half your people die. Change the `population` variable immediately, before going on to the next part.
- There are a couple of ways you could write this function. You could just return `true` 15% of the time, and `false` the other 85% of the time. Or, you could pass in the number of people you have, and return the number who died (or the number you have left).
- Similar comments apply to the rest of the functions in this list.

b. **How many people starved**

- Each person needs 20 bushels of grain to survive. If you feed them more than this, the grain has been wasted.
- If more than 45% of the people starve, you will be immediately thrown out of office (in this case, you should print a suitably nasty message), and the game ends.

c. **How many people came to the city**

- Nobody will come to the city if people are starving. If everyone is well fed, compute how many people come to the city as:

  ```
  (20 * number of acres + amount of grain in storage) / (100 * population) + 1
  ```

d. **How good the harvest is**

- Choose a random number between 1 and 8, inclusive. Each acre that was planted with seed will yield this many bushels of grain. (Example: if you planted 50 acres, and the random number is 3, you harvest 150 bushels of grain).

e. **If you have a problem with rats**

- There is a 40% chance that you will have a rat infestation. When this happens, rats will eat somewhere between 1/10 and 3/10 of your grain.

f. **How much land will cost next year**

- The price of land is random, and ranges from 17 to 23 bushels per acre. The player will need this information in order to buy or sell land.

9. At the end of the game (inside the `hammurabi` function but outside the "year loop"), use a function to print out a final summary, and to tell the player how good a job he/she did. I'll leave the details up to you, but the usual evaluation is based on how many people starved, and how many acres you end up with.

10. To start your program as soon as it is loaded, add the line `Hammurabi.hammurabi` at the very end of the file.

4 Additional information

4.1 Arithmetic

All the required arithmetic in this program should be **integer**. You do not need doubles; rounding errors of one or two percentage points are acceptable.

4.2 Random numbers

To get random numbers, you need to get a random number generator from Scala's vast library of pre-written code. Put this line at the very beginning of your program:

```
import scala.util.Random
```

To get a new random number in the range `0` to `n-1`, call `Random.nextInt(n)`. For example, to simulate the roll of a die, you could use `Random.nextInt(6)+1`. (The `+1` is necessary because the function call will return a number between 0 and 5, inclusive.) To do something that happens `p` percent of the time, use

```
if (Random.nextInt(100) < p) {
  // do something
}
```

4.3 Summary of program structure

```
import scala.util.Random

object Hammurabi {
  def hammurabi = {...}
  Other function definitions
}
Hammurabi.hammurabi
```