CSCI 1320 (Principles of Programming I), Fall 2011

Homework 3

Credit: 30 points.

1 Reading

Be sure you have read chapter 7.

2 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course number and the assignment (e.g., "csci 1320 homework 4"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (15 points) This problem compares various ways of operating on arrays and lists in Scala. Your mission is to write two Scala programs, one using arrays and one using lists, each with four different functions that count how many times a particular element appears in an array/list. To make it somewhat easier to get non-trivial arrays/lists to search, the programs will use `util.Random.nextInt` to generate integer data.

   So, each program should start by prompting the user for the number of elements in the array or list and the maximum value. It should then generate the array or list, and then prompt the user for a number to search for and call each of the different functions to count how many times the number to search for appears in the array or list. The four functions should all accomplish the same goal, but using different methods:

   - Using only recursion (no collection methods).
   - Using the `count` collection method.
   - Using the `filter` collection method.
   - Using the `map` collection method.

   Here is a sample execution (with text **in boldface** what you type and text `in typewriter font` what the program prints):

   ```
   how many numbers?
   20
   maximum value?
   10
   the numbers:
   0
   9
   ```

```
4
2
8
2
3
10
2
6
0
8
10
6
8
10
10
2
5
9
number to search for?
10
count using recursion = 4
count using count = 4
count using filter = 4
count using map = 4
```

Hints:

- As mentioned in the textbook, you can fill an array with randomly-generated integers like this (to generate an array of 10 integers with largest value 99):

  `Array.fill(10)(util.Random.nextInt(100))`

  The same thing works with lists (replace `Array` with `List`).

- The simplest method I have found for printing arrays and lists with one value per line uses `foreach`:

  `array_or_list.foreach(println(_))`

- The two programs should be very similar, perhaps almost identical, so you will probably find it easier to get one of them working (start with the one you think you will find easier) and then copy it and make the changes needed for the other.

- Try to think of a way on your own to use the various collection methods (`count`, `filter`, and `map`) to accomplish the desired goal, but don't hesitate to ask for additional hints if you need to.

2. (15 points) For this problem your mission is to write a Scala program to evaluate polynomials — i.e., expressions of the form

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

The program should prompt for the coefficients a_n through a_0 and the value of the variable x and then print the value of the above expression. It's probably simplest to first ask how many coefficients there will be, then ask for that many values (so you can use `fill` to get the values), and finally ask for the value of the variable x. Sample execution (as usual, text **in boldface** is what you type and text `in typewriter font` is what the program prints):

```
how many coefficients?
4
enter 4 coefficients, one per line
1
3
2
4
enter value for variable
100
value is 1030204.0
```

(Probably the program should allow entering doubles rather than integers; the example uses integers to make the answer easier to check.)

Hints:

- You can use either recursion or collection methods to compute the value. Which you choose is up to you, or you can even submit multiple solutions for extra credit.
- One way to approach a recursive solution is to use Horner's rule, which rewrites the polynomial shown earlier thus:

  $$(a_n x + a_{n-1})x + \cdots a_2)x + a_1)x + a_0$$

  More details about this hint on request, but think about it a bit first. Or you may have a different idea that you find more straightforward!