Examination Advanced R Programming

Linköpings Universitet, IDA, Statistik

Course code and name:	732A94 Advanced R Programming
Date:	2024/10/29, 8–12
Teacher:	Bayu Beta Brahmantio
Allowed aids:	The extra material is included in the zip file **exam_help_material_732A94.zip**
Grades:	A= $[18-20]$ points B= $[16-18)$ points C= $[14-16)$ points D= $[12-14)$ points E= $[10-12)$ points F= $[0-10)$ points
Instructions:	Write your answers in R scripts named according to the pattern **[your exam account]_*.R** The R code should be complete and readable code, possible to run by calling `source()` directly on your `*.R` files. You may have a separate file for each question, or answer everything in one file. Comment directly in the code whenever something needs to be explained or discussed. Follow the instructions carefully. There are **THREE** problems (with sub–questions) to solve.

Problem 1 (2p)

Discuss the pros and cons of external cloud–based solutions for data storage and computations.

Problem 2 (12p)

READ THE WHOLE QUESTION BEFORE STARTING TO IMPLEMENT! Remember that your functions should **ALWAYS** check for correctness of user input! For each subquestion please provide **EXAMPLE CALLS!**

You are a poultry seller that has in stock various birds each with its own price, and you want to write an object oriented (S3, S4, or RC) program that will help you keep track of your stock and sells.

a) (3p) In this task you should use object oriented programming in S3, S4 or RC to write code that keeps track of your stock and sales. You should record how many birds of each type you have, how much they cost, and how much you have earned (0 at the start). Depending on your chosen OO

system you can do it through a constructor or by implementing a function `create_poultry_stock()`. The constructing function should not take any arguments. The object should contain for each bird its name, amount, and price (either per individual; or a certain number of individuals, if the given type is not sold individually).

```
## example call to create a stocks object
my_poultry_stock <- create_poultry_stock() # S3
my_poultry_stock <- poultry_stock$new() # RC
```

b) (4p) Now implement a function called `add_poultry()` that allows one to add new birds to the stock. The function should have four parameters describing the acquired birds: name, amount, price, and number of individuals that cost this price.

```
## S3 and RC example calls
my_poultry_stock <-add_poultry(my_poultry_stock,"chicken",30,1,3)
my_poultry_stock <-add_poultry(my_poultry_stock,"rooster",5,5,1)
my_poultry_stock <-add_poultry(my_poultry_stock,"hen",10,3,1)
## if using RC you may also call in this way
my_poultry_stock$add_poultry("chicken",30,1,3)
my_poultry_stock$add_poultry("rooster",5,5,1)
my_poultry_stock$add_poultry("hen",10,3,1)
```

c) (4p) Now implement a function called `sell_poultry()` that is called when someone wants to buy from you. However, the buyers do not give a list what they want to buy but only give the amount of birds they want and how much they will pay. You should check in your stock whether this is possible: if yes then print a list of the number of birds of each type you sell, appropriately decrease your stock and increase your earnings; if not (in particular the cost of the stock has to exactly agree with the amount the buyer wants to pay) write that such a transaction is impossible. Implement a greedy algorithm for this (you are **NOT** expected to implement an optimal nor correct algorithm for the problem, i.e., you will **NOT** be penalized if you return that no offer is possible when a solution exists; however you **will be** penalized if you return a solution that does not meet the requirements on the number of individual birds or selling price), e.g., propose as many as you can birds of the first type, then of the second, third, e.t.c., until you reach the correct number of birds and price; or you cannot add any more birds.

```
## S3 and RC example call
my_poultry_stock<-sell_poultry(my_poultry_stock,4,8)
## if using RC you may also call in this way
my_poultry_stock$sell_poultry(4,8)
```

In the above example 3 chickens and 1 rooster is a solution.

d) (1p) Implement a function that displays the state of your poultry stocks. You are free to choose yourself how to report the state! This function has to also work directly with `print()`.

```
# calls to show state of the poultry stock
my_poultry_stock; print(my_poultry_stock)
```

Problem 3 (6p)

a) (3p) Please implement a function that finds the smallest number, x, that satisfies a specific divisibility property. Namely you will be given a certain number of divisors, $Q = \{q_i\}$ and remainders, $R = \{r_i\}$, when x is divided by them, i.e.,. $x/q_i = k_i$ remainder r_i, where k_i is some (unknown to us) integer. For example for $Q = \{3, 5, 7\}$ and $R = \{2, 3, 2\}$ we will have $x = 23$, as $23/3 = 7$ remainder 2; $23/5 = 4$ remainder 3; $23/7 = 3$ remainder 2. The input of your function is to be the vectors Q and R (notice that they have to be of the same length), and a maximum value, N, after which you should stop searching for x. Implement, using one of the loops, a brute force search algorithm, that checks all the integers from 1 to N until it finds the one that satisfies the required conditions.
b) (2p) In the course we discussed that the `apply` family of functions can be a more effective alternative to loops. Discuss whether it will be in this case here.
c) (1p) Implement a unit test, using the example with $x = 23$ and another one that you make up yourself.