Examination Advanced R Programming

Linköpings Universitet, IDA, Statistik

Course code and name:	732A94 Advanced R Programming
Date:	2021/12/01, 8–12
Teacher:	Krzysztof Bartoszek phone 013–281 885
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 an R script file named **[your exam account].R** The R code should be complete and readable code, possible to run by copying directly into a script. 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 (5p)

a) (3p) Discuss some ways in which code implemented in `R` can be optimized and improved with respect to speed.
b) (2p) Explain why code profiling is critical in the code optimization process.

Problem 2 (10p)

READ THE WHOLE QUESTION BEFORE STARTING TO IMPLEMENT! Remember that your functions should **ALWAYS** check for correctness of user input!

a) (2p) In this task you should use object oriented programming in S3 or RC to write code that keeps track of cars brought to a car mechanic. It should contain information on the car, and what spare parts are required for it.

Your goal is to first initialize the workshop object, which should be of class `"workshop"`. Depending on your chosen OO system you can do it through a constructor (RC) or by implementing a function `create_workshop()` (S3). The constructing function should take one argument, `max_number_cars`, the maximum number of cars that can be handled simultaneously.

The workshop object should contain for each car information describing it, in particular a unique id of the car, a text name (you may have the same one e.g. `"a"` for every car), identifiers (you choose yourself what these should be) of spare parts that will be required for the car, and the status of the car (waiting to be fixed or fixed). The workshop should also contain information on spare parts that it has stored, the identifier of the part and how much of it. If needed, you may assume that the number of distinct spare parts is bounded by some constant, and if needed set this constant yourself but it has to be greater than 20. Provide some example calls to your code.

```
## example call to create a garden object
my_workshop <- create_workshop(max_number_cars=100) # S3
my_workshop <- workshop$new(max_number_cars=100) # RC
```

b) (3p) Now implement a function called `car_delivered_to_mechanic()` that adds a car to the workshop for fixing. The function should have two parameters: the text name of the car, and what parts needs replacement. The id is to be automatically generated. A car that requires parts to be replaced should have a status indicating that it requires fixing. You should check if some parts can be immediately replaced, do not forget then to update the information on stored spare parts. Do not forget that the workshop has a maximum capacity! Provide some example calls to your code.

```
## S3 and RC example calls
my_workshop <-car_delivered_to_mechanic(my_workshop,"SAAB",c(1,4,5))
my_workshop <-car_delivered_to_mechanic(my_workshop,"SAAB",c(1,4,5))

## if using RC you may also call in this way
my_workshop$car_delivered_to_mechanic("SAAB",c(1,4,5))
```

c) (3p) Now implement a function called `spare_parts_delivery()` that is responsible for a delivery of spare parts to the workshop. You choose its interface, but different spare parts should be delivered in one call and also information on how many of each part was delivered needs to be provided alongside. The function then should do replacement of parts in the cars where this is possible. A spare part can be used **ONLY ONCE**. A car that has all required parts replaced should change status to fixed. Remember that a car might need to have parts replaced in multiple calls to `spare_parts_delivery()` before it is fixed. Unused delivered spare parts should go to the workshop's storage. Provide some example calls to your code.

```
## S3 and RC call
my_workshop<-spare_parts_delivery(my_workshop,parts=c(1,2,5),amount=c(10,5,4))
my_workshop<-spare_parts_delivery(my_workshop,parts=c(1,2,5),amount=c(10,5,4))


## if using RC you may also call in this way
my_workshop$spare_parts_delivery(parts=c(1,2,5),amount=c(10,5,4))
```

d) (2p) Implement a function that displays the state of the workshop. 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 garden
my_workshop; print(my_workshop)
```

Problem 3 (5p)

a) (2p) The Babylonian method for finding a square root of a (positive) number is given as

```
1: procedure BabylonianSquareRoot(z,N)
2:     x:=1
3:     for i = 1 ... N do
4:         x:=(x/2)+z/(2*x)
5:     end for
6:     return x
7: end procedure
```

Implement a function that takes as its input a positive number and returns an approximation of the square root according to the pseudocode above. You need to choose the number of iterations, `N`, yourself. Provide example calls and results of runs of your code. Do not forget that your function should check for correctness of input and react appropriately.

b) (2p) What is the computational complexity in terms of `N` of the following code

```
1: procedure FunFunction(N)
2:     x:=rep(0,N)
3:     for i = 1 ... N do
4:         x[i]:=BabylonianSquareRoot(2,N)
5:     end for
6:     return x
7: end procedure
```

How can you significantly improve the above code with respect to its complexity?

c) (1p) Implement a unit test that compares your implementation with direct calculation of the square root using `R`'s `sqrt()` function.