Examination Advanced R Programming

Linköpings Universitet, IDA, Statistik

Course code and name:	732A94 Advanced R Programming
Date:	2025/03/06, 8–12
Teacher:	Krzysztof Bartoszek
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 (4p)

Provide a simple example of a recursive function in `R`. Provide an non–recursive implementation of it. Which implementation the recursive or non–recursive one is more effective computationally? What optimization technique can be used to improve the recursive implementation?

Problem 2 (10p)

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 organizing a birthday party for your child and need to keep track of the invited guests. You want to write an object oriented (S3, S4, or RC) program that will do this for you.

a) (2p) In this task you should use object oriented programming in S3, S4 or RC to write code

that keeps track of your invitations. For each invited child you should keep track whether the child accepted or declined the invitation and whether there are any dietary restrictions. You should also store the child's name and surname. Depending on your chosen OO system you can do it through a constructor or by implementing a function `create_guest_list()`. The constructing function should take one arguments—the maximum number of children that can be invited.

```
## example call to create a guest list
my_guests <- create_guest_list(max_num=20) # S3
my_guests <- guests$new(max_num=20) # RC
```

b) (2p) Now implement a function called `invite_child()` that allows you to add an invited child. The function should take one argument—the child's name and surnames. The other properties are at this stage unknown. Do not forget about the limit on the number of children.

```
## S3 and RC example calls
my_guests<-invite_child(my_guests,"Jane Doe")
my_guests<-invite_child(my_guests,"John Doe")
my_guests<-invite_child(my_guests,"John Smith")
## if using RC you may also call in this way
my_guests$invite_child("Jane Doe")
my_guests$invite_child("John Doe")
my_guests$invite_child("John Smith")
```

c) (2p) Now implement a function called `respond()` that is called when the parents of the invited child respond to the invitation. It should take as its input the child, whether the invitation was accepted or not, and whether there are any dietary restrictions. Consider how you will handle children that have the same name and surname. The below example calls need not be the correct ones. Consider how you will handle meaningless input combinations.

```
## S3 and RC example call
my_guests<-respond(my_guests,"Jane Doe","no","")
my_guests<-respond(my_guests,"John Doe","yes","no crustacean")
my_guests<-respond(my_guests,"John Smith","yes","")
## if using RC you may also call in this way
my_guests$respond("Jane Doe","no","")
my_guests$respond("John Doe","yes","no crustacean")
my_guests$respond("John Smith","yes","")
```

d) (3p) Now implement a function called `update_response()` that is called when the parents of the invited child change their mind, or remember something about the dietary restrictions. It should take the same input as the `respond()` function and you should update the state. Do not forget about the limit on the number of children. Consider how you will handle children that have the same name and surname. The below example calls need not be the correct ones. Consider how you will handle meaningless input combinations.

```
## S3 and RC example call
my_guests<-update_response(my_guests,"Jane Doe","yes","no strawberries")
```

```
my_guests<-update_response(my_guests,"John Doe","yes","no peanuts, no crustacean")
my_guests<-update_response(my_guests,"John Smith","no","")
## if using RC you may also call in this way
my_guests$respond("Jane Doe","yes","no strawberries")
my_guests$respond("John Doe","yes","no peanuts, no crustacean")
my_guests$respond("John Smith","no","")
```

e) (1p) Implement a function that displays the current state of your invitation list. 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 guest list
my_guests; print(my_guests)
```

Problem 3 (6p)

a) (2p) Hero of Alexandria in the first century proposed a recursive scheme

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{2}{x_n}\right),$$

with initial condition $x_1 = 2$ to calculate $\sqrt{2}$. Implement a recursive and non–recursive function for the above approximation scheme. The degree of approximation, i.e., n should be an argument of your function.

b) (2p) What is the computational complexity of your implementations? Do both have the same complexity? Which one do you think is more effective.

c) (2p) It is know that the error of the approximation decays as

$$|x_{n+1} - \sqrt{2}| \leq \frac{1}{2\sqrt{2}}|x_n - \sqrt{2}|^2.$$

With the help of the above formula and `R`'s `sqrt(2)` implement a unit test for your implementation.