# Examination Advanced R Programming

Linköpings Universitet, IDA, Statistik

| | |
|---|---|
| Course code and name: | 732A94 Advanced R Programming |
| Date: | 2017/10/20, 8–12 |
| Teacher: | Krzysztof Bartoszek |
| Allowed aids: | The extra material is included in the zip file **exam_material.zip** |
| Grades: | A= $[19-20]$ points<br>B= $[17-19)$ points<br>C= $[12-17)$ points<br>D= $[10-12)$ points<br>E= $[8-10)$ points<br>F= $[0-8)$ points **(FAIL)** |
| Instructions: | Write your code in an R script file named **[your exam account]_Main.R**<br>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.<br>There are **THREE** problems (with sub–questions) to solve. |

## Problem 1 (5p)

**a) (1p)** Provide a command to empty your workspace.
**b) (1p)** Now make sure you have a clean workspace and create a simple function that takes one number as its parameter and returns its square (you do not have to write code checking for correctness of the input.
**c) (3p)** Run the function `package.skeleton()` and inspect the created directory structure. In your solution file write what directories and files were created and next to the name of each file and directory write **IN YOUR OWN WORDS** a few (not more than three) sentences what is its role in an R package and what information it should contain.

# Problem 2 (8p)

**READ THE WHOLE QUESTION BEFORE STARTING TO IMPLEMENT**

**a) (2p)** In this task you should use object oriented programming in S3 or RC to implement a simulator for British blockade runners transporting ball bearings from Sweden during World War II. The first task is to implement a function called `build_mgb()` that returns a blockade running motor gun boat (MGB) object. The `build_mgb()` function should take two arguments: **name** and **speed**. The object should contain a data structure that stores the history of the runs (a run is described by two positive numbers: weight of the cargo and travel time). Furthermore, the object should have a **status** field. At creation the value of the **status** field should be `"in_service"`.

```
## S3 and RC call to build_mgb() function
MGB_504 <- build_mgb(name="Hopewell",speed=46)
```

**b) (4p)** Now implement a function called `simulate_home_run()` to simulate the transport of the precious ball bearings cargo on the route from the Swedish port of Lysekil to the Humber estuary in the United Kingdom. This function can be added to the MGB object (if RC used) or it may be an ordinary function. The function should take one argument (none if RC used)—the mgb. It should randomly draw the weight of the cargo (normal distribution with mean 40 and standard deviation 2, see function `rnorm()`) and the travel time (exponential distribution with

$$\text{rate} = 1/\left((\text{cargo weight}) \cdot (900/\text{speed})/250\right),$$

see function `rexp()`, 900km is the approximate distance from Lysekil to the port in Hull, United Kingdom). The function should return (update if RC) the mgb object with information on the trip stored. There is however, one catch still. The Luftwaffe and Kriegsmarine are out there hunting for our blockade runner. So if the travel time is longer than 10 the MGB did not manage to cross the North Sea under the cover of darkness, is spotted and sunk. In such a case the `status` field should be changed to `"sunk"`. The function `simulate_home_run()` should check the `status` field before doing anything. If it equals `"in_service"`, then it should do the above described simulation. However, if it equals `"sunk"` it should print "[Boat name] is reported lost" (where Boat name is the boat's name) and not modify the object (RC) or return an unmodified object (S3).

```
## S3 and RC call
MGB_504 <-simulate_home_run(MGB_504)


## if using RC you may also call in this way
## MGB_504$simulate_home_run()
```

**c) (2p)** Run the simulation 100 times. Inspect and write if the boat was sunk and if so at which run. Implement a plot function for the MGB object. The plot should relate the travel times to the cargo carried. You are free to choose how to visualize this.

```
## Plotting call
plot(MGB_504)
```

**TIP:** If you see that your simulation does not return any or very few trips rerun it. Perhaps your boat was sunk on its first run. However, if this happens all the time inspect your code carefully.

# Problem 3 (7p)

**a) (3p)** Create a function called `find_max_value()` that takes a vector `x` and returns a list with two elements: the maximum value in the vector (name of element `max_value`) and its position (name of element `max_value_position`). If there are multiple values equalling the maximum value `max_value_position` should point at the last occurrence. When implementing you may **NOT** use the functions `min()`, `max()`, `sort()`, nor similar ones. You must manually go through the vector to find the maximum.

```
> x<-c(1,2,3,56,4,5)
> find_max_value(x)
$max_value
[1] 56

$max_value_position
[1] 4

> x<-c(1,2,3,56,4,56,5)
> find_max_value(x)
$max_value
[1] 56

$max_value_position
[1] 6
```

**b) (1p)** What is the complexity of your solution in terms of the number of elements of `x`?
**c) (1p)** You are now allowed to use the function `min()`. Recall that

$$\max\{x_1, \ldots, x_n\} = (-1) \cdot \min\{-x_1, \ldots, -x_n\}$$

and write a new function `find_max_value_2()` that uses `min()`. What is the computational complexity of `find_max_value_2()` in terms of the number of elements of `x`?
**d) (2p)** Implement unit tests that check both `find_max_value()` and `find_max_value_2()`. Check if the functions return the correct values and identify wrong input.


Figure 1: Source: Google maps