# Debugging

STAT/BIOS 823

Homework 14

## Directions

Using RMarkdown in RStudio, complete the following questions. Launch RStudio and open a new RMarkdown file and save it on your working directory as a name.Rmd file. At the end of the activity, knit your document to pdf generated from RMarkdown+Knitr and submit your homework on the Blackboard.

## Q1

In order to calculate $f(x) = e^{-x^2}$

A function called **calculate.exp()** was defined below. However, this function is buggy.

```
calculate.exp <- function(my.number){
  exp.num <- (-my.number) ^ 2
  result <- exp(exp.num)
  return(result)
}

calculate.exp(1)
```

```
## [1] 2.718282
```

After looking at the result when input equals of 1, we believe the **calculate.exp(1)** does not produce the correct reuslt.

(a). Modify this function by adding one line of code. After modification, when this function is being called, it prints the value of **exp.num**.

(b). Call the modified function with input value equals to 1.

(c). Add one line of code after the function is being defined, so that next time the **calculate.exp()** function is being called, the function enters into debug mode.

(d). Describe the difference between **debug()** and **debugonce()**. No code is required to answer this subquestion.

(e). Insert a **browser()** function in the middle of your **calculate.exp()**. So that a browser window opens after **exp.num** was calculated.

(f). Describe an equivlent alternative to do this using RStudio debugging tools.

## Q2

Suppose we were asked to generate a function to run simulations with steps below:

Step 1: Sample 10 values from a normal distribution with **mean = mu** and **SD = 1**.

Step 2: Calculate the sample mean of the 10 values being simulated.

Step 3: Repeat 100,000 times and save sample means.

```
my.simulation <- function(mu){
  # initiate an empty value
  mean.vec <- NA
  for (i in 1:100000){
    # Step 1
    simu.data <- rnorm(n = 10, mean = mu, sd = 1)
    # Step 2
    mean.simu <- mean(simu.data)
    # Step 3
    mean.vec <- c(mean.vec, mean.simu)
  }

  result <- mean(mean.vec)
  return(result)
}
# print the time it takes to execute the function
system.time(print(my.simulation(10)))
```

```
## [1] NA

##    user  system elapsed
##  19.596   9.845  29.519
```

We called this function by using **mu = 10** as input, we got an NA value as a result and it took about 30 seconds to get this result.

(a). Modify this function by reduce the number of iterations so that it returns the same NA result with same **mu = 10** as input but takes shorter time to run.

(b). Debug the function. After your debug, call the function with **mu = 10** and print the result. Your result should be a numberic number close to 10.

## Q3

Suppose that we are interested in finding runs of k consecutive TRUE values in a vector **x** that contains TRUE/FALSE. Below is a buggy version of a get.runs() function.

```
get.runs <- function(x, k){
  n <- length(x)
  runs <- NULL
  for (i in 1:(n-k)){
    if(all(x[i:(i+k-1)] == TRUE)){
      runs <- c(runs, i)
  }
  return(runs)
}
```

(a). If you copy and paste the buggy version of this function and try to run it, you will get a **"Error: Incomplete expression:"**. Fix this error. (Hint: by adding matching parentheses, brackets, braces)

(b). By looking at a input vector **x = c(TRUE, FALSE, FALSE, TRUE, TRUE, TRUE, FALSE, TRUE, TRUE)** we see there are three places where we can find **k = 2** TRUEs, they are at 4,5,8. However, after you fixed the error in part (a). this function returns a vector of **c(4,5)** instead of **c(4,5,8)**. By using different debugging tools, find out where the bug is and fix the bug. Show your bug free function and execute the below code so that you get a result of vector **c(4,5,8)**.

```
# this function below should return a vector
# of (4,5,8) because there are a run of two TRUEs
# in those indices.
get.runs(c(TRUE, FALSE, FALSE, TRUE, TRUE, TRUE, FALSE, TRUE, TRUE), 2)
```

```
## [1] 4 5
```