Problem 1

To have a function that lists the objects in the workspace in a nicely formatted manner, serveral auxiliary printing function (*printObj*, *printUnit*) is created first for cleaner coding.

```
## Aux Printing Fun: Define printing control global parameter
wid = 20  # object name priting width
digit = 10  # byte size printing digits
## Print out file size according to auto-match units
printUnit <- function(obj) {
    units = c("", "k", "M", "G")
    i = 1
    while (obj >= 1024) {
        obj = obj/1024
        i = i + 1
    }  # get proper units
    if (i == 1) {
        cat(format(obj, width = digit, justify = "right"), "\n", sep = "")
    } else {
        cat(format(as.integer(obj), width = digit - 1, justify = "right"), units[i],
            "\n", sep = "")
    }
}
## Print out the object list in a nicely formatted way
printObj <- function(objList, objName, pretty) {
    cat(format("object", width = wid), format("bytes", width = digit, justify = "right"),
        "\n", sep = "")
    if (pretty == FALSE) {
        # pretty==TRUE: it will call the printUnit to print out proper units for sizes
        t <- sapply(1:length(objList), function(i) {
            cat(format(objName[i], width = wid), format(objList[i], width = digit,
                justify = "right"), "\n", sep = "")
        })
    } else {
        # default:      it will just print out number of bytes
        t <- sapply(1:length(objList), function(i) {
            cat(format(names(objList)[i], width = wid))
            printUnit(objList[i])
        })
    }
}
```

The main function will list objects ordered by size from largest to smallest; it allows the user to either specify how many objects to list or to specify that objects larger than a certain size to be listed. The argument *pretty*, when TRUE, results in the size of the objects being printed with the units tailored by function *printUnit*. By default, *lsObj<-function(numls=100, sizeB=0, pretty=FALSE)* will give sensible results. Also, the main function shown below is already written to be able to print out only objects in the frame of the function call when called from within another function.

```
lsObj <- function(numls = 100, sizeB = 0, pretty = FALSE) {
    listLs <- ls(envir = parent.frame())  # object list
    sizeLs <- sapply(listLs, function(x) {
        object.size(get(x, envir = parent.frame(n = 4)))
    })
    # get all object sizes from within the called frame;
    # n=4:FUN->sapply->lsObj->call(lsObj)
    tmpEnv <- NULL
    nameEnv <- NULL  # store object sizes and names within a certain user-defined ENV
    tmpFun <- NULL
    nameFun <- NULL  # store object sizes and names within a function closure
    for (item in listLs) {
        # elborate objects to find hidden ones
        objitem <- get(item, envir = parent.frame())
        if (is.environment(objitem)) {
            # get sizes and store names with ENV prefix
            tt <- sapply(ls(envir = objitem), function(x) {
                object.size(get(x, envir = objitem))
            })
            tmpEnv <- c(tmpEnv, tt)
            nameEnv <- c(nameEnv, paste(item, "$", ls(envir = objitem), sep = ""))
        }
        if (is.function(objitem) && identical(environment(objitem), parent.frame()) ==
            FALSE) {
            # get sizes and store names with FUN prefix
            tt <- sapply(ls(envir = environment(objitem)), function(x) {
                object.size(get(x, envir = environment(objitem)))
            })
            tmpFun <- c(tmpFun, tt)
            nameFun <- c(nameFun, paste(item, "$", ls(envir = environment(objitem)),
                sep = ""))
        }
    }
    names(tmpEnv) <- nameEnv
    names(tmpFun) <- nameFun
    if (length(tmpEnv) > 0)
        {
            sizeLs <- c(sizeLs, tmpEnv)
        }  #if there is use-defined ENV
    if (length(tmpFun) > 0)
        {
            sizeLs <- c(sizeLs, tmpFun)
        }  #if there is FUN closure
    sizeLs <- sizeLs[sizeLs[] > sizeB]  # find objects larger than sizeB(bytes)
    if (numls > length(sizeLs))
        {
            numls = length(sizeLs)
        }  #prune incorrect listing number
    sizeLs <- sort(sizeLs, decreasing = TRUE)[1:numls]  # find the numls largest objects
    nameLs <- names(sizeLs)  # get the name of vector
    printObj(sizeLs, nameLs, pretty)  # print using the aux functions
}
```

The above main function has the capability to deal with user-created environments and objects within a closure. It elborates the objects obtained from the *ls()* function to see if there are user-defined ENV or FUN closure. When detected, it then goes one frame down to the enclosing environment to get the hidden objects and assign names to them with ENV/FUN prefix.

To create objects of various sizes for testing, I generated vectors of random normals of different lengths, along with serveral hidden ENV/FUN examples for testing.

```
#### Generate objects test cases
ob0 <- rnorm(1)
ob1 <- rnorm(10)
ob2 <- rnorm(100)
ob3 <- rnorm(1000)
ob4 <- rnorm(10000)
ob5 <- rnorm(1e+05)
ob6 <- rnorm(1e+06)
ob7 <- rnorm(1e+07)
```

```
## **********Test Case 1**********
## lsObj(5,128)

## object                   bytes
## ob7                      8e+07
## ob6                      8e+06
## ob5                      8e+05
## ob4                      80040
## ob3                       8040

## **********Test Case 2**********
## lsObj(sizeB = 1024)

## object                   bytes
## ob7                      8e+07
## ob6                      8e+06
## ob5                      8e+05
## ob4                      80040
## ob3                       8040

## **********Test Case 3**********
## lsObj(numls = 3)

## object                   bytes
## ob7                      8e+07
## ob6                      8e+06
## ob5                      8e+05

## **********Test Case 4**********
## lsObj(pretty = TRUE)

## object                   bytes
## ob7                        76M
## ob6                         7M
## ob5                       781k
## ob4                        78k
```

```
## ob3                          7k
## ob2                        840
## ob1                        168
## ob0                         48
```

```
testFunc <- function () {
             data <- rnorm(10000); data2 <- rnorm(100000)
             myFun <- function(theta) {
                       dist <- rnorm(theta); return(exp(dist / theta)); }
lsObj()}
testEx <- function () {
           x <- rnorm(100000); data <- rnorm(1000)
           e <- new.env(); e$a <- x # an object hidden in an environment
           e2 <- new.env(); e2$a2 <- x; e2$b2 <- 52
           myFun <- function(theta) {
                     dist <- rnorm(theta); return(exp(dist / theta)); }
           myFun1 <- function(theta) {
                      dist <- rnorm(theta); return(exp(dist / theta)); }
           myFun2 <- with(list( data = x ), # an object hidden in a closure
                          function(theta) {
                          dist <- rdist(data); return(exp(dist / theta)); } )
            myFun3 <- with(list( data2 = x ),
                          function(theta2) {
                          dist <- rdist(data); return(exp(dist / theta2)); } )
lsObj()}
```

```
## **********Test Case 5**********
## lsObj() in testFunc()

## object                    bytes
## data2                     8e+05
## data                      80040
## myFun                      6072

## **********Test Case 6**********
## lsObj() in test testEx()

## object                    bytes
## x                         8e+05
## e$a                       8e+05
## e2$a2                     8e+05
## myFun2$data               8e+05
## myFun3$data2              8e+05
## data                       8040
## myFun                      6072
## myFun1                     6072
## myFun2                     6072
## myFun3                     6072
## e                            56
## e2                           56
## e2$b2                        48
```