Sixth R function: path to a file, listing file names in a folder, string search, constructing and writing a dataframe

Alan E. Berger December 17, 2020

version 1

available at https://github.com/AlanBerger/Practice-programming-exercises-for-R

Construct a function that will search for filenames in a given directory that contain specified patterns and output information on those files

Introduction

This is the sixth in a sequence of programming exercises in "composing" an R function to carry out a particular task. Several of these "exercise files" likely
will take several sessions to master the content - so one should not expect to "breeze through this" quickly. The material below centers on using data frames, a core area of R programming.

The idea of this set of exercises is to practice correct use of R constructs and built in functions (functions that "come with" the basic R installation), while learning how to "put together" a correct sequence of blocks of commands that will obtain the desired result.
Note these exercises are quite cumulative - one should do them in order.

In these exercises, there will be a statement of what your function should do (what are the input variables and what the function should return) and a sequence of "hints", or an outline of a sequences of steps to use in writing the function. To get the most out of these exercises, try to write your function using as few hints as possible.
Note there are often several ways to write a function that will obtain the correct result. For these exercises the directions and hints may point toward a particular approach intended to practice particular constructs in R and a particular line of reasoning, even if there is a more efficent way to obtain the same result.
There may also be an existing R function or package that will do what is stated for a given practice exercise, but here the point is to practice formulating a logical sequence of steps, with each step a section of code, to obtain a working function, not to find an existing solution or a quick solution using a more powerful R construct that is better addressed later on.

Motivation for this exercise

In my hands, searching for files that have one or more specified text strings within the file name using my computer's search facility has not always given back what I thought it should. The function I constructed in R to do this is also a good example for discussing: file paths; how to get a list of files in a folder (I am using **folder** and **directory** as synonyms) that contain a given pattern (meaning character string);
how to check if a character string str2 contains a given character string str1; and several ways to construct a data frame "piece by piece". A typical way to write a data frame out to a file is also displayed. Note I have included a brief introduction to more advanced ways of doing text string searches (*regular expressions*) since this is a natural setting to do so, but for the first read through it is fine to not worry about regular expressions (leave that material for future reference). The programming exercises here do *not* require you to know about regular expressions.

A working version of the desired function,

search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)

will be given, and then you will be asked to make several modifications to it.

The steps in this function are as follows: first it will get a character vector containing the names of all the files in the specified directory. It will then check each of these file names to see if it contains, somewhere in the file name, each of the entries in the character vector **search.strings** (one can also specify that an entry occurs at the beginning or at the end of the file name as detailed below). If none of the file names contain all the strings in the search, the function exits with a message saying there were none. Otherwise the function returns, in a data frame, each of the file names that contain all the strings in search.strings (in column 1 of the returned data frame), along with the time and day the file was last modified (in column 2), and the file size in bytes (in column 3). It also writes out the data frame to a tab delimited text file called **scrlisting.txt** located in the directory being searched (I use `scr` as an abbreviation for "scratch" indicating a temporary object or file - here the user can look over the output file (best opened in Excel or an equivalent), and the user can rename it to keep it if desired).

Note file names are best constructed using only standard upper and lower case letters and numbers and - (dash), _ (underscore) and `.` (period). For R and Unix the delimiter between directories in a path is `/` (forward slash). Other characters may be interpreted by R (and Unix style operating systems) as "special characters" (giving instructions to R or to the operating system) and so should not be used in file or folder names. Also, blanks are best not used in file or folder names that will be used in R or in file or folder names that will be used for example in Git (based on Unix) and GitHub (even though Windows is breathtakingly lenient
in what characters are allowed in file names).

Background

First, let's review **directory paths**. Not correctly handling what directory on your computer R will "look for" a file you have "asked R" to read leads to frustrating error messages of the form

```
filename <- "some.csv.file"  # csv stands for "comma separated values"
#                              (for a text file with commas as the delimiter)
# some.csv.file does not exist in my R working directory
# so the line below will result in an error message
df <- read.csv(filename)  # read in the data in the specified file as a data frame
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
  cannot open file 'some.csv.file': No such file or directory
```

What this error message is saying is that R can't open the file it was going to try to **r**ead as a **t**ext file (that is what the "rt" in the error message above means) because there was no such file in the folder where R "thought it was supposed to find it".

The R working directory is where R will "look for" a file it has been asked to read (unless R is "told" to find it somewhere else as described below), and where R will write a file (unless another location is specified).

If one does **getwd()** R will return the **full path** for the R working directory. In a Unix operating system, a full path name for (to) a given directory will start with `"/"` (the **root** directory). In my Windows PC and looking at the file system with File Explorer, a full path starts with `"C:\"` but R has the Unix viewpoint for path names and uses "forward slashes", so with R one needs to use `"C:/"` This is because in Unix, the backslash `\` is an "escape" used to tell the operating system to interpret the character that follows in some special way (for example `sep = "\t"` specifies that the column separator is a tab in read.table and write.table); `\` can also "tell" the operating system (or some function) to interpret the following character simply as that text character, rather than as an instruction to the operating system; there will be more on using `\` later).

I am going to use the analogy of folders (here a folder is the same thing as a directory) being boxes in a room. The room is the root directory. Each box can contain files as well as other boxes (i.e., folders). So if boxA is in the room, and it contains boxB, and boxB contains boxC, and boxC contains some

file called file1.in.boxC.csv; then on a Unix computer: the **full path** (also called the **absolute path**) to boxA is `"/boxA"` and the full path to boxB is `"/boxA/boxB"` etc. and the full path to file1.in.boxC.csv is `"/boxA/boxB/boxC/file1.in.boxC.csv"` I am putting the path in quotes since one "gives" R a path location as a character string. On a Windows computer: the **full path** (also called the **absolute path**) to boxA is (for use in R): `"C:/boxA"` and the full path to boxB is `"C:/boxA/boxB"` etc. and the full path to file1.in.boxC.csv is `"C:/boxA/boxB/boxC/file1.in.boxC.csv"`

Since I am using a Windows computer, from now on I am going to use the Windows form. If I want R to read in data from file1.in.boxC.csv and this file is suitable for reading in into a data frame using the read.csv function, then no matter what the R working directory is, we can always use the absolute path name for the file:

```
filename <- "C:/boxA/boxB/boxC/file1.in.boxC.csv"
df <- read.csv(filename)
```

If the R working directory happened to be "C:/boxA/boxB/boxC" then we could read in this file simply by `df <- read.csv("file1.in.boxC.csv")` since this folder being the R working directory means that is where R will look for a given file (unless other directions were given).

If the R working directory happened to be boxA; then from "that vantage point" (think of already "being in boxA", meaning being in `"C:/boxA"`); then the **relative path** from boxA to boxC is: `"boxB/boxC"` and the **relative path** to file1.in.boxC.csv from boxA is `"boxB/boxC/file1.in.boxC.csv"` and we could read in the file via `read.csv("boxB/boxC/file1.in.boxC.csv")` Relative path names are convenient if we know (correctly) where the file we want is located relative to "where we are" (i.e., the R working directory). Absolute path names (full path names) may be longer, but are always "safe" since they are valid regardless of what the R working directory is. For future reference, note in some settings, a period `.` is used to denote the name of the R working directory (or in a command line window, where you are currently located in the file system).

So in the **search_for_filenames_containing_multiple_patterns_and_output_file_info**(directory, search.strings) function, directory will be an absolute path to the desired folder, or directory will be a relative path (relative to the R working directory) to the desired folder.

The list.files function

The first step in constructing the desired function,

search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)

will be to get a character vector containing the names of all the files in the specified directory. We will then check each these file names to see if it contains, somewhere in the file name, each of the entries in the character vector search.strings (using the R **grep** function). The **list.files** function can be used to get all the file and folder names in a given directory. If you haven't previously done so, do `?list.files` to see what the options are (or Google it - sometimes explanations on the web are easier to follow) - it's always a good idea to look over the information on a function every time you encounter a function you aren't familiar with, to be sure of what it does, and also just to get an idea of what options are available, even if you are only going to use the function in a basic way for the time being.

For our purposes, we are going to use list.files in the following way

```
filenames <- list.files(directory, pattern = search.strings[1],
                        full.names = FALSE, ignore.case = TRUE)
```

Then filenames will contain only the file and folder names located in directory which contain the character string that **pattern** is set to somewhere in their name (here pattern is set to the first entry of search.strings).

(If pattern is not specified in a call to list.files then no files will be eliminated through a pattern seach, which is the default.) full.names = FALSE means the names of the files will not contain the folder names part of the path to the file. ignore.case = TRUE means the pattern match will be case insensitive (upper and lower case letters are considered to be the same). **directory** is an absolute path or a path relative to the R working directory to the desired folder to be searched. `directory <- "."` is a short way to specify the R working diectory, rather than using its full path name.

If, for example, "subfolder" was a folder in the R working directory and one wanted to read in each of the files in subfolder containing csv in their file name, one at a time, one could do

```
filenames <- list.files("subfolder", pattern = "csv", full.names = TRUE)
# to use read.csv on these files one needs to have   full.names = TRUE   so R can find them
nfiles <- length(filenames)  # so can, if desired, check that there are some files
for (file in filenames) {
   df <- read.csv(file)  # assuming these files are suitable for using read.csv
#  do something with the data in the read in data frame df
}
# note if filenames is empty (i.e., character(0)) then the loop does nothing
# with no warning message
```

There are some more features in what one can set pattern to, which we will briefly touch on in the following. One may well want to come back to this paragraph later on when you have occasion to need more advanced capabilities with pattern matching (how to use the capabilities of "**regular expressions**" in text string pattern matching). To be more specific and specify files that have .csv in their name, one needs to have `pattern = "\\.csv"` since otherwise in pattern matching, . is a "special character" meaning it will match any single character (yes one needs 2 backslashes in front of the period here in order to get R to interpret the period as a period, due to the way the *escape*
character \ (backslash) is processed within a character string that pattern has been set to).
Getting a bit "further into the weeds", `"\\.csv$"` means .csv occurring at the **end** of the file name and `"^abc"` would mean abc occurring at the **beginning** of the file name. This is a brief introduction to "regular expressions" which allow powerful additional instructions within a text string that
can be used as the value of the pattern argument within list.files (and within the grep function described below).

The R grep function

We will use the **grep** function with the following option choices

xmatching <- grep(pattern, x, ignore.case = TRUE, value = TRUE)

Here pattern is a character string (which can be a "regular expression"); x is a character vector; and xmatching returned by grep will be the character vector containing only the entries x[k] of x which contain (also phrased as **match**) the text string that pattern has been set to, somewhere in the character string x[k] (or at the beginning or at the end of the character string x[k] if so specified). Here value = TRUE means return the entries x[k] for which pattern is a match, rather than returning the indices k for which there is a match (which is what is returned when value = FALSE). If there are no entries of x for which there is a match for pattern, or if x is an empty character vector, i.e., a character vector of length 0 which is given by character(0); then grep returns character(0) when value is TRUE, integer(0) when value is FALSE. ignore.case = TRUE means upper and lower case letters are considered to be the same for the purpose of matching.

How the function **search_for_filenames_containing_multiple_patterns_and_output_file_info** works

After checking that search.strings is a non-empty character vector, list.files is used to get a character vector of the file and folder names that contain (match) the first entry of **search.strings** Then any names of

directories are eliminated. Then for each of the rest of the entries of search.strings (if there is more than 1 entry) the grep function is used to retain only the file names containing that entry of search.strings After all the entries of search.strings are checked, one has a vector of the desired file names. If no files remain, the function exits with a message to the effect that there were no files matchng all the entries of search.strings If there were file(s) that matched all the entries, a data frame, df, is constructed containing the file names in column 1. The **file.info** function is used to find the last modification date for each of these files (placed in column 2 of df), and the file size in bytes (placed in column 3 of df). The data frame df is then written to a file located in directory using write.table, and then df is returned.

Exercises

A working version of this function is given below. There are a couple of possible techniques for constructing the data frame df in this setting, and the first set of exercises (given below the function) will be to program several useful to know ways of doing so, following specified directions. In the next (7th) programming exercise file, one will construct a modified version of this function called

search_for_filenames_containing_any_of_the_patterns_and_output_file_info(directory, search.strings)

that will return the vector of file names that match ANY entry of search.strings, rather than ALL the entries of search.strings The final exercise in this sequence (also in the 7th programming exercise file) will be to construct a version of this function that will return the vector of file names that **DO NOT** match **ANY** entry of search.strings.

Here is a working version of the function that is the subject of this programming exercise:

```
search_for_filenames_containing_multiple_patterns_and_output_file_info <-
                                        function(directory, search.strings){

# directory is an absolute path (full path) or a path relative to the R working directory to the
# folder to be searched. If want to search the R working directory itself,
# can set directory = "."
# or could set directory to be the full path to the R working directory.

# search.strings is a character string vector

# Return a data frame of the file names (not including folders) in directory that contain
# ALL the entries of the search.strings vector somewhere in their file name,
# the search will be case insensitive (treats lower case and upper case letters as the same).

# Use R's list.files function to list all the files (file names) in directory that contsin
# search.strings[1] in their name, then eliminate names of folders, then use R's
# grep function to find out, for each file name, whether or not each
# character string in search.strings appears in the file name.
# For the files whose names contain all the character strings in search.strings, use
# R's file.info function to get the file size and last modification time.

# The output data frame will contain the file size and the last modification time
# for each file that has each member of search.strings somewhere in its file name.
#
# We will get the file names without the folder path leading to the files included in the name.

# check that search.strings is a non-empty character vector

ns <- length(search.strings)
```

```
if(ns < 1) stop("no entries in search.strings")
if(!is.character(search.strings)) stop("search.strings is not a character vector")

# We will return a data frame whose first column contains the files in directory
# that contain each character string in the search.strings vector somewhere in their name.
# The second column will contain the last time (and date) the file was modified,
# and the third column will be the file size (in bytes).

# The first step is to get all the file names in directory that contain the first string in
# search.strings somewhere in their file name; do not include the path
# to the file in the file name.

filenames <- list.files(directory, pattern = search.strings[1],
                        full.names = FALSE, ignore.case = TRUE)

# If at any point there are no files then return a message that there are none

if(length(filenames) == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# Eliminate names of any folders (directories) that are in the filenames character vector.
# paste(directory, "/", filenames, sep = "") will get the filenames including either
# the relative path from the R working directory or the absolute path (depending on what
# directory is); use these names in the R dir.exists function to check for
# folder (directory) names in filenames

filenames <- filenames[!dir.exists(paste(directory, "/", filenames, sep = ""))]
# exclude directory names

if(length(filenames) == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# if there is more than 1 search string, search over the rest of them
if(ns > 1) {
   for(k in 2:ns) {
   filenames <- grep(search.strings[k], filenames, ignore.case = TRUE, value = TRUE)
#               grep returns the subset of filenames that contain
#               search.strings[k] in the file name
#               if filenames is an empty character vector, grep will just return
#               an empty character vector
   }
}

nf <- length(filenames)
if(nf == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}
```

```
# If got to here, at least 1 file has all the character strings in search.strings
# in its name, so get the information on these file(s) into a data frame.

################################### get the data frame to be output

# Get the information into a character matrix, then will convert it to a data frame.

file.matrix <- matrix("0", nrow = nf, ncol = 3)
dfcolnames <- c("file.name", "modif.date", "size.in.bytes")
for(k in 1:nf) {
    finfo <- file.info(paste(directory, "/", filenames[k], sep = ""))
# needed to include the path to the file so file.info can locate it
    file.matrix[k, 1] <- filenames[k]
    file.matrix[k, 2] <- as.character(finfo$mtime)
    file.matrix[k, 3] <- as.character(finfo$size)
}
df <- data.frame(file.matrix, stringsAsFactors = FALSE)
colnames(df) <- dfcolnames

################################### finished getting the data frame to be output

# Write the data frame out to a tab delimited text file called scrlisting.txt in directory
# (i.e., in the folder specified by the argument directory this function was called with).

outpfilename <- paste(directory, "/", "scrlisting.txt", sep = "")
# One can rename this "scratch file" as desired after viewing it (best viewed in Excel).
write.table(df, file = outpfilename,
            append = FALSE, quote = FALSE, sep = "\t",
            row.names = FALSE, col.names = TRUE)
# This call to write.table will write out a data frame
# as one would usually want; it specifies the column separator to be a tab

return(df)
}
```

Test runs on my computer

I have constructed a folder called test_dir in my R working directory which is "C:/berger/R_course_cs" containing a small number of files with names I have picked to conveniently test the

`search_for_filenames_containing_multiple_patterns_and_output_file_info`

function. Here are all the file names in the test_dir folder; note here folder001txtcsv is a folder not a file, and so should not be included in any search result:

```
directory <- "test_dir"
list.files(directory)  # 9 files and 1 folder
 [1] "001.csv"         "002.csv"         "003.txt"
 [4] "004csvfile.txt"  "005txt2csv"      "1.csv"
 [7] "308.csv"         "folder001txtcsv" "scrlisting.txt"
[10] "txt.csv"
```

The results of checking this function with some appropriate choices for search.strings are given below (one can easily check "by eye" whether or not the function above was working correctly). You can easily recreate a test.dir folder in your computer, located as a folder in your R working directory, and then create in it arbitrary files with these file names (or copy in some files and rename them). Also create the folder folder001txtcsv as a subfolder of test_dir (i.e., as a folder in test_dir). Then you can use this setup to test your functions (note the file sizes and dates will be different from mine).

```
# here is a sequence of commands that can be used to test this function

directory <- "test_dir"
# note the  scrlisting.txt  file contains the output from
# the previous run of the search function so its modification date and file size will vary
list.files(directory)

search.strings <- c("00", "txt")  # should NOT get the folder name folder001txtcsv
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)

search.strings <- c("1")
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)

search.strings <- c("txt")
scr <-
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)
nrow(scr)  # should get 5

search.strings <- ".txt"  # note that the period here is interpreted as: any 1 character
#                  since for pattern matching a period is a special character with that meaning
#                  should NOT get the file txt.csv
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)

search.strings <- "\\.txt"  # the two backslashes in front of the period
#                             result in the period being interpreted as an actual period
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)

search.strings <- "^txt"  # the special character ^ here means: occurs at the
#                           beginning of the string being matched
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)

search.strings <- "txt$"  # the special character $ here means: occurs at the end
#                           of the string being matched
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)

search.strings <- "AD"  # test what happens when there are no matching file names
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)
```

Here are the results of doing the test runs:

```
# results of test runs on my computer
# you should get the same file names (the dates and sizes for your files will be different)

directory <- "test_dir"
# note the  scrlisting.txt  file contains the output from
# the previous run of the search function so its modification date and file size will vary
```

```
list.files(directory)
 [1] "001.csv"         "002.csv"         "003.txt"
 [4] "004csvfile.txt"  "005txt2csv"      "1.csv"
 [7] "308.csv"         "folder001txtcsv" "scrlisting.txt"
[10] "txt.csv"

search.strings <- c("00", "txt")  # should NOT get the folder name folder001txtcsv
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)
       file.name          modif.date size.in.bytes
1        003.txt 2020-04-26 12:10:47             9
2 004csvfile.txt 2020-12-13 10:04:08             9
3     005txt2csv 2020-12-13 10:03:14            11

search.strings <- c("1")
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)
  file.name          modif.date size.in.bytes
1   001.csv 2012-09-21 16:00:20         31271
2     1.csv 2020-04-26 14:21:53         28278

search.strings <- c("txt")
scr <-
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)
nrow(scr)  # should get 5
[1] 5

search.strings <- ".txt"  # note that the period here is interpreted as: any 1 character
#                  since for pattern matching a period is a special character with that meaning
#                  should NOT get the file txt.csv
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)
       file.name          modif.date size.in.bytes
1        003.txt 2020-04-26 12:10:47             9
2 004csvfile.txt 2020-12-13 10:04:08             9
3     005txt2csv 2020-12-13 10:03:14            11
4 scrlisting.txt 2020-12-16 13:32:47           211

search.strings <- "\\.txt"  # the two backslashes in front of the period
#                             result in the period being interpreted as an actual period
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)
       file.name          modif.date size.in.bytes
1        003.txt 2020-04-26 12:10:47             9
2 004csvfile.txt 2020-12-13 10:04:08             9
3 scrlisting.txt 2020-12-16 13:32:47           180

search.strings <- "^txt"  # the special character ^ here means: occurs at the
#                           beginning of the string being matched
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)
  file.name          modif.date size.in.bytes
1   txt.csv 2020-04-26 12:11:18             9

search.strings <- "txt$"  # the special character $ here means: occurs at the end
#                           of the string being matched
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)
       file.name          modif.date size.in.bytes
1        003.txt 2020-04-26 12:10:47             9
```

```
2 004csvfile.txt 2020-12-13 10:04:08             9
3 scrlisting.txt 2020-12-16 13:32:47            67

search.strings <- "AD"  # test what happens when there are no matching file names
search_for_filenames_containing_multiple_patterns_and_output_file_info(directory, search.strings)
[1] "no files contain all the search strings"
[1] "no files contain all the search strings"
```

First exercise

The first thing to do is some trial runs of the function above on your computer. Create the test.dir folder as above and run the tests.
Or pick a directory (an absolute path or a relative path (relative to your R working directory)) and choose search.strings so you will get a match to only a couple of files so you know what matches you should get, and verify that the function search_for_filenames_containing_multiple_patterns_and_output_file_info is indeed doing what has been described (finding the correct files for a given value of search.strings, and outputting the correct information on these files).

In the function above, the construction of the data frame df containing information on the files whose names match all the strings in search.strings is done between the two comment lines

```
################################### get the data frame to be output
################################### finished getting the data frame to be output
```

The first exercise is to use rbind to obtain df in the following fashion. The only needed changes to the function above will occur between these 2 comment lines (unless you decide to change the function name and/or add other comment lines). First initialize df to be an empty data frame by: `df <- data.frame()` One can rbind any data frame to an empty data frame.

Then successively for each file whose name matches all the strings in search.strings: use the **data.frame** function to get a 1 row data frame df1 containing the file name, the last modification date (as a character string), and the file size in bytes (use the file.info function as done above). Have the names of the columns of df1 be those in the vector `dfcolnames <- c("file.name", "modif.date", "size.in.bytes")` This could be done in the call to data.frame, or afterwards by doing `colnames(df1) <- dfcolnames` Use the option stringsAsFactors = FALSE in the call to data.frame (to get character, not factor, data for character columns). You don't need to convert finfo$size (which is numeric) to character (we did so explicitly in the original version of the function since we were putting everything into a character matrix). Then do the rbind: `df <- rbind(df, df1)`

Try doing this modification and check to see that it works correctly - there are no changes needed other than between the two comment lines

```
################################### get the data frame to be output
################################### finished getting the data frame to be output
```

Here is a version using rbind:

```
search_for_filenames_containing_multiple_patterns_and_output_file_info <-
                                        function(directory, search.strings){

# directory is an absolute path (full path) or a path relative to the R working directory to the
# folder to be searched. If want to search the R working directory itself,
# can set directory = "."
```

```r
# or could set directory to be the full path to the R working directory.

# search.strings is a character string vector

# Return a data frame of the file names (not including folders) in directory that contain
# ALL the entries of the search.strings vector somewhere in their file name,
# the search will be case insensitive (treats lower case and upper case letters as the same).

# Use R's list.files function to list all the files (file names) in directory that contsin
# search.strings[1] in their name, then eliminate names of folders, then use R's
# grep function to find out, for each file name, whether or not each
# character string in search.strings appears in the file name.
# For the files whose names contain all the character strings in search.strings, use
# R's file.info function to get the file size and last modification time.

# The output data frame will contain the file size and the last modification time
# for each file that has each member of search.strings somewhere in its file name.
#
# We will get the file names without the folder path leading to the files included in the name.

# check that search.strings is a non-empty character vector

ns <- length(search.strings)
if(ns < 1) stop("no entries in search.strings")
if(!is.character(search.strings)) stop("search.strings is not a character vector")

# We will return a data frame whose first column contains the files in directory
# that contain each character string in the search.strings vector somewhere in their name.
# The second column will contain the last time (and date) the file was modified,
# and the third column will be the file size (in bytes).

# The first step is to get all the file names in directory that contain the first string in
# search.strings somewhere in their file name; do not include the path
# to the file in the file name.

filenames <- list.files(directory, pattern = search.strings[1],
                        full.names = FALSE, ignore.case = TRUE)

# If at any point there are no files then return a message that there are none

if(length(filenames) == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# Eliminate names of any folders (directories) that are in the filenames character vector.
# paste(directory, "/", filenames, sep = "") will get the filenames including either
# the relative path from the R working directory or the absolute path (depending on what
# directory is); use these names in the R dir.exists function to check for
# folder (directory) names in filenames

filenames <- filenames[!dir.exists(paste(directory, "/", filenames, sep = ""))]
# exclude directory names
```

```
if(length(filenames) == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# if there is more than 1 search string, search over the rest of them
if(ns > 1) {
   for(k in 2:ns) {
   filenames <- grep(search.strings[k], filenames, ignore.case = TRUE, value = TRUE)
#               grep returns the subset of filenames that contain
#               search.strings[k] in the file name
#               if filenames is an empty character vector, grep will just return
#               an empty character vector
   }
}

nf <- length(filenames)
if(nf == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# If got to here, at least 1 file has all the character strings in search.strings
# in its name, so get the information on these file(s) into a data frame.

################################### get the data frame to be output

# Get the desired output data frame using rbind

df <- data.frame()
dfcolnames <- c("file.name", "modif.date", "size.in.bytes")

for(k in 1:nf) {
    finfo <- file.info(paste(directory, "/", filenames[k], sep = ""))
# needed to include the path to the file so file.info can locate it
    df1 <- data.frame(filenames[k], as.character(finfo$mtime), finfo$size, stringsAsFactors = FALSE)
    colnames(df1) <- dfcolnames
    df <- rbind(df, df1)  # the rbind command stacking df1 below df
}

################################### finished getting the data frame to be output

# Write the data frame out to a tab delimited text file called scrlisting.txt in directory
# (i.e., in the folder specified by the argument directory this function was called with).

outpfilename <- paste(directory, "/", "scrlisting.txt", sep = "")
# One can rename this "scratch file" as desired after viewing it (best viewed in Excel).
write.table(df, file = outpfilename,
            append = FALSE, quote = FALSE, sep = "\t",
            row.names = FALSE, col.names = TRUE)
# This call to write.table will write out a data frame
# as one would usually want; it specifies the column separator to be a tab
```

```
return(df)
}

# do test runs of your function on your computer
```

second exercise

This time construct the data frame df using 3 vectors to "hold" the file names, last modification times, and file sizes. Then when these vectors have been formed, use the data.frame function to create the desired data frame from these 3 vectors. Again, the only section that needs any changes is the one noted in the first exercise that forms the data frame df to be output. Try doing this - a working version is given below.

```
search_for_filenames_containing_multiple_patterns_and_output_file_info <-
                                        function(directory, search.strings){

# directory is an absolute path (full path) or a path relative to the R working directory to the
# folder to be searched. If want to search the R working directory itself,
# can set directory = "."
# or could set directory to be the full path to the R working directory.

# search.strings is a character string vector

# Return a data frame of the file names (not including folders) in directory that contain
# ALL the entries of the search.strings vector somewhere in their file name,
# the search will be case insensitive (treats lower case and upper case letters as the same).

# Use R's list.files function to list all the files (file names) in directory that contsin
# search.strings[1] in their name, then eliminate names of folders, then use R's
# grep function to find out, for each file name, whether or not each
# character string in search.strings appears in the file name.
# For the files whose names contain all the character strings in search.strings, use
# R's file.info function to get the file size and last modification time.

# The output data frame will contain the file size and the last modification time
# for each file that has each member of search.strings somewhere in its file name.
#
# We will get the file names without the folder path leading to the files included in the name.

# check that search.strings is a non-empty character vector

ns <- length(search.strings)
if(ns < 1) stop("no entries in search.strings")
if(!is.character(search.strings)) stop("search.strings is not a character vector")

# We will return a data frame whose first column contains the files in directory
# that contain each character string in the search.strings vector somewhere in their name.
# The second column will contain the last time (and date) the file was modified,
# and the third column will be the file size (in bytes).

# The first step is to get all the file names in directory that contain the first string in
# search.strings somewhere in their file name; do not include the path
# to the file in the file name.
```

```
filenames <- list.files(directory, pattern = search.strings[1],
                        full.names = FALSE, ignore.case = TRUE)

# If at any point there are no files then return a message that there are none

if(length(filenames) == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# Eliminate names of any folders (directories) that are in the filenames character vector.
# paste(directory, "/", filenames, sep = "") will get the filenames including either
# the relative path from the R working directory or the absolute path (depending on what
# directory is); use these names in the R dir.exists function to check for
# folder (directory) names in filenames

filenames <- filenames[!dir.exists(paste(directory, "/", filenames, sep = ""))]
# exclude directory names

if(length(filenames) == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# if there is more than 1 search string, search over the rest of them
if(ns > 1) {
   for(k in 2:ns) {
   filenames <- grep(search.strings[k], filenames, ignore.case = TRUE, value = TRUE)
#               grep returns the subset of filenames that contain
#               search.strings[k] in the file name
#               if filenames is an empty character vector, grep will just return
#               an empty character vector
   }
}

nf <- length(filenames)
if(nf == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# If got to here, at least 1 file has all the character strings in search.strings
# in its name, so get the information on these file(s) into a data frame.

################################### get the data frame to be output

# Get the desired output data frame using vectors

dfcolnames <- c("file.name", "modif.date", "size.in.bytes")
# initialize the 3 vectors that will hold this information on the files
# whose names matched all the members of search.strings

fname <- character(0)
```

```r
fdate <- character(0)
fsize <- numeric(0)

for(k in 1:nf) {
    finfo <- file.info(paste(directory, "/", filenames[k], sep = ""))
# needed to include the path to the file so file.info can locate it
    fname <- c(fname, filenames[k])
    fdate <- c(fdate, as.character(finfo$mtime))
    fsize <- c(fsize, finfo$size)
}

df <- data.frame(fname, fdate, fsize, stringsAsFactors = FALSE)
colnames(df) <- dfcolnames

################################## finished getting the data frame to be output

# Write the data frame out to a tab delimited text file called scrlisting.txt in directory
# (i.e., in the folder specified by the argument directory this function was called with).

outpfilename <- paste(directory, "/", "scrlisting.txt", sep = "")
# One can rename this "scratch file" as desired after viewing it (best viewed in Excel).
write.table(df, file = outpfilename,
            append = FALSE, quote = FALSE, sep = "\t",
            row.names = FALSE, col.names = TRUE)
# This call to write.table will write out a data frame
# as one would usually want; it specifies the column separator to be a tab

return(df)
}

# do test runs on your computer
```

Hope this sequence of exercises was informative and good practice. The next set of exercises will have more practice with using logic and data frames in composing a function.

= =