CSC 121 — Lab Exercise 3

This is a non-credit excercise, which you do not hand in.
You may work on your own or together with another student, as you please.

In this lab, you will write several small R functions that manipulate vectors and lists. You should put the definitions of these functions in an R script file called `lab3defs.R`. You can "source" this script to define the functions, and then test them by typing calls of these functions with various arguments. You should start with just the first function below, test it and modify it until it is working, then go on to writing the next function, and so on.

You should write all these functions using only the facilities of R that we have covered in lectures (through week 4), or that are suggested below, even though some of these problems could be solved more easily using facilities of R that we haven't covered yet.

You can also try out the R debugger. You can ask R to stop when it starts a function by using `debug(f)`, where `f` is the name of the function. When this function is called, R will stop, and display a message. You can then go through the function one step at a time, by typing "n" (or just Enter/Return) for each step. You can also examine variables or see the values of expressions computed from them by just typing the variable name or expression (but you will need to put the name in parentheses if it is the same as a debugger command, such as "n"). You can exit the debugger and let the function currently being debugged finish with "c", or exit the debugger and terminate the program you're currently running with "Q".

RStudio displays the position in the function you are debugging in a separate pane of the window. It also displays the variables in the current function.

You can set more than one function to be debugged. To undo debugging, use `undebug(f)`. Note that redefining a function (eg, by clicking the "Source" button for the script that defines it) will also have the effect of undoing debugging.

You should try using the debugger when writing the functions described below, when you need to find out why something isn't working. If you happen to get your functions to work first time, you can still try out the debugger just to see how it lets you step through a function.

1) Write a function `sum_positive` that takes a numeric vector as its only argument, and computes the sum of the positive (greater than zero) elements of this vector. For example, `sum_positive(c(7,-1,0,2))` should be `9`. If none of the elements are positive, the value returned should be zero.

   Hints: Set a variable `s` to zero, and then do a `for` loop that looks at each element of the vector, adding that element to `s` if it is positive. Finally, write just `s` as the last step in the function, so that the final value of `s` will be the function's value.

2) Write a function called `successive_pairs` that takes a vector as its only argument, and returns a list that contains all the pairs of successive elements of the vector, as vectors of length two — that is, the first element of the list will be the vector of the 1st and 2nd elements of the vector, the second element of the list will be the vector of the 2nd and 3rd elements of the vector, etc. The length of the list returned will be one less than the length of the vector. You can assume that the vector has at least two elements.

Here is an example:

```
> successive_pairs (c("Fred","George","Mary","Jane"))
[[1]]
[1] "Fred"   "George"

[[2]]
[1] "George" "Mary"

[[3]]
[1] "Mary" "Jane"
```

Hints: Recall that you can create a list with zero elements with `list()`, and that assigning to an element of a list that doesn't exist yet will extend the list as needed.

3) Write a function `find_cos_integer` that takes a number, x, as its only argument and returns the smallest non-negative integer, i, such that $|\cos(i) - x| < 0.001$. Here is an example:

```
> find_cos_integer(0.1234)
[1] 38282
> cos(38282)
[1] 0.1224119
```

For this exercise, you will need to use the `abs` function, which gives the absolute value of its argument.

What happens when you try `find_cos_integer(1.2)`? You may find the red STOP sign in RStudio to be useful!

4) Write a function `count_char` that takes two arguments, a vector of strings, `str`, and a string consisting of a single character, `chr`, and returns a vector of numbers the same length as `str` that gives the number of times `chr` occurs in each of the strings in `str`. For example:

```
> count_char( c("fred","mary","george","bert","helen"), "e")
[1] 1 0 2 1 2
```

For this exercise, you will need to use the `nchar` function, which gives the number of characters in a string, and the `substring` function mentioned in the week 1 lecture slides. You will also need to use the `numeric` function, which creates a numeric vector of a given length (containing all zeros) — for example, `numeric(3)` gives the same result as `c(0,0,0)`.