STAT 540: Exercise 3

Let's start by looking at an additional arithmetic operator. Store a random normal sample of size 20 (with mean 10 and standard deviation 3) with the following command:

```
a=rnorm(20,10,3)
a
```

Now try modulo (or remainder) math on this vector to obtain the remainder for division by 3:

```
a%%3
```

What happened? The divisor doesn't have to be an integer; try the same statement using 2.5 in place of 3. Explain your results. Other useful integer-style commands include `floor` and `ceiling`:

```
floor(a)
ceiling(a)
```

The names are intuitive, but explain what these commands do based on the output you have observed. What is the output for these functions for an integer input? Demonstrate using your own example.

Let's take a look at arithmetic operations on two variables of different length. Comment on the output from the following commands:

```
a=1:9
a^c(1,2)
```

The warning message printed is *not* an error message; it simply notes that the length of the exponent is not an integer multiple of **a**. Don't ignore it, though; it could be a sign that a mistake has been made. Let's try a similar operation when **a** is a matrix:

```
a=matrix(a,ncol=3)
a
a^c(1,2,3,4)
```

In what order were the elements of **a** processed? Row by row or column by column? In general, R operates on matrices column by column, but you should always double-check.

Let's look at logical operators now. Prof. Edwards' handout has an extended discussion of various operators; pay attention in particular to his warnings about testing whether two

numeric objects are equal. Situations like the following can occur:

```
b=1.0
c=1.00000000000000001
b==c
```

What happened? The above definition of `c` used 16 0's between the two 1's; what happens when `c` is constructed with 14 0's between the two 1's? Complications like these are why programmers consider testing for equality of real numbers to be a bad practice, unless done carefully.

The `if` statement is one of the real workhorses of function writing. In terms of transforming variables, though, it can often be replaced by more efficient functions since it can only operate on a single element. The following commands generate a half-normal random variable (i.e., $x = |Y|$, where $Y \sim N(0, \sigma^2)$:

```
x=rnorm(10)
x
nx=length(x)
for(i in 1:nx) {if(x[i]<0) x[i]=-x[i]}
x
```

Of course, a simple absolute value command, `x=abs(x)`, would work more easily here. Or if we didn't have an absolute value command available, we could use the `sign` command too:

```
x=rnorm(10)
y=x*sign(x)
x
sign(x)
y
```

You may have run across the `ifelse` statement in Excel. I use it there frequently for recoding. It could be used in the same way in R. Suppose we have a variable that we want to change from factor coding to character coding, such as this vector of cardinal compass directions given in degrees:

```
factor.code=c(0,90,270,90,180,270,0,0,270,90,180,180)
```

In this case, it wouldn't be that hard to recode the four unique values one by one, but `ifelse` can handle it iteratively:

```
Compass=ifelse(factor.code==0, "North", ifelse(factor.code==90, "East",
            ifelse(factor.code==270, "West", "South")))
Compass
```

Explain what that code just did.

The `if` command can execute the same code, but it can be hard to remember the syntax, and hard to keep track of which condition is being tested. Worse yet, it can only be executed an element at a time. Run the code below and confirm that `Compass` contains the same compass directions as `ifelse`. Which approach do you prefer?

```
Compass=NULL
for(i in 1:length(factor.code)){
if(factor.code[i]==0) Compass[i]="North"
else if(factor.code[i]==90) Compass[i]="East"
else if(factor.code[i]==270) Compass[i]="West"
else Compass[i]="South"
}
Compass
```