Family Name: *Given Name:* *Student Number:*

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL EXAMINATIONS 2015

CSC 120H1S (L0201)

Duration - 3 hours

No Aids Allowed

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
T	

Answer all questions in the space provided; if you run out of space, use the back of a page, and indicate where the answer continues.

No books, notes, or calculators allowed.

The twelve questions are worth equal numbers of marks.

You must obtain at least 40% on this final exam to pass the course.

1. In the nine blank spaces below, fill in what R will print if the commands shown are entered one after the other into the R console. (Note that the `>` characters are the R command prompts, not something that was typed in.)

```
> fn <- function (x,y) { z <- rep(x,y); z*10 }
> fn(c(1,3),4)
```

```
> w <- c(8,7,2,3)
> x <- c(5,1,-4)
> y <- 2
> z <- 8
> fn(w[-3],2)
```

```
> fn(9,x[1])
```

```
> y
```

```
> z
```

```
> fn(x,sum(x))
```

```
> gn <- function (x) c(z,x)
> gn(1:3)
```

```
> fn(gn(7),2)
```

```
> gn(fn(4,3))
```

2. Here is the definition of a function called `mystery1`:

```
mystery1 <- function (v) {
    n <- length(v)
    i <- 1
    s <- 0
    while (i <= n/2) {
        s <- s + v[i] - v[n+1-i]
        i <- i + 1
    }
    s
}
```

For each of the three calls of this function below, write down the value that R will print:

```
> mystery1(c(5,9,2,7))
```

```
> mystery1(c(10,7,9,1,2))
```

```
> mystery1(c(101:170,1:70))
```

3. Here is the definition of a function called `mystery2`:

```
mystery2 <- function (L) {
    n <- length(L)
    X <- matrix (0, nrow=n, ncol=n)
    for (j in 1:n)
        X[j,1:j] <- L[[j]]
    X
}
```

Write below what R will print when this function is called as follows:

```
> mystery2 (list (4, c(9,2), 7:9, rep(3,4)))
```

4. Write down a definition for an R function called `alternating_sum` that takes one argument that is a numeric vector, and returns the sum of the elements of this vector with alternating signs, with the sign for the first element being positive, the sign for the second element being negative, etc. Here are two example calls of this function:

```
> alternating_sum(c(50,9,100))      # answer is 50 - 9 + 100
[1] 141
> alternating_sum(c(-1,-7,2,1,10)) # answer is (-1) - (-7) + 2 - 1 + 10
[1] 17
```

5. Write down a definition for an R function called `sum_reverse_diagonal` that takes one argument that is a square matrix, and returns the sum of the elements on the diagonal from the upper-right element to the lower-left element. For example:

```
> M
     [,1] [,2] [,3]
[1,]    5    8    7
[2,]    1    2    6
[3,]    3    9    4
> sum_reverse_diagonal(M)  # should return 7 + 2 + 3
[1] 12
```

6. Write down a definition for an R function called `increasing_sum` that takes three arguments, which you may assume are each single numbers. This function should return as its value the sum of its three arguments if they are in increasing order (each less than or equal to the next), and zero otherwise. For example:

```
> increasing_sum(2,9,10)
[1] 21
> increasing_sum(4,5,1)
[1] 0
```

7. Write down a definition for an R function called `first_and_last_fred` that takes one argument that is a vector of character strings, and returns a vector of two numbers, in which the first number is the index of the first occurrence of `"fred"` in this string vector, and the second number is the index of the last occurrence of `"fred"`. If there is only one occurrence of `"fred"`, the two numbers will be the same. If `"fred"` does not occur in the string vector, the two numbers should both be zero. Here are some examples:

```
> first_and_last_fred (c ("mary","fred","bert","fred","fred","helen"))
[1] 2 5
> first_and_last_fred (c ("mary","fred"))
[1] 2 2
> first_and_last_fred (c ("joe"))
[1] 0 0
```

8. In the three blank spaces below, fill in what R will print if the commands shown are entered one after the other into the R console. (Note that the `>` characters are the R command prompts, not something that was typed in.)

```
> set.seed(789)
>
> sample(1:4)
[1] 3 1 4 2
> runif(1)
[1] 0.4921494
> sample(1:4)
[1] 1 2 4 3
> runif(1)
[1] 0.3462633
>
> set.seed(789)
>
> x <- sample(1:4)
> x



> if (runif(1) < 0.5) y <- 1:4 else y <- x
> y



> z <- sample(1:4)
> z + runif(1)
```

9. Write down a definition for an R function called `all_the_same` that takes one argument that is a vector, and returns `TRUE` if all the elements of this vector are equal to each other, and `FALSE` otherwise. You may assume that none of the elements in the vector are `NA`, and that the vector has at least one element.

10. In the nine blank spaces below, fill in what R will print.

```
> v <- c(10,30,20,80,70)
> u <- c(8,9,1,5,2)
> s <- c("mary","fred","bert","eve","joe")
> v[c(3,5)]




> v[c(TRUE,TRUE,FALSE,FALSE,TRUE)]




> u[v>35]




> s[c(-3,-1)]




> u [u>2 & u<9]




> s [v==80]




> M <- matrix(1:15,nrow=3,ncol=5)
> M
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    7   10   13
[2,]    2    5    8   11   14
[3,]    3    6    9   12   15
>
> M[,-3]






> M[c(1,3),2:4]





> M[1:2,u>4]
```

11. In the nine blank spaces below, fill in what R will print.

```
> # Here is an example to remind you what 'cat' and 'rep' do...
> cat(rep("abc",3),"\n")
abc abc abc
>
> # We define some functions and set some variables...
> doit <- function (x) UseMethod("doit")
> doit.default <- function (x) cat(rep(".",x),"\n")
> more <- function (x) UseMethod("more")
> more.default <- function (x) x+1
> doit.hopper <- function (x) cat(rep("Hop!",x),"\n")
> more.jumper <- function (x) 2*x
> a <- 4; class(a) <- "hopper"
> b <- 3; class(b) <- "jumper"
>
> # Now fill in what is printed after each of the following...
> doit(5)



> doit(more(5))



> doit(a)



> doit(b)



> doit(more(a))



> doit(more(b))



> doit(more(more(a)))



> doit(more(more(b)))



> doit(unclass(more(b)))
```

12. Write down a definition for an R function called `fill_in_height` that takes as its only argument a data frame, with rows representing various people, and several columns that include a column named `sex` and a column named `height`. This function should return a data frame that is the same as its argument, except that values for `height` that are missing (NA) have been filled in with values that the function computes.

    The value for the `sex` variable will be "M" or "F" or NA (missing). The value for the `height` variable will be a number or NA (missing). The `fill_in_height` function should start by checking that there is at least one person with sex "M" for which `height` is not missing, and at least one person with sex "F" for which `height` is not missing. It should stop with an error message if this is not true.

    The `fill_in_height` function should then compute the mean (average) of heights for people for which `sex` is "M" and `height` is not missing, and the mean of heights for people for which `sex` is "F" and `height` is not missing, and use these to fill in values for `height` that are missing. The value to be filled in for a missing height for a person with sex "M" should be the mean that was computed for that sex, and similarly for sex "F". If both the height and the sex for a person are missing, the value filled in for the height should be the average of the means for people with sex "M" and with sex "F" (which may not be the same as the mean height for all people for which the height is not missing).

    Here is an example (but note that your function should work for any data frame that is as described above, not just this one):

```
> d
   sex age height
1    M  51    181
2    F  NA    154
3 <NA>  43    171
4    F  32     NA
5    F  22    146
6    M  31     NA
> fill_in_height(d)
   sex age height
1    M  51    181
2    F  NA    154
3 <NA>  43    171
4    F  32    150
5    F  22    146
6    M  31    181
```

    Write your function definition on the next page. Remember that comparisons with NA always produce NA (not `TRUE` or `FALSE`), and that you can check for NA with the `is.na` function.