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

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL EXAMINATIONS 2016

CSC 120H1S (L0201)

Duration - 3 hours

No Aids Allowed

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
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 eleven questions are worth equal numbers of marks.

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

1. In the ten 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.)

```
> plum <- 100
> bob <- 12
> plum * bob
```

```
> plum * bob + 3
```

```
> plum + bob / 2
```

```
> (plum + bob) / 2
```

```
> a <- 2:6
> b <- c(10,20,30,50,70)
> x <- a + b
> y <- c(a,b)
> a[2]
```

```
> b[4]
```

```
> a <- 1:5
> a + x
```

```
> x[2]
```

```
> y[7]
```

```
> y*100
```

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

```
mystery1 <- function (v) {
    s <- 0
    i <- 1
    j <- length(v)
    while (i < j) {
        s <- s + v[i] * v[j]
        i <- i + 1
        j <- j - 1
    }
    s
}
```

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

```
> mystery1 (c(5,6,10,20))


> mystery1 (1:5)
```

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

```
mystery2 <- function (a, b) {
    if (nchar(a) < nchar(b))
        n <- nchar(a)
    else
        n <- nchar(b)
    repeat {
        if (substring(a,1,n) == substring(b,1,n))
            return (n)
        n <- n - 1
    }
    return (0)
}
```

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

```
> mystery2 ("pineapple", "pickle")


> mystery2 ("apple", "pear")


> mystery2 ("grapefruit", "grape")
```

4. We would like to have an R function, called `sum_by_neg`, that takes two arguments, which are both numeric vectors of the same length, which is at least one. (But the function doesn't have to check that the arguments are actually like this.) The value returned by `sum_by_neg` should be the sum of the elements of its first argument for which the corresponding element in the second argument is negative. The value returned should be zero if none of the elements in the second argument are negative.

   Here are two examples:

```
> sum_by_neg (c(2,7,10,3,1), c(-1,0,3,-2,4))
[1] 5
> sum_by_neg (c(5,4), c(3,7))
[1] 0
```

   a) Write a definition for `sum_by_neg` that uses a "for" loop, and that when subscripting uses only indexes that are single numbers (not vectors).

   b) Write a definition for `sum_by_neg` that does not have a loop, using indexes that are numeric or logical vectors.

5. In the ten 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.)

```
> fun <- function (x, y) (x+y)*2
> fun (3, 5)
```

```
> 10 * fun (2, 2)
```

```
> another_fun <- function (up, down) c (up+1, down-1)
> another_fun (7, 2+2)
```

```
> another_fun (fun(1,2), fun(3,4))
```

```
> yet_another_fun <- function (x) x-1
> fun (7, yet_another_fun(5))
```

```
> p <- 7
> x <- 3
> up <- 2
> down <- 9
> another_fun (7, yet_another_fun(5))
```

```
> one_last_fun <- function (a) { p <- a*x;  p-1 }
> one_last_fun (6)
```

```
> one_last_fun (p)
```

```
> one_last_fun (x)
```

```
> one_last_fun (fun (p,x))
```

6. Write down a definition for an R function called `sum_edges` that takes one argument that is a numeric matrix with at least two rows and two columns, and returns the sum of the elements of this matrix that are on the top, bottom, left, or right edge.

   Here is an example call of this function:

```
> M <- matrix (c(4,1,9,6,10,2,7,3,8), nrow=3, ncol=3)
> M
     [,1] [,2] [,3]
[1,]    4    6    7
[2,]    1   10    3
[3,]    9    2    8
> sum_edges (M)
[1] 40
```

   The value returned above is 40 because $40 = 4 + 6 + 7 + 3 + 8 + 2 + 9 + 1$. Your function should work when its argument is any numeric matrix, with at least two rows and columns, not just for the example shown above. The matrix will not necessarily have the same number of rows as columns.

7. Here is the definition of a function called `mystery3`:

```
mystery3 <- function (L) {
    M <- matrix (0, nrow=length(L), ncol=2)
    for (i in 1:length(L)) {
        M[i,1] <- length(L[[i]])
        M[i,2] <- mean(L[[i]])
    }
    M
}
```

a) For the two calls of this function below, write down the value that R will print:

```
> testlist <- list (1:3, c(8,0,0,7,0), c(9,1))
> mystery3 (testlist)
```

```
> x <- 0:10
> y <- 0:4
> mystery3 (list(x,y))
```

b) Below, write down a new definition for `mystery3` that computes the same results as the function defined above, but which is defined by a short R expression that uses `sapply` and `cbind`, and which does not contain a loop.

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

```
> bigger <- function (x) UseMethod("bigger")
>
> bigger.abc <- function (x) x+1
> bigger.pqr <- function (x) 2*x
> bigger.xyz <- function (x) x^2
>
> bigger_and_bigger <- function (z,n) {
+     r <- numeric(n)
+     for (i in 1:n) {
+         z <- bigger(z)
+         r[i] <- z
+     }
+     r
+ }
>
> u <- 2
> class(u) <- "abc"
> v <- 2
> class(v) <- "pqr"
> w <- 2
> class(w) <- "xyz"
>
> bigger_and_bigger (u,3)



> bigger_and_bigger (v,3)



> bigger_and_bigger (w,3)
```

9. Write down a definition for a function called `NA_for_bad_age_education` that takes a data frame of information on people as its only argument, and returns a data frame that is like its argument except that inconsistent values in the data frame for the variables `age` (age in years) and `education` (years of formal educuation) are replaced by `NA`, R's missing data indicator.

   In detail, you should replace the value of both `age` and `education` by `NA` if `education` is greater than `age` minus 4 (since it is implausible that someone would start formal education before age 4).

   You may assume without checking that variables (columns) called `age` and `education` exist in the data frame passed as the argument to `NA_for_bad_age_education`, and that there are no existing `NA` values for these variables.

   Here is an example of what this function should do:

```
> df
   name age gender education
1  mary  23      F        16
2   joe  14      M        13
3  fred  92      M        19
4  bert  34      M        42
5 holly  12      F         7
6  gwen  17      F         0
>
> NA_for_bad_age_education(df)
   name age gender education
1  mary  23      F        16
2   joe  NA      M        NA
3  fred  92      M        19
4  bert  NA      M        NA
5 holly  12      F         7
6  gwen  17      F         0
```

   This is just an example, however — your function should work for any data frame of the sort described, not just the one shown above.

10. In the five 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 at the starts of lines are the R command prompts, not something that was typed in.)

```
> set.seed(123)
>
> runif(4)
[1] 0.2875775 0.7883051 0.4089769 0.8830174
> runif(4)
[1] 0.9404673 0.0455565 0.5281055 0.8924190
>
> set.seed(123)
>
> a <- runif(4)
> b <- runif(4)
> any(a>b)



> all(a>b)



> all(4*a>b)



> set.seed(123)
>
> a-runif(4)



> a[a>runif(4)]
```

11. Write down a definition of a function called `is_and_is_not` that takes as its only argument a vector of character strings, and returns a vector of two numbers, the first of which is the number of occurrences of the string `"is"` in its argument, and the second of which is the number of these occurences of `"is"` that are followed immediately by `"not"`.

    Here is an example of what this function should do:

```
> is_and_is_not (c("if","he","is","at","home",
+                  "he","is","not","at","work",
+                  "and","he","is","not","at","school"))
[1] 3 2
```

    This is just an example, however — your function should work when passed any vector of strings (but you don't have to check that the argment actually is a vector of strings).