Stats 102A - Homework 1 - Output File

Miles Chen (example)

To receive full credit the functions you write must pass all tests. We may conduct further tests that are not included on this page as well.

Academic Integrity Statement

By including this statement, I, **Yuer Tang**, declare that all of the work in this assignment is my own original work.

At no time did I use an AI tool to generate or debug my solutions.

At no time did I look at the code of other students nor did I search for code solutions.

I understand that forbidden AI usage or plagiarism on any single part of this assignment will result in a 0 for the entire assignment.

```
source("102A_hw_01.R")  # edit with your file name
```

Part 1: Tests for `by_type()`

Do not edit any of the code in this section. The code here will run test cases to test if your function is working properly.

```
# Test Case 1: Standard mixed input with sorting OFF
x1 <- c("house", "6", "2.2", "a", "3.4", "1")
result1 <- by_type(x1)
print(result1)
```

```
$integers
[1] 6 1

$doubles
[1] 2.2 3.4

$character
[1] "house" "a"
```

```r
# Test Case 2: Standard mixed input with sorting ON
result2 <- by_type(x1, sort = TRUE)
print(result2)
```

```
$integers
[1] 1 6

$doubles
[1] 2.2 3.4

$character
[1] "a"     "house"
```

```r
# Test Case 3: Input with only integers
x3 <- c("10", "5", "3", "1")
result3 <- by_type(x3)
print(result3)
```

```
$integers
[1] 10  5  3  1

$doubles
numeric(0)

$character
character(0)
```

```r
# Test Case 4: Input with only doubles
x4 <- c("3.14", "2.71", "0.99")
result4 <- by_type(x4)
print(result4)
```

```
$integers
integer(0)

$doubles
[1] 3.14 2.71 0.99

$character
character(0)
```

```r
# Test Case 5: Input with only characters
x5 <- c("cat", "dog", "elephant")
result5 <- by_type(x5)
print(result5)
```

```
$integers
integer(0)

$doubles
numeric(0)

$character
[1] "cat"      "dog"      "elephant"
```

```r
# Test Case 6: Mixed input with NA values
x6 <- c("10", "house", NA, "3.2", "2")
result6 <- by_type(x6, sort = TRUE)
print(result6)
```

```
$integers
[1]  2 10

$doubles
[1] 3.2

$character
[1] "house" NA
```

```r
# Test Case 7: Input with logical values (should be treated as character)
x7 <- c(TRUE, FALSE, "7", "apple", "4.5")
result7 <- by_type(x7)
print(result7)
```

```
$integers
[1] 7

$doubles
[1] 4.5

$character
[1] "TRUE"  "FALSE" "apple"
```

```r
# Test Case 8: Input with floating-point numbers that can be integers
x8 <- c("3.0", "5.0", "7.5", "4")
result8 <- by_type(x8)
print(result8)
```

```
$integers
[1] 3 5 4

$doubles
[1] 7.5

$character
character(0)
```

```r
# Test Case 9: Empty vector input
x9 <- c()
result9 <- by_type(x9)
print(result9)
```

```
$integers
integer(0)

$doubles
numeric(0)

$character
character(0)
```

```r
# Test Case 10: Input with non-numeric special characters
x10 <- c("!", "@", "#", "5", "2.3")
result10 <- by_type(x10)
print(result10)
```

```
$integers
[1] 5

$doubles
[1] 2.3

$character
[1] "!" "@" "#"
```

Part 2: Tests for `prime_factor()`

This section will print out a bunch of test cases to see if your `prime_factor()` function is working properly. Do not edit the following code which runs tests.

```r
# a helper function for testing if a value x is prime.
is_prime <- function(x){
  if(!is.numeric(x)){
    stop("Input must be numeric")
  }
  if(length(x) != 1){
    stop("Input must have length of 1")
  }
  if(as.integer(x) != x){ # non-integer values are not prime
    return(FALSE)
  }
  if(x <= 1){ # negative numbers, 0, and 1 are all non-prime
    return(FALSE)
  }
  if(x == 2){ # special case for 2
    return(TRUE)
  }
  # we only need to check for prime factors less than sqrt(x)
  max <- ceiling(sqrt(x))
  for(i in 2:max){
    if(x %% i == 0){ # if the values divides evenly, it is not prime
      return(FALSE)
    }
  }
  TRUE
}
```

```
prime_factor(0)
```

```
Error in prime_factor(0): Input must be greater than or equal to 2.
```

```
prime_factor(1)
```

```
Error in prime_factor(1): Input must be greater than or equal to 2.
```

```
prime_factor(2.2)
```

```
Error in prime_factor(2.2): Input must be an integer with no decimal values.
```

```
for(x in c(2:50, 2001:2050)){
  factors <- prime_factor(x)
  cat(x, "has prime factors:", factors, "\n")
  if(!all(sapply(factors, is_prime))){
    cat("Problem: not all factors found are prime.\n")
  }
  if(prod(factors) != x){
    cat("Problem: the product of factors does not equal x.\n")
  }
}
```

```
2 has prime factors: 2
3 has prime factors: 3
4 has prime factors: 2 2
5 has prime factors: 5
6 has prime factors: 2 3
7 has prime factors: 7
8 has prime factors: 2 2 2
9 has prime factors: 3 3
10 has prime factors: 2 5
11 has prime factors: 11
12 has prime factors: 2 2 3
13 has prime factors: 13
14 has prime factors: 2 7
15 has prime factors: 3 5
16 has prime factors: 2 2 2 2
17 has prime factors: 17
```

```
18 has prime factors: 2 3 3
19 has prime factors: 19
20 has prime factors: 2 2 5
21 has prime factors: 3 7
22 has prime factors: 2 11
23 has prime factors: 23
24 has prime factors: 2 2 2 3
25 has prime factors: 5 5
26 has prime factors: 2 13
27 has prime factors: 3 3 3
28 has prime factors: 2 2 7
29 has prime factors: 29
30 has prime factors: 2 3 5
31 has prime factors: 31
32 has prime factors: 2 2 2 2 2
33 has prime factors: 3 11
34 has prime factors: 2 17
35 has prime factors: 5 7
36 has prime factors: 2 2 3 3
37 has prime factors: 37
38 has prime factors: 2 19
39 has prime factors: 3 13
40 has prime factors: 2 2 2 5
41 has prime factors: 41
42 has prime factors: 2 3 7
43 has prime factors: 43
44 has prime factors: 2 2 11
45 has prime factors: 3 3 5
46 has prime factors: 2 23
47 has prime factors: 47
48 has prime factors: 2 2 2 2 3
49 has prime factors: 7 7
50 has prime factors: 2 5 5
2001 has prime factors: 3 23 29
2002 has prime factors: 2 7 11 13
2003 has prime factors: 2003
2004 has prime factors: 2 2 3 167
2005 has prime factors: 5 401
2006 has prime factors: 2 17 59
2007 has prime factors: 3 3 223
2008 has prime factors: 2 2 2 251
2009 has prime factors: 7 7 41
2010 has prime factors: 2 3 5 67
```

```
2011 has prime factors: 2011
2012 has prime factors: 2 2 503
2013 has prime factors: 3 11 61
2014 has prime factors: 2 19 53
2015 has prime factors: 5 13 31
2016 has prime factors: 2 2 2 2 2 3 3 7
2017 has prime factors: 2017
2018 has prime factors: 2 1009
2019 has prime factors: 3 673
2020 has prime factors: 2 2 5 101
2021 has prime factors: 43 47
2022 has prime factors: 2 3 337
2023 has prime factors: 7 17 17
2024 has prime factors: 2 2 2 11 23
2025 has prime factors: 3 3 3 3 5 5
2026 has prime factors: 2 1013
2027 has prime factors: 2027
2028 has prime factors: 2 2 3 13 13
2029 has prime factors: 2029
2030 has prime factors: 2 5 7 29
2031 has prime factors: 3 677
2032 has prime factors: 2 2 2 2 127
2033 has prime factors: 19 107
2034 has prime factors: 2 3 3 113
2035 has prime factors: 5 11 37
2036 has prime factors: 2 2 509
2037 has prime factors: 3 7 97
2038 has prime factors: 2 1019
2039 has prime factors: 2039
2040 has prime factors: 2 2 2 3 5 17
2041 has prime factors: 13 157
2042 has prime factors: 2 1021
2043 has prime factors: 3 3 227
2044 has prime factors: 2 2 7 73
2045 has prime factors: 5 409
2046 has prime factors: 2 3 11 31
2047 has prime factors: 23 89
2048 has prime factors: 2 2 2 2 2 2 2 2 2 2 2
2049 has prime factors: 3 683
2050 has prime factors: 2 5 5 41
```

Part 3: Tests for `month_convert()`

Do not edit the following code which runs tests.

```
month_names <- read.delim("month_names.txt", encoding="UTF-8", row.names=1)

x <- factor(c("March", "March", "February", "June"))
month_convert(x, "English", "Spanish")
```

```
[1] marzo   marzo   febrero junio
Levels: febrero junio marzo
```

```
x <- factor(c("March", "March", "February", "June", "Jaly", "Hamburger", "December"))
month_convert(x, "English", "German")
```

```
[1] März     März     Februar  Juni     <NA>     <NA>     Dezember
Levels: Dezember Februar Juni März
```

```
x <- factor(c("gennaio", "febbraio", "marzo", "aprile", "maggio", "giugno", "luglio",
              "agosto", "settembre", "ottobre", "novembre", "dicembre"))
month_convert(x, "Italian", "English")
```

```
 [1] January   February  March     April     May       June      July
 [8] August    September October   November  December
12 Levels: August April December February January June July May ... September
```

```
x <- factor(c("janeiro", "março", "abril", "maio", "junho", "julho", "maio", "setembro",
              "outubro", "novembro", "dezembro", "setembro", "setembro", "março"))
y <- month_convert(x, "Portuguese", "French")
print(y)
```

```
 [1] janvier   mars      avril     mai       juin      juillet   mai
 [8] septembre octobre   novembre  décembre  septembre septembre mars
10 Levels: avril décembre janvier juillet juin mai mars novembre ... septembre
```

```
y <- month_convert(y, "French", "Danish")
print(y)
```

```
 [1] januar    marts     april     maj       juni      juli      maj
 [8] september oktober   november  december  september september marts
10 Levels: april december januar juli juni maj marts november ... september
```

```
y <- month_convert(y, "Danish", "Dutch")
print(y)
```

```
 [1] januari   maart     april     mei       juni      juli      mei
 [8] september oktober   november  december  september september maart
10 Levels: april december januari juli juni mei maart november ... september
```

```
y <- month_convert(y, "Dutch", "Icelandic")
print(y)
```

```
 [1] janúar    mars      apríl     maí       júní      júlí      maí
 [8] september október   nóvember  desember  september september mars
10 Levels: apríl desember janúar júlí júní maí mars nóvember ... september
```

Part 4: Questions to Answer

Replace 'write your answer here' with your responses. Be sure your answers have been 'highlighted' using the triple hash `###` which makes the text large and bold.

1. Coercion: For each of the following, explain what type of output you will receive and why R is producing that output.

    a. `c(0, TRUE)`
    b. `c("F", F)`
    c. `c(list(1), "b")`
    d. `c(FALSE, 1L)`

a. will be of type double because 0 is type double and it will coerece the TRUE to type double.

b. will be of type character because "F" is type character and it will coerce the False to type character.

c. will be of type list becasue list(1) is type list and it will coerce the "b" to type list.

d. will be of type integer becasue 1L is type integer and it will coerce the False to type integer.

2. What is the difference between NULL, NA, and NaN?

NA is used to represent missing or unknown values. There are NA for each type. NULL is used to represent an empty or nonexistent value. NULL is its own type. NaN is type double and is used to represent indeterminate forms in mathematics (such as 0/0 or -Inf + Inf).

3. What is the difference between logical(0) and NULL? Write a command (other than `logical(0)`) that will produce logical(0) as the output. Write a command (other than `NULL`) that will produce NULL as the output.

logical(0) is an empty logical vector. It has a type (logical) and a length (0). It is a valid vector. It can be combined with other vectors. On the other hand NULL means no object at all. It has no type and no length.

```
# results in logical(0)
TRUE[0]
```

```
logical(0)
```

```
# results in NULL
print(cat())
```

```
NULL
```

4. A vector `c(TRUE, FALSE)` is a logical vector. Other than `TRUE` or `FALSE`, what can you insert into the vector so that it increases to a length of 3 and remains a logical vector and does not get coerced into another class?

insert NA.

```
length(c(TRUE, FALSE, NA))
```

```
[1] 3
```

5. What are the lengths of the following lists? Use bracket notation to subset them to the equivalent of `c("h","i")`. Be sure to print the result so it shows the subset. ### l1 has length 3 because it contains three list elements. l2 has length 2 because it contains two list elements.

```
l1 <- list(letters[1:5], letters[3:9] , letters[4:7])
l1
```

```
[[1]]
[1] "a" "b" "c" "d" "e"

[[2]]
[1] "c" "d" "e" "f" "g" "h" "i"

[[3]]
[1] "d" "e" "f" "g"
```

```
l2 <- list( c(letters[1:5], letters[3:9]), letters[4:7] )
l2
```

```
[[1]]
 [1] "a" "b" "c" "d" "e" "c" "d" "e" "f" "g" "h" "i"

[[2]]
[1] "d" "e" "f" "g"
```

```
l1[[2]][6:7]
```

```
[1] "h" "i"
```

```
l2[[1]][11:12]
```

```
[1] "h" "i"
```

6. What will `c(4:7) * c(2:4)` produce? Briefly, why?

c(4:7) is (4,5,6,7) while c(2:4) is (2,3,4) then it will be: (4*2*, 5*3*, 6*4*, 7*2*) = (8, 15, 24, 14). The rule is R multiplies vectors element-wise and applies the recycling rule, so the shorter vector is repeated to match the length of the longer one

7. Take a look at the following code chunks. What are some of the differences between `cat()` and `print()`?

```
cat(5 + 6)
```

```
11
```

```
print(5 + 6)
```

```
[1] 11
```

```
x8 <- cat(5 + 6)
```

```
11
```

```
y8 <- print(5 + 6)
```

```
[1] 11
```

```
x8
```

```
NULL
```

```
y8
```

```
[1] 11
```

```
cat(letters[1:3], letters[24:26])
```

```
a b c x y z
```

```
print(letters[1:3], letters[24:26]) # Why are we getting the following error?
```

```
Warning in print.default(letters[1:3], letters[24:26]): NAs introduced by
coercion
```

```
Error in print.default(letters[1:3], letters[24:26]): invalid printing digits -2147483648
```

```
# Error in print.default(letters[1:3], letters[24:26]) : invalid 'digits' argument
cat(l1)
```

```
Error in cat(l1): argument 1 (type 'list') cannot be handled by 'cat'
```

```
print(l1)
```

```
[[1]]
[1] "a" "b" "c" "d" "e"

[[2]]
[1] "c" "d" "e" "f" "g" "h" "i"

[[3]]
[1] "d" "e" "f" "g"
```

print() will display and return an object and also show the structure of that objects. cat() just concatenates and displays objectes but returns NULL. cat() can handle multiple vectors but can't handle list. print() can handle list but can only expect one single object.

8. What happens to a factor when you reverse its levels?

```
f1 <- factor(c("A","A","B","C","D","A","C"))
f1
```

```
[1] A A B C D A C
Levels: A B C D
```

```r
levels(f1) <- rev(levels(f1))
f1
```

```
[1] D D C B A D B
Levels: D C B A
```

When you reverse the levels, the observation stay unchanged. Only the level labels are reordered so the printed values looks reserved but the actual data observation stay unchanged.

9. How do f2 and f3 differ from the unmodified f1?

```r
f1 <- factor(c("A","A","B","C","D","A","C"))
f1
```

```
[1] A A B C D A C
Levels: A B C D
```

```r
f2 <- factor(rev(c("A","A","B","C","D","A","C")))
f2
```

```
[1] C A D C B A A
Levels: A B C D
```

```r
f3 <- factor(c("A","A","B","C","D","A","C"), levels = rev(c("A","B","C","D")))
f3
```

```
[1] A A B C D A C
Levels: D C B A
```

f2 differs from f1 because the order of observation is reversed but the factor level remains in default alphebetical order. On the other hadn, f3 differs from f1 because the levels are set to be reversed has reversed vector but the levels will not reverse even the vector reverse. f3 has reversed level because it sets level to be reversed.

10. What attributes does a data frame possess?

It has names, class, and row.names

```
df <- data.frame(
  x = c(1, 2, 3),
  y = c("a", "b", "c")
)
attributes(df)
```

```
$names
[1] "x" "y"

$class
[1] "data.frame"

$row.names
[1] 1 2 3
```

11. What does `as.matrix()` do when applied to a data frame with columns of different types? Create a simple `data.frame` with two columns: one numeric and one string. Use `as.matrix` and show the results.

All columnes will be coerced to a single type. It will the coercion order: logical → integer → numeric → character.

```
df <- data.frame(
  x = c(1, 2, 3),
  y = c("a", "b", "c")
)

df
```

```
  x y
1 1 a
2 2 b
3 3 c
```

```
as.matrix(df)
```

```
     x   y
[1,] "1" "a"
[2,] "2" "b"
[3,] "3" "c"
```