Data Structures and Advanced Programming, Spring 2018 Programming Assignment 1 (bonus)

Instructor: Ling-Chieh Kung Department of Information Management National Taiwan University

To submit your work, please upload a CPP file for Problems 1 to PDOGS at http://pdogs.ntu.im/judge/. Each student must submit her/his individual work. No hard copy. No late submission. The due time of this homework is 1:00 PM, April 2.

The total points you may get for the program correctness part of Programming Assignment 1 is calculated in the following way. Let x be your grades of that part in the original Programming Assignment 1 and y be those of this bonus problem. Your grades for the program correctness part of Programming Assignment 1 is then $\min\{x+y,60\}$.

Problem 1

(30 points) One very popular tool for statistical analysis is the R programming language. In R, one may very easily create vectors and matrices. For example, to create a vector x = (1, 2, 3), the instruction is $x \leftarrow c(1, 2, 3)$ (the function c() will concatenate multiple inputs into one single vector literal). To later modify x to (5, 4, 3, 2, 1), the instruction is nothing but $x \leftarrow c(5, 4, 3, 2, 1)$. If there exists another vector y = (5, 0, 1, 1), one may also create a new vector z by executing $z \leftarrow y$.

If you think about these instructions, you quickly realize that each vector in R is just like a C++ object, and that assignment operator <- is overloaded for R vectors so that the right-hand-side (RHS) value of <- can be either a vector literal (like c(1, 2, 3)) or a vector object (like y). Besides the assignment operator, the indexing operator is also overloaded. We may use z[2] to access the second element of z, whose value is 0. Note that the indices of R vectors fit out common sense; you do not need to worry about the cumbersome i or i-1 issue in C++.

Most arithmetic operators have also been overloaded. R defines some vector arithmetic operations in an interesting way: recycling (which is the word R uses to mean repeating). Consider the addition operation as an example. Let $x_1 = (1, 2, 3, 4, 5)$ and $x_2 = (8, 4, 2, 1, 0)$, naturally $x_1 + x_2 = (9, 6, 5, 5, 5)$. What if we have $x_3 = (9, 1, 5)$ and we want to calculate $x_1 + x_3$? In this case, R calculate the outcome in two steps:

- 1. Recycle x_3 (because it is shorter) into x_3' to make x_3' as long as x_1 . The elements in x_3 would be repeated and appended to x_3 until x_3' is long enough. In this example, x_3' would be (9, 1, 5, 9, 1).
- 2. Add x_1 and x'_3 , the recycled x_3 , to get (10, 3, 8, 13, 6).

Therefore, the outcome of $x_1 + x_3$ is defined to be (10,3,8,13,6). For subtraction, it is the same. For example, if $y_1 = (9,9,9,9,9,9,9,9,9,9)$ and $y_2 = (1,2,4,0)$, we have $y_1 - y_2 = (8,7,5,9,8,7,5,9,8)$. For comparison operators like less-than, greater-than, or equal-to, the same recycling rule is still applied: The shorter vector is recycled to be as long as the longer one, and then a natural pairwise comparison gives the result. For example, $z_1 = (1,2,-1,4,1) < z_2 = (2,3)$ is false and $z_3 = (1,2,-1,0,1) < z_2$ is true. For this problem, a < b (respectively, a > b) is defined as all elements in a is strictly smaller (respectively, larger) than the corresponding element in b.

In this problem, you will be implementing these vector arithmetic with C++ operator overloading. You will need to create a class RecylcingVector with overloaded operators =, +, -, [], <, >, and ==. The codes you need to write will be quite similar with the codes on the slides. In fact, you are welcome to modify the class MyVector given by the instructor to complete this problem.

¹http://www.r-project.org/.

Input/output formats

There are 30 input files. In each file, there are several lines. Each line start with a single English letter, which may be N, A, C, I, S, and M. These letters corresponds to the following events:

- If the first letter is "N" (for "new"), it will be followed by an English lowercase letter c, a positive integer n, and then n integers $x_1, x_2, ...,$ and x_n . This will create a new vector $c = (x_1, x_2, ..., x_n)$. Note that the character c is the name of the vector, and thus there will be at most 26 vectors when you run your program.
- If the first letter is "A" (for "assignment"), it will be followed by an English lowercase letter c, a positive integer n, and then n integers $x_1, x_2, ...,$ and x_n . This will assign $(x_1, x_2, ..., x_n)$ to the vector c (and remove all the original values in c) even if n is different from the current dimension of c. You may assume that c has been created with an appropriate N event.
- If the first letter is "C" (for "comparison"), it will be followed by a character x that is L (for "less than"), G (for "greater than"), or E (for "equal to"), an English lowercase letter c_1 , and another English lowercase letter c_2 . This will print out the result of $c_1 < c_2$, $c_1 > c_2$, and $c_1 = c_2$, respectively. If it is true, please print out 1; otherwise, please print out 0. Each output number should occupy a single line.
- If the first letter is "I" (for "indexing"), it will be followed by an English lowercase letter c, an integer i, and an integer v. This will assign v to c_i . You may assume that $1 \le i \le n$.
- If the first letter is "S" (for "sum"), it will be followed by a positive integer m and m English lowercase letters c_1 , c_2 , ..., and c_m . This will assign $c_2 + c_3 + \cdots + c_m$ to c_1 . You may assume that c_i has been created with an appropriate N event for all i = 2, ..., m. However, c_1 may not be an existing vector. In this case, you need to create a new vector whose value is $c_2 + c_3 + \cdots + c_m$.
- If the first letter is "M" (for "minus"), it will be followed by a positive integer m and m English lowercase letters c_1 , c_2 , ..., and c_m . This will assign $c_2 c_3 \cdots c_m$ to c_1 . You may assume that c_i has been created with an appropriate N event for all i = 2, ..., m. However, c_1 may not be an existing vector. In this case, you need to create a new vector whose value is $c_2 c_3 \cdots c_m$.

In each line, two consecutive values are separated by a white space. You may assume that $n \leq 20$, $m \leq 10$, and all values that may be assigned into a vector can be stored in an int variable.

For example, consider a file containing

```
N a 3 0 0 0
N b 5 1 2 1 2 5
A a 8 3 4 3 4 3 4 5 6
N c 2 -1 -2
S 4 a a b c
I a 1 0
C L a b
```

as the input data. The first two lines creates two vectors a = (0,0,0) and b = (1,2,1,2,5). Then the third line modifies a into a = (3,4,3,4,3,4,5,6). After the fourth line creates c = (-1,-2), the fifth line calculates a + b + c and then modify a into a + b + c (in other words, this lines adds b + c into a). As

$$a+b+c = (3,4,3,4,3,4,5,6) + (1,2,1,2,5,1,2,1) + (-1,-2,-1,-2,-1,-2,-1,-2) \\ = (3,4,3,4,7,3,6,5),$$

the vector a now becomes (3,4,3,4,7,3,6,5). The sixth line then modifies a_1 to 0; a thus becomes (0,4,3,4,7,3,6,5). Finally, we check whether a < b. As the second element-wise comparison yields $4 \nleq 2$, the result is false. We should print out 0 as the outcome.

Please note that in each input file, there may be multiple C events. Each output number should occupy a single line. You are considered making the right answer only if you are right for all the C events.

What should be in your source file

You are required to implement a class RecyclingVector by completing the operator overloading tasks given in this problem. Then in your main function, you should use the class to complete the given task. If you need any member of RecyclingVector that is not defined above, feel free to add it. However, your RecyclingVector must have at least the members defined in this problem.

Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. For this problem, you may use any technique you prefer. You are suggested to write relevant comments for your codes.

Grading criteria

30 points for this program will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. For each set of input data, if your program outputs correctly without violating the space limit, you get 1 point. The 30 testing files are organized in the following way:

- In the first ten files, there are only N, A, and C events.
- In the eleventh to the twentieth files, there are also I events.
- For the last ten files, there are all kinds of events.