CS 101 **Project 1** **Summer 2017**

Write a C++ program project1.cpp that behaves as described below. Your program will read from standard input and write to standard output.

First read integer N which denotes the number of dimensions of a multi-dimensional array. Next read the dimensions d_0 d_1 … d_{N-1}, so the array will have dimensions d_0 by d_1 by … by d_{N-1}. Next read the initial values of the array using array aggregate notation. The array elements will always include the value 0 and will always consist of distinct consecutive non-negative integers, but these integers might be arranged in any order. Example:

2 3 4 {{4,1,2,7},{11,8,3,9},{5,10,0,6}}	$\begin{bmatrix} 4 & 1 & 2 & 7 \\ 11 & 8 & 3 & 9 \\ 5 & 10 & 0 & 6 \end{bmatrix}$

Next read a sequence of moves until end-of-input occurs. Each move relocates the 0 element to a new position, and is specified as a dimension and a distance. The dimension k will be in the range 0 to N–1, and the distance indicates how far the 0 element will travel (positive indicates increasing indexes, and negative indicates decreasing indexes). However, to prevent reaching outside the range of valid indexes 0 to d_k–1, the indexes along each dimension will always wrap around in a circular fashion. As the value 0 travels the specified distance, it swaps places with each other value it encounters. The output contains the sequence of all array configurations, both initially and after each move is complete, displayed using array aggregate notation. Example:

2 3 4 {{4,1,2,7},{11,8,3,9},{5,10,0,6}} 1 1 0 -1 1 1 0 -1 1 1 0 -1 1 2	<u>Output:</u> 0: {{4,1,2,7},{11,8,3,9},{5,10,0,6}} 1: {{4,1,2,7},{11,8,3,9},{5,10,6,0}} 2: {{4,1,2,7},{11,8,3,0},{5,10,6,9}} 3: {{4,1,2,7},{0,8,3,11},{5,10,6,9}} 4: {{0,1,2,7},{4,8,3,11},{5,10,6,9}} 5: {{1,0,2,7},{4,8,3,11},{5,10,6,9}} 6: {{1,10,2,7},{4,8,3,11},{5,0,6,9}} 7: {{1,10,2,7},{4,8,3,11},{5,6,9,0}}

The goal is to reach an array configuration in which all the array elements are displayed in ascending consecutive order. Whenever this occurs, display the message "Success after XX moves", as shown in the next three examples.

2 2 2 {{0,1},{2,3}} 0 1 1 1 1 1 0 1	0: {{0,1},{2,3}} Success after 0 moves 1: {{2,1},{0,3}} 2: {{2,1},{3,0}} 3: {{2,1},{0,3}} 4: {{0,1},{2,3}} Success after 4 moves

0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1	5: {{2,1},{0,3}} 6: {{2,1},{3,0}} 7: {{2,0},{3,1}} 8: {{0,2},{3,1}} 9: {{3,2},{0,1}} 10: {{3,2},{1,0}} 11: {{3,0},{1,2}} 12: {{0,3},{1,2}} 13: {{1,3},{0,2}} 14: {{1,3},{2,0}} 15: {{1,0},{2,3}} 16: {{0,1},{2,3}} Success after 16 moves

2 4 5 {{5,1,2,3,4},{10,6,7,8,9},{11,12,13,18,14},{15,0,16,17,19}} 1 2 0 -1 1 -3 0 -2
0: {{5,1,2,3,4},{10,6,7,8,9},{11,12,13,18,14},{15,0,16,17,19}} 1: {{5,1,2,3,4},{10,6,7,8,9},{11,12,13,18,14},{15,16,17,0,19}} 2: {{5,1,2,3,4},{10,6,7,8,9},{11,12,13,0,14},{15,16,17,18,19}} 3: {{5,1,2,3,4},{10,6,7,8,9},{0,11,12,13,14},{15,16,17,18,19}} 4: {{0,1,2,3,4},{5,6,7,8,9},{10,11,12,13,14},{15,16,17,18,19}} Success after 4 moves

3 3 3 3 {{{0,4,2},{3,7,5},{6,25,8}},{{9,10,11},{12,13,14},{15,16,17}},{{1,19,20},{18,22,23},{21,24,26}}} 0 -1 1 2 2 1 0 1 1 -2 2 -1
0: {{{0,4,2},{3,7,5},{6,25,8}},{{9,10,11},{12,13,14},{15,16,17}},{{1,19,20},{18,22,23},{21,24,26}}} 1: {{{1,4,2},{3,7,5},{6,25,8}},{{9,10,11},{12,13,14},{15,16,17}},{{0,19,20},{18,22,23},{21,24,26}}} 2: {{{1,4,2},{3,7,5},{6,25,8}},{{9,10,11},{12,13,14},{15,16,17}},{{18,19,20},{21,22,23},{0,24,26}}} 3: {{{1,4,2},{3,7,5},{6,25,8}},{{9,10,11},{12,13,14},{15,16,17}},{{18,19,20},{21,22,23},{24,0,26}}} 4: {{{1,4,2},{3,7,5},{6,0,8}},{{9,10,11},{12,13,14},{15,16,17}},{{18,19,20},{21,22,23},{24,25,26}}} 5: {{{1,0,2},{3,4,5},{6,7,8}},{{9,10,11},{12,13,14},{15,16,17}},{{18,19,20},{21,22,23},{24,25,26}}} 6: {{{0,1,2},{3,4,5},{6,7,8}},{{9,10,11},{12,13,14},{15,16,17}},{{18,19,20},{21,22,23},{24,25,26}}} Success after 6 moves

Your program should behave as illustrated in the above examples. The program should be runnable by typing either of the following commands:

./a.out
./a.out < *inputfile* > *outputfile*

For the first line above, your program will read from the keyboard and write to the screen. The second line redirects standard input to come from the specified *inputfile*, and it redirects standard output to go to the specified *outputfile*, but your program is unaware of these files.

One last requirement: Because the output can become lengthy, we also need a way to condense the output but still be able to verify whether your program is working correctly. Therefore, if any command-line argument is present, then only display the output lines that begin with the word "Success". Example: If you run the program by typing one of these commands

./a.out x
./a.out 1 < *inputfile* > *outputfile*

Then your program should behave as follows:

2 2 2 {{0,1},{2,3}} 0 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1	Success after 0 moves Success after 4 moves Success after 16 moves

To earn full credit, your program should work correctly when the array has at most 3 dimensions. To earn extra credit, your program should work correctly for arbitrary number of dimensions.

Motivation: With 2 dimensions this problem is similar to the well-known 8-puzzle or 15-puzzle. With 3 dimensions this problem is similar to Rubik's cube. Note that your program is not actually creating a new solution, rather it is just examining the effects caused by a given sequence of moves.

Please carefully read the following requirements:

- You must do your own work. You must not borrow any code from any other person, book, website, or any other source. You also must not share your code with any other person, or post it on any website. We run plagiarism detection software on every project. So if you violate these rules, you may receive an invitation to the dean's office to discuss the penalties for academic misconduct.
- Because a purpose of this course is to learn how to properly implement data structures and algorithms, it is not permitted in general to include STL libraries such as <list>, <vector>, <stack>, <queue>, <deque>, <algorithm>, <numeric>, <utility>, … in your program. You are always permitted to use <iostream>, <string>, and any C libraries such as <cstdlib> and <cmath>. But do not include any other libraries unless explicitly allowed on the assignment.
- Make sure your program runs properly on the cs-intro.ua.edu server, because this is where your program will be graded. In particular, make sure your program initializes the values of all variables when they are declared or allocated. Otherwise it might behave differently on Linux than it does on a PC or Mac.
- Your program will be compiled using this command: g++ project1.cpp –Wall –lm –std=c++11. Alternatively, if you split your program into multiple files, then it will instead be compiled using this command: g++ *.cpp –Wall –lm –std=c++11.
- Verify that all the necessary .h and .cpp files which are needed to compile your program are included in the same directory before you submit your program. There should be no extra subdirectories and no extra .h or .cpp files, otherwise your program might not compile the way you intended.
- Compress your project into a zip file that contains your C++ program source file. Right-click (or secondary click) on your project directory, and then (depending on your operating system) select either the Compress option or Send To → Compress from the popup menu. Finally upload your .zip file that contains your .cpp file for this project to Blackboard.
- If you violate the above requirements such that it breaks our grading script, your project will be assessed a significant point deduction, and extreme or multiple violations may cause the project to be considered ungradable.
- Every semester many students lose some points because they don't follow all the instructions. So please read and follow all the project specifications precisely to prevent losing points unnecessarily. If anything is unclear, please ask for clarification well before the project is due. Please pay particular attention to input and output formats.
- Submit your project on Blackboard by the due date (11:59pm Friday). There is a grace period of 24 hours (until 11:59pm Saturday). Projects submitted on Sunday will be assessed a late penalty of 5% per hour. No projects will be accepted after Sunday. Once it is graded, your project score will be posted on Blackboard and the results of the grading script will be sent to your Crimson email account.
- Double-check and triple-check your submission when you submit it. Errors discovered later cannot be fixed and resubmitted after the project is graded. Projects will not be re-graded unless an error is found in the grading script or in the input/output files that are used during grading.