COMP2119A Introduction to Data Structures and Algorithms
Programming Assignment 1 **Due Date:** 7pm, Oct 23, 2018

Rules: discussion of the problems is permitted, but writing the assignment together is not (i.e. you are not allowed to see the actual solution of another student).

Course Outcomes

- **[O4]. Implementation**

[O4] In this assignment, you are requested write a program to solve four tasks, **A**, **B**, **C** and **D**, using ADT we learned during class.

Your program will be judged by a computer automatically, please follow the exact format. You should read from standard input and write to standard output. e.g, scanf/print, cin/cout.

Languages.

We only accept C/C++ programming languages.

Judging.

Please note that your solution is automatically judged by a computer.

Solutions that fail to compile or execute get 0 points.

We have designed 40 test cases, 10 for each task. Each test case is of 2.5 points.

The time limit for each test case is 1 second.

For each test case, you will get 2.5 points if your program passes it otherwise you will get 0.

Self Testing.

You should test your program by yourself using the provided sample input/output file.

The sample input/output is **different** from the ones used to judge your program, and it is designed for you to test your program by your own.

Note that your program should always use standard input/output.

To test your program in Windows:
1. Compile your program, and get the executable file, "main.exe"
2. Put sample input file, "input.txt", in the same directory as "main.exe"
3. Open command line and go to the directory that contains "main.exe" and "input.txt"
4. Run "main.exe < input.txt > myoutput.txt"
5. Compare myoutput.txt with sample output file. You may find "fc.exe" tool useful.
6. Your output needs to be **exactly** the same as the sample output file.

To test your program in Linux or Mac OS X:
1. Put your source code "main.cpp" and sample input file "input.txt" in the same directory.
2. Open a terminal and go to that directory.
3. Compile your program by "g++ main.cpp -o main"

4. Run your program using the given input file, "./main < input.txt > myoutput.txt"
5. Compare myoutput.txt with sample output file.
6. Your output needs to be **exactly** the same as the sample output file.
7. You may find the **diff** command useful to help you compare two files. Like "diff -w myOutput.txt sampleOutput.txt". The $-w$ option ignores all blanks (SPACE and TAB characters)

Note that myoutput.txt file should be **exactly** the same as sample output. Otherwise it will be judged as wrong.

Sketch file. We provide a source file as a sketch for task **A**. You write your code based on that. You can also write your own code as long as the format is correct.

Submission.

Please submit your source files through moodle.

You should submit four source files(**without** compression), one for each task.

Please name your files in format UniversityNumber-X.cpp for X in {A, B, C, D}, e.g. 1234554321-A.cpp.

Task A. Recall the last question appeared in HW1.

Consider the ADT stack. Given a sequence of operations on a stack, we can easily calculate the sequence of pushed values and the sequence of popped values.

For example, for the sequence "push 1; push 2; pop; push 3; pop; pop", its sequence of pushed values is "1 2 3" and its sequence of popped values is "2 3 1".

Now you are given two sequences of length n: $pushseq = (a_1, a_2, \cdots, a_n)$ and $popseq = (b_1, b_2, \cdots, b_n)$. We also guarantee that the a_i's are distinct, e.g., for $i \neq j$, $a_i \neq a_j$. You are required to determine whether there exists a sequence of stack operations with push sequence *pushseq* and pop sequence *popseq*. If the answer is "yes", you program is also required to **output a sequence of stack operations** that gives the sequences. Your program needs to run in $O(n)$ time.

For example, for $pushseq = (1, 2, 3)$ and $popseq = (2, 3, 1)$, the answer is "yes". You program needs to output a sequence of such stack operations: "push 1 push 2 pop push 3 pop pop". For $pushseq = (1, 2, 3)$ and $popseq = (3, 1, 2)$, no such sequences exist.

You are required to write a **C++** program which solves this problem.

Input.

The input contains multiple lines.

The first line contains a char 'A', which means the coming input is for Task A.

The second line contains an integer n, $1 \leq n \leq 10^6$. The third line contains n distinct numbers corresponding to the $pushseq = (a_1, \ldots a_n)$.

The fourth line contains n distinct numbers corresponding to the $popseq = (b_1, \ldots b_n)$.

$1 \leq a_i, b_i \leq n$

Output. Output "NO" if no such valid operations. Otherwise output each operation in new line.

Examples.

1. Input:

```
A
3
1 2 3
3 2 1
```

   Output:

```
push 1
push 2
push 3
pop
pop
```

```
pop
```

2. Input:

   ```
   A
   3
   1 2 3
   3 1 2
   ```

   Output:

   ```
   NO
   ```

Task B. People are standing in a circle waiting to be executed. Counting begins at a specified point in the circle and proceeds around the circle in a specified direction. After a specified number of people are skipped, the next person is executed. The procedure is repeated with the remaining people, starting with the next person, going in the same direction and skipping the same number of people, until only one person remains, and is freed.

Initially, each person is randomly assigned an integer l representing the number of people are skipped, where $-10 \leq l \leq 10$. The minus sign means reversed direction.

Input.

The input contains multiple lines.

The first line contains a char 'B', which means the coming input is for Task B.

The second line contains an integer n, $1 \leq n \leq 10^6$. The third line contains n distinct integers corresponding to the id of each person, each id lies in range $[1, n]$. The fourth line contains l's corresponding to the number of people skipped.

Output.

Print a single integer in a line, which is the id of the freed person.

Examples.

1. Input:

   ```
   B
   3
   1 2 3
   3 2 1
   ```

   Output:

   ```
   3
   ```

   Explanation:

   Starting from person 1 with skipping number 3, 1 is exectued after the first round. Starting from person 2 with skipping number 2, 2 is exectued after the second round. Finally, 3 is survived.

2. Input:

   ```
   B
   5
   5 2 1 3 4
   2 1 3 -4 9
   ```

   Output:

2

Explanation:

round	circle	starting	skipping number	being killed
1	(5 2 1 3 4)	5	2	1
2	(5 2 3 4)	3	-4	3
3	(5 2 4)	4	9	4
4	(5 2)	5	2	5
5	2			

Notice: In round 2, after 3 is exectuted, the next starting person is 4, not 2.

Task C.

Output all permutations for n distinct integers in lexicographically increasing order.

$a_1 a_2 ... a_n$ is lexicographically less than $b_1 b_2 ... b_n$ if $a_i < b_i$ for the first i where a_i and b_i differ.

Input.

The input contains multiple lines.

The first line contains a char 'C', which means the coming input is for Task C.

The second line contains an integer n, $1 \leq n \leq 8$.

The third line contains n distinct integers a_i in increasing order, where $0 \leq a_i \leq 10^9$.

Output.

Output all permutations in lexicographically increasing order.

1. Input:
   ```
   C
   1
   1
   ```
   Output:
   ```
   1
   ```

2. Input:
   ```
   C
   3
   1 2 3
   ```
   Output:
   ```
   1 2 3
   1 3 2
   2 1 3
   2 3 1
   3 1 2
   3 2 1
   ```

Task D.

Given an integer n and 1 <= i <= $n!$, return the i-th permutation in the above lexicographically increasing ordering for [n]={1, 2, ..., n}.

Input.

The input contains multiple lines.

The first line contains a char 'D', which means the coming input is for Task D.

The second line contains two integers n and i, $1 \leq n \leq 20$ and $1 \leq i \leq n!$.

Output.

Output the i-th permutation in a single line.

Examples.

1. Input:
   C
   1 1

   Output:

   1

2. Input:
   C
   3 3

   Output:

   2 1 3

   Explanation: The 3rd of all permutations of {1, 2, 3} is $2, 1, 3$.

OPTIONAL

Solve task D for $n \leq 1000$. This task is optional and will not be judged or tested.

The below are hints for some tasks.

Task B

[Hint]: Use doubly linked list.

Task D

[Notice]: The **int** type using 32-bit in C/C++ lies in range $[-2147483648, 2147483647]$, wherever $20! = 2.432902 * 10^{18}$.

Task E

[Hint]: Implement your own integer type which supports big integer's operations, such as '+' and '-'.