# 2022_DS_Fall_Exercises 1

# Exercise #1

The Fibonacci numbers are defined as:

- $f_0 = 0$
- $f_1 = 1$, and
- $f_i = f_{i-1} + f_{i-2}$ for $i > 1$

Write both a recursive and iterative C function to compute $f_i$ .

## Input Format

The first line of input contains a number $n$. It represents the amount of questions.

After the first-line, the following will have $n$ line of text. Each line contains a string $S$ and a number $i$, separate by a whitespace character. The $S$ can be `recursive` or `iterative`, which stands for which Fibonacci algorithm your program is going to execute. The number $i$ will be the subscript of the Fibonacci number $f_i$.

## Output Format

Output $n$ line of text. Each line contain the corresponding $f_i$ value.

## Technical Specification

- $1 \leq n \leq 10$
- If $S$ is `recursive`, then $0 \leq i \leq 10$
- If $S$ is `iterative`, then $0 \leq i \leq 40$

## Sample Input

```
6
recursive 0
iterative 1
recursive 2
iterative 3
recursive 4
iterative 5
```

## Sample Output

```
0
1
1
2
3
5
```

# Exercise #2

There are three towers and $n$ disks of different diameters placed on the first tower. The disks are in order of decreasing diameter as one scans up the tower. Monks were reputedly supposed to move the disk from tower 1 to tower 3 obeying the rules:

- Only one disk can be moved at any time.
- No disk can be placed on top of a disk with a smaller diameter.

Write a recursive function that prints out the sequence of moves needed to accomplish this task.

## Input format

A line of input contains a number $n$. It represents the amount of disks which you placed on the first tower.

## Output format

Output the sequence of moves needed to accomplish your tower.

- First column shows the number of disk.
- Second column shows the disk that you determined in first column where it is in which tower.
- Third column shows the destination of the disk you moved.

## Technical Specification

$1 \leq n \leq 20$

## Sample input

```
3
```

```
5
```

## Sample output

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

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

```
4 1 2
1 3 2
2 3 1
1 2 1
3 3 2
1 1 3
2 1 2
1 3 2
5 1 3
1 2 1
2 2 3
1 1 3
3 2 1
1 3 2
2 3 1
1 2 1
4 2 3
1 1 3
2 1 2
1 3 2
3 1 3
1 2 1
2 2 3
1 1 3
```

# Exercise #3

Rewrite `fastTranspose` so that it uses only one array rather than the two arrays required to hold `rowTerms` and `startingPos`.

## Intput Format

Input consist of $T + 1$ line.

First-line contain three number $R, C, T$. The matrix size will be $R \times C$. $T$ is the number of enties in the matrix.

Next $T$ lines, each line contains three numbers $r_t$, $c_t$, $v_t$. Represent the row-index, column-index and element-value.

The input matrix entries are sorted by $r$ and $c$ in ascending order.

## Output Format

Output $T + 1$ line.

First-line contains the $R, C, T$ related to the transposed matrix.

Following $T$ line, each contains three number $r$, $c$, $v$.

The output matrix entries are sorted by $r$ and $c$ in ascending order.

## Technical Specification

- $0 < T < 100$
- $0 < R, C < 1000000$

- $1 \leq r \leq R$
- $1 \leq c \leq C$
- $0 \leq v \leq 1000000$

## Sample Input

```
6 6 8
0 0 15
0 3 22
0 5 -15
1 1 11
1 2 3
2 3 -6
4 0 91
5 2 28
```

## Sample Output

```
6 6 8
0 0 15
0 4 91
1 1 11
2 1 3
2 5 28
3 0 22
3 2 -6
5 0 -15
```

# Exercise #4

There are a number of problems, known collectly as "random walk" problems, that have been of longstanding interest to the mathematical community. All but the most simple of these are extremely diffcult to solve, and, for the most part, they remain largely unsolved. One such problem may be stated as:

A (drunken) cockroach is placed on a given square in the middle of a tile floor in a rectangular room of size $n \times m$ tiles. The bug wanders (possibly in search of an move from his present tile to any of the eight tiles surrounding him (unless he is against a wall) with equal porbability, how long will it take him to touch every tile on the floor at least once?

Hard as this problem may be to solve by pure probability techniques, it is quite easy to solve using a computer. The technique for doing so is called "simulation". This technique is widely used in industry to predict traffic flow, inventory control, and so forth. The problem may be simulated using the following method:

An $n \times m$ array count is used to represent the number of times our cockroach has reach each tile on the floor. All the cells of this array are initialized to zero. The position of the bug on the floor is represented by the coordinates (ibug, jbug). The eight possible moves of the bug are represented by the tiles located at $(\mathrm{ibug} + \mathrm{imove}[k]), \mathrm{jbug} + \mathrm{jmove}[k])$, where $0 \leq k \leq 7$, and

$$
\begin{aligned}
\text{inmove}[0] &= -1 \quad \text{jmove}[0] = \ \ 1 \\
\text{inmove}[1] &= \ \ 0 \quad \text{jmove}[1] = \ \ 1 \\
\text{inmove}[2] &= \ \ 1 \quad \text{jmove}[2] = \ \ 1 \\
\text{inmove}[3] &= \ \ 1 \quad \text{jmove}[3] = \ \ 0 \\
\text{inmove}[4] &= \ \ 1 \quad \text{jmove}[4] = -1 \\
\text{inmove}[5] &= \ \ 0 \quad \text{jmove}[5] = -1 \\
\text{inmove}[6] &= -1 \quad \text{jmove}[6] = -1 \\
\text{inmove}[7] &= -1 \quad \text{jmove}[7] = \ \ 0
\end{aligned}
$$

A random walk to any one of the eight neighbor squares is simulated by generating a random value for $k$, lying between $0$ and $7$. Of course, the bug cannot move outside the room, so that coordinates that lead up a wall must be ignored, and a square is incremented so that a nonzero entry shows the number of times the bug has landed on that square. When every square has been entered at least once, the experiment is complete.

Write a program to perform the specified simulation experiment. Your program MUST:

(a). handle all values of $n$ and $m$, $2 < n \leq 40, 2 \leq m \leq 20$;
(b). perform the experiment for (1) $n = 15, m = 15$, starting point (10, 10), and (2) $n = 39, m = 19$, starting point $(1, 1)$.
(c). have an iteration limit, that is, a maximum number of squares that the bug may enter during the experiment. This ensures that your program will terminate. A maximum of $50,000$ is appropriate for this exercise.

For each experiment, print (1) total number of legal moves that the cockroach makes and (2) the final count array. This will show the "density" of the walk, that is, the number of times each tile on the floor was touched during the experiment.

This exercise was contributed by Olson.

# Input Format

The input consist of multiple lines. Each line have any one the following format.

- A line consists of one character 'a' and two integer numbers $n$, $m$ which $2 < n \leq 40$, $2 \leq m \leq 20$. Represent problem (a).
- A line consists of one character 'b' and one integer number $s$ which is $1$ or $2$. Represent problem (b)(1) and (b)(2).
- A line consists of one character 'q'. (Quit the program.)

# Output Format

For each problem:

- A line consists of one integer which is the total number of steps;
- following with $n$ lines, each consist of $m$ numbers. Each lines represents the row of cells. They are the numbers of visit of that cell.

# Sample Input

```
a 2 2
a 2 3
a 3 2
b 1
b 2
q
```

## Sample Output

The output will vary from run to run. We only test the following conditions.

1. All cells walked.
2. The sum of steps match the sum of all number in the cells.

```
15
   3   1
   6   5
11
   1   2   1
   3   3   1
28
   1   1
   6   4
   9   7
2703
  12  24  18  11   3   6   6   7   6   5   2   3   5   7   1
  19  25  20  15  17  10  13  15   5   6   6   6   7   8   5
  18  19  19  13  13  15  10  10   7   4   7   6   9   9   6
  22  31  11   8   8   5  10   8   8   8   4   6   5   9   4
  21  29  21  14  13   3   5  12   6  10   5   3   9   5   2
  19  37  31  18  11   8   8   3   9   7   6  12   3   4   4
  18  21  23  18  15   4  13   9   9  14   9   8   3   4   4
  20  23  19  12   5   8  10  11   7   9   9   6  12   6   1
  13  17  12   9   4  12   6   9   5   8   7   8   6   6   4
  12  19  18  12   8  10   8   3   4   5   6   9   7  12  10
  13  19  15  10  10  13  14   8   5   4   4  17  21  22  19
  13  28  21  18  14  18  20  11  10   8  10  23  18  27  22
   7  14  14  19  18  17  20  17  14  12  27  25  22  17  24
   5   7  14  21  22  21   8  13  11  19  42  24  20  24  25
   3   9  16  13  10   7   4   3   6  18  26  17   6  14  11
11582
  11  27  14   6   1   2   5   4   5   6   2   4   3   3   4   5   4   5   1
  19  18  23  16   9   6   7  10   6   6   6   8  11   7   7   5   9   4   2
  11  14  13  13  13   5   2   6   8   5   3   6   7   9   6   7   9  11   4
  20  18  12  13   7   3   6   5   3   3   3   5  11   9  15  19   9  17   9
  16  21  14   6  11   8   9   8   5   3   5   7   8  12  23  22  18  14   8
  13  19  11   9   7   6   9  11   5   7   6   7  13  22  27  15  16  13  12
   8  16  11   9   9   9   6  13   7   5   4   7  12  21  14  23  23  16  12
  11  13  12  15  12  11   5  10   8   8   5   8   7  17  26  14  22  18  18
   5  11  14  10  12  11   5  16   8   6   7  10  14  10  16  27  21  19  17
   4  14  11  14  13  13  13  12   8   7  14  11  13  18  19  23  21  26  16
  11  15  17  18  21  11  13  13  13   7  13  15  18  17  14  19  16  19  14
  13  15  12  16  10  12  17  17  12  10  14  18  11  16  14  14  16  23  19
   9  13  15  15  17  11  13  15  11  14  15  23  12  12  14  15  20  32  18
```

```
  3  12   9  13  17  16   9   8  11  15  19  11  14  15  14  16  34  27  17
  7  12  11   9  16  14   7   5   6  23  12  10  23  20  18  21  25  31  12
  8  13  12  17  13  16  12   8  14  17  14  20  20  17  14  23  20  22  18
  8  12  16   9  12   9  10  10  11  16  14  17  12  20  20  22  19  16  13
 14  26  15  20  13  10   9   7  16  10  14  15  15  23  21  13  20  20  17
 18  24  24  17  10   9   9  10   8  11  10  16  12  14   9  15  22  17  16
 15  20  13   7  11  11   7   6   7  10  10  12   9   8  15  20  22  18   8
 11  15  17  12  12   8   8   7   3  14  18   9   7  15  21  23  20  17  15
  9  11  14   9  17  15   8  11   7   7   9  14  14  12  11  32  30  27  21
  8  12  14  17  15  19  13   5   9   7  10  11  12  16  21  38  49  35  21
 10   8  17  22  17  18  14   8  12   4   6   5  15  25  29  47  41  35  17
  6  14  11  14  14  12  15  18   5   4   8   5  15  24  21  37  27  22  12
  7  17  14  11  12  13  18  10  13   6  10  15   7  21  25  19  26  21   9
 10  13  21   9  12  12  16  18  18  16  16  17  12  16  26  22  23  16  10
 14  25  19  13  18  17  10  10  23  26  23  17  18  25  25  16  20  19   6
 14  25  26  18  22  23  15  16  22  17  23  17  18  15  17  16  12  19  11
 18  25  27  28  23  20  21  29  21  20  20  20  18  22  13   7  14  18  10
 24  29  21  18  17  22  14  22  23  18  27  26  20  16  13  18  13  10  10
 30  40  20  16  19  19  15  13  22  32  31  29  23  17  21  15  18  13   9
 32  38  28  27  23  19  25  14  20  24  27  37  24  21  20  16  18  17   9
 24  36  29  29  20  20  19  17  17  21  19  28  19  15  18  18  21  14   9
 20  31  27  31  20  14  19  20  23  20  20  17  18  14  19  19  22  24  14
 14  28  35  28  14  13  13  23  29  25  31  23  18  21  17  24  23  30  14
 16  29  26  22  15  12  10  13  37  34  30  33  27  26  31  28  35  28  20
 10  23  18  13   7  15  18  25  31  24  30  24  21  33  41  37  36  38  14
  6  11   9   4   4   5  14  14  12  18  13  11  14  24  37  24  15  25  14
```

# Exercise #5

Using the information provided in the text, write a complete program to search a maze. Print out the entrance to exit path if successful.

## Input Format

First-line contain two number $H$ and $W$. Represent the height and width of the input maze.

After first-line there will be $H$ lines of input. Each line contain $W$ numbers $w_0, w_1, \ldots w_W$.

All numbers are separated by a whitespace.

The value of number $w_x$ might be $0$ or $1$.

- $0$ represent a road in the maze.
- $1$ represent a wall in the maze.

## Output Format

The entry point of the maze will be the most northwest point at $(0, 0)$.
Your goal is reach the most southeast point at $(H - 1, W - 1)$.

Output the path from the entry to the exit, each line contain two number $h$ and $w$. Represent the coordinate of current position.

If there is no such path, output `no answer`.

## Technical Specification

- $3 \leq H \leq 50$
- $3 \leq W \leq 50$
- You can assume there is only one way or no way to walk from the entry to the exit.

## Sample Input

```
3 5
0 1 1 0 1
0 1 0 1 0
1 0 1 1 0
```

```
3 3
0 1 1
1 1 1
1 1 0
```

## Sample Output

```
0 0
1 0
2 1
1 2
0 3
1 4
2 4
```

```
no answer
```

# Exercise #6

Design and build a linked allocation system to represent and manipulate polynomials. You should use circularly linked lists with header nodes. Each term of the polynomial will be represented as a node, using the following structure:

| coef | expon | link |
|---|---|---|

In order to erase polynomials efficiently, use the available space list and associated functions discussed in this seciton.

Write and test the following functions:

- `pread`. Read in a polynomial and convert it to its circular representation. Return a pointer to the header node of this polynomial.
- `pwrite`. Output the polynomial using a form that clearly displays it.
- `padd`. Compute $c = a + b$. Do not change either $a$ or $b$.
- `psub`. Compute $c = a - b$. Do not change either $a$ or $b$.
- `pmult`. Compute $c = a \times b$. Do not change either $a$ or $b$.
- `eval`. Evaluate a polynomial at some point, $a$ where $a$ is a floating point constant. Return the result as a floating point.
- `perase`. Return the polynomial represented as a circular list to the available space list.

# Input Format

Input consist of $n + 1$ line.

First-line contain one number $n$. Represent how many queries in following line.

Following $n$ line will be one of the following format.

- pread $[\text{expression name}][ec]$ $c_0$ $e_0$ $c_1$ $e_1$ $\dots$ $c_{ec}$ $e_{ec}$
  - $[\text{expression name}]$ a string. Represent the name for the expression.
  - $[\text{ec}]$ a number.  how many terms in the polynomial expression.
  - $c_x$ the x'th term's coefficient.
  - $e_x$ the x'th term's exponent.
- pwrite $[\text{expression name}]$
  - $[\text{expression name}]$ a string. Represent the name for the expression.
- padd $[\text{result name}]$ $[\text{left expression name}]$ $[\text{right expression name}]$
  - $[\text{result name}]$ a string. the name for $c$ of $c = a + b$
  - $[\text{left expression name}]$ a string. Represent the expression name of $a$ of $c = a + b$
  - $[\text{right expression name}]$ a string. Represent the expression name of $b$ of $c = a + b$
- psub $[\text{result name}]$ $[\text{left expression name}]$ $[\text{right expression name}]$
  - $[\text{result name}]$ a string. the name for $c$ of $c = a - b$
  - $[\text{left expression name}]$ a string. Represent the expression name of $a$ of $c = a - b$
  - $[\text{right expression name}]$ a string. Represent the expression name of $b$ of $c = a - b$
- pmult $[\text{result name}]$ $[\text{left expression name}]$ $[\text{right expression name}]$
  - $[\text{result name}]$ a string. the name for $c$ of $c = a \times b$
  - $[\text{left expression name}]$ a string. Represent the expression name of $a$ of $c = a \times b$
  - $[\text{right expression name}]$ a string. Represent the expression name of $b$ of $c = a \times b$
- eval $[\text{result name}]$ $[a]$
  - $[\text{result name}]$ a string. Represent the expression name to evaluate.
  - a a floating point number.
- perase $[\text{expression name}]$
  - $[\text{expression name}]$ a string. Represent the expression name to erase.

# Output Format

- pread $[\text{expression name}][ec]$ $c_0$ $e_0$ $c_1$ $e_1$ $\dots$ $c_{ec}$ $e_{ec}$
  - output `ok`
- pwrite $[\text{expression name}]$
  - output the polynomial string.
  - output terms in exponent descending order.
- padd $[\text{result name}]$ $[\text{left expression name}]$ $[\text{right expression name}]$
  - output `added`
- psub $[\text{result name}]$ $[\text{left expression name}]$ $[\text{right expression name}]$
  - output `subtracted`
- pmult $[\text{result name}]$ $[\text{left expression name}]$ $[\text{right expression name}]$
  - output `multiplied`
- eval $[\text{result name}]$ $[a]$

    - output the evaluate the expression with $x = a$
    - $a$ can be a floating number.
    - round down the result to the 2nd digit ($floor(0.4567) = 0.45$)
- perase $[\text{expression name}]$
    - output `erased`

## Technical Specification

- length of string $|\text{expression name}| < 50$
- $0 < ec < 10$
- $-100 < c_0, c_1, \ldots, c_{ec} < 100$
- $-100 < e_0, e_1, \ldots, e_{ec} < 100$

## Sample Input

```
13
pread expressionA 2 2 2 1 1
pread expressionB 2 3 2 2 1
pwrite expressionA
pwrite expressionB
padd resultC expressionA expressionB
pwrite resultC
perase resultC
psub resultC expressionA expressionB
pwrite resultC
perase resultC
pmult resultC expressionA expressionB
pwrite resultC
eval resultC 2.5
```

## Sample Output

```
ok
ok
2x^2+1x^1
3x^2+2x^1
added
5x^2+3x^1
erased
subtracted
-1x^2-1x^1
erased
multiplied
6x^4+7x^3+2x^2
356.25
```

$$\text{resultC}(x) = 6x^4 + 7x^3 + 2x^2$$
$$\text{resultC}(2.5) = 356.25$$

# Exercise #7

Write the following C functions.

- [read] accept a tree represented as a parenthesized list as input and create the generalized list representation of the tree.
- [copy] make a copy of a tree represented as a generalized list.
- [isequal] test for equality between two trees represented as generalized lists.
- [clear] delete a tree represented as a generalized list.
- [write] output a tree in its parenthesized list notation.

# Input Format

First-line of input contain one number $n$. It represent the amount of commands.

After first-line, there will be $n$ line, each line consist of one command.

Each command having  their own format.

- $\text{read}\ [\text{tree id}]\ [\text{generalized-list}]$
  - `tree id` is a number, the name for given tree.
  - `generalized-list` is a string. Represent a tree in generalized-list form.
- $\text{copy}\ [\text{src tree id}]\ [\text{destination tree id}]$
  - `src tree id` is a number. The tree to copy.
  - `destination treeid` is another number. The new id for copied tree.
- $\text{isequal}\ [\text{tree id 1}]\ [\text{tree id 2}]$
  - `tree id 1` is a number. Represent the first tree to compare with.
  - `tree id 2` is a number. Represent the second tree to compare with.
- $\text{clear}\ [\text{tree id}]$
  - `tree id` is a number. Represent the tree to clear
- $\text{write}\ [\text{tree id}]$
  - `write id` is a number. Represent the tree to output.

# Output Format

For each received command, output the following corresponding info.

- `read`, output `readed`
- `copy`, output `copied`
- `isequal`, output `true` or `false` based on comparing result.
- `clear`, output `cleared`
- `write`, output the corresponding tree in generalized list notation.

# Technical Specification

- $n < 50$
- For each `tree id` number $i$, we have $i < 100$.
- For each `generalized-list` string $S$, we have $|S| < 100$.
  - The node data will be an uppercase alphabet, possible value range from $\text{A}$ to $\text{Z}$.
- You can assume there is no operation will lead whole program to illegal state. For example: copy/write/test equal to/delete a non-exists generalized list, or read a malformed generalized list.

- The generalized list describe tree node values in a unordered manner. To define a strict output order result, the output node should be order by their alphabet order.
    - For example: since $\text{char 'B'} < \text{char 'C'}$ in alphabet order, we should output `(A(B,C))` instead of `(A(C,B))`. Given `(A(C(X),B(Z)))`, we should output `(A(B(Z),C(X)))`.

## Sample Input

```
12
read 0 (A(B,C))
read 1 (A(C,B))
isequal 0 1
write 0
write 1
copy 0 2
write 2
clear 0
clear 1
clear 2
read 0 (A(B(E(L,K),F),C(G),D(H(M),I,J)))
write 0
```

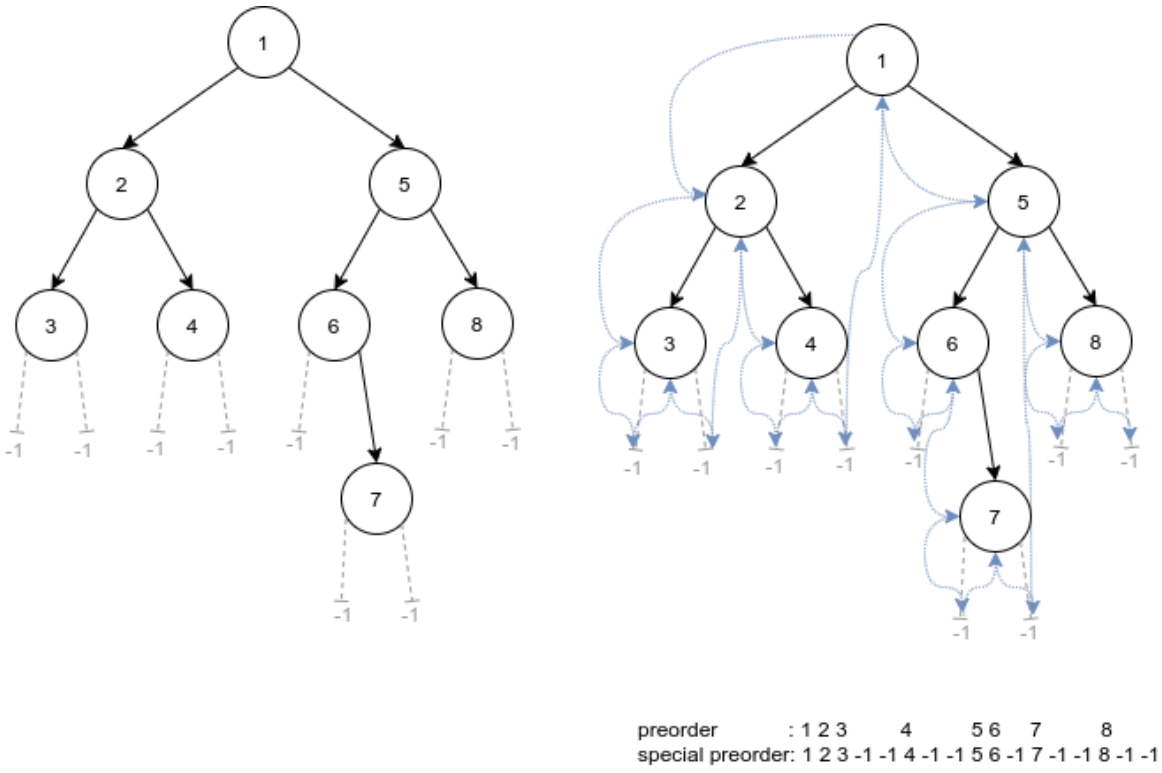
## Sample Output

```
readed
readed
true
(A(B,C))
(A(B,C))
copied
(A(B,C))
cleared
cleared
cleared
readed
(A(B(E(K,L),F),C(G),D(H(M),I,J)))
```

# Exercise #8

1. Write a nonrecursive version of funciton `preorder`
2. Write a nonrecursive version of function `postorder`

## Input Format




The input contains two lines.

1. First-line contains one number $n$. It represents how many number in the next line.
2. Second-line contains $n$ numbers $i_0, i_1, \ldots, i_n$, each separated by a whitespace. These numbers describe a binary tree with a **special preorder traversals**.

> Usually, during the preorder traversal of a binary tree, we ignore **null** child reference. But with this **special preorder traversals**, we output **null** child reference as $-1$. With this information, you should be able to generate only one kind of binary tree.

# Output Format

The output contains two line.

1. First-line contains a series of number. It represents the preorder of input binary tree.
2. Second-line contains a series of number. It represents the postorder of input binary tree.

# Technical Specification

- $0 < n < 10^3$
- $0 < i_0, i_1, i_2, \ldots, i_n < 10^3$

# Sample Input

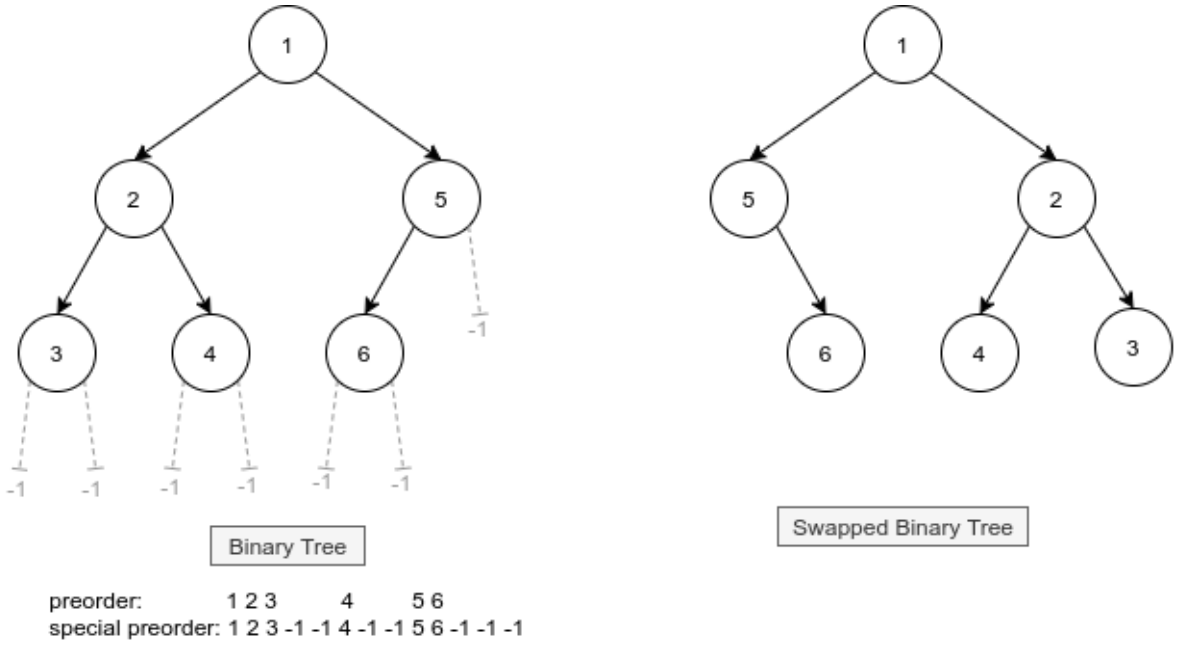
```
17
1 2 3 -1 -1 4 -1 -1 5 6 -1 7 -1 -1 8 -1 -1
```

# Sample Output

```
1 2 3 4 5 6 7 8
3 4 2 7 6 8 5 1
```

# Exercise #9

Write a C function `swapTree` that takes a binary tree and swaps the left and right children of every node. An example is given in following image.




## Input Format

The input contains two lines.

1. First-line contains one number $n$. It represents how many number in the next line.
2. Second-line contains $n$ numbers $i_0, i_1, \ldots, i_n$, each separated by a whitespace. These numbers describe a binary tree with a **special preorder traversals**.

> Usually, during the preorder traversal of a binary tree, we ignore **null** child reference. But with this **special preorder traversals**, we output **null** child reference as $-1$. With this information, you should be able to generate only one kind of binary tree.

## Output Format

The output contains two line.

1. First-line contains a series of number. It represents the **preorder traversal** of swapped binary tree.
2. Second-line contains a series of number. It represents the **inorder traversal** of swapped binary tree.

## Technical Specification

- $0 < n < 10^3$
- $0 < i_0, i_1, i_2, \ldots, i_n < 10^3$

## Sample Input

```
13
1 2 3 -1 -1 4 -1 -1 5 6 -1 -1 -1
```

## Sample Output

```
1 5 6 2 4 3
5 6 1 4 2 3
```