Day 21 / 100 :

Topic - Stack, String

1 Problem statement: **Simplify Path** (Medium)

Given a string path, which is an absolute path (starting with a slash '/') to a file or directory in a Unix-style file system, convert it to the simplified canonical path.

In a Unix-style file system, a period '.' refers to the current directory, a double period '..' refers to the directory up a level, and any multiple consecutive slashes (i.e. '//') are treated as a single slash '/'. For this problem, any other format of periods such as '...' are treated as file/directory names.
The canonical path should have the following format:
The path starts with a single slash '/'.
Any two directories are separated by a single slash '/'.
The path does not end with a trailing '/'.
The path only contains the directories on the path from the root directory to the target file or directory (i.e., no period '.' or double period '..')
Return the simplified canonical path.

Example 1:
Input: path = "/home/"
Output: "/home"
Explanation: Note that there is no trailing slash after the last directory name.

Example 2:
Input: path = "/../"
Output: "/"
Explanation: Going one level up from the root directory is a no-op, as the root level is the highest level you can go.

Example 3:
Input: path = "/home//foo/"
Output: "/home/foo"
Explanation: In the canonical path, multiple consecutive slashes are replaced by a single one.

Solutions :
Approach 1 - Stack, string

To proceed with it, we will firsst of all declare 2 support variables:

- res is our accumulator variable, but, just to further optimise and squeeze even more efficiency, we will also use it to create temporary strings;
- tmp is a vector of strings we will use to store (or pop) our parsed strings as we go.

We will then parse the string character by character, other than the first one (that we know is always going to be '\', so no point in considering it).

Looping from i from 1 to the last character (lmt), we will:

- assign the value of path[i] to c;
- deal with c, depending if:
    - c != '/', we will append it to res;
    - in any other case and if i == lmt (ie: we are parsing the last character - and note that this might happen even if after we appended it to res above):
    - if res == ".." and we have at least one element in tmp, we will pop the last one from it;
    - if res is not empty and != ".", we will add it to tmp;
    - reset res to be "".

Once done, time to finally use res for its proper function: if tmp is not empty, we will set it to be "", "/" otherwise.

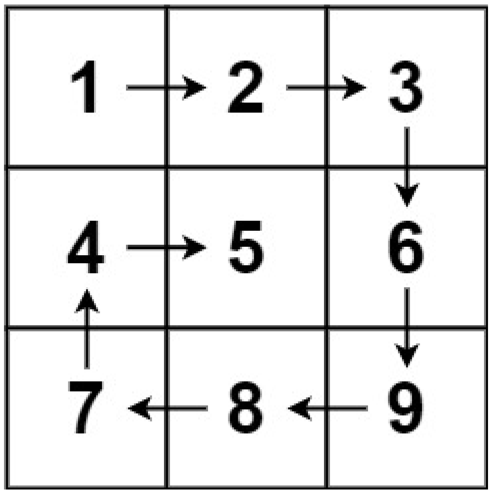
We will then loop through all the strings w collected in tmp and append "/" + w to res.

Once done, we can return it.

```
class Solution {
public:
    string simplifyPath(string path) {
        // support variables
        string res;
        vector<string> tmp;
        // parsing path
        // for (int i = i, lmt = path.size(); i < lmt; i++) {
            // if (path[i] != '/') res.append(path[i]);
        for (int i = 1, c, lmt = path.size() - 1; i <= lmt; i++) {
            c = path[i];
            // case 1: c is part of a folder name
            if (c != '/') res.append(1, c);
            // case 2: end of a folder name
            if (c == '/' || i == lmt) {
                // sub-case 2-1: parent directory
                if (res == "..") {
                    if (tmp.size()) tmp.pop_back();
                }
                // sub-case 2-2: res is valid and not the current directory
                else if (res.size() && res != ".") tmp.push_back(res);
                // resetting res
                res = "";
            }
        }
        // composing res
        res = tmp.size() ? "" : "/";
        for (string w: tmp) res += "/" + w;
        return res;
    }
};
```

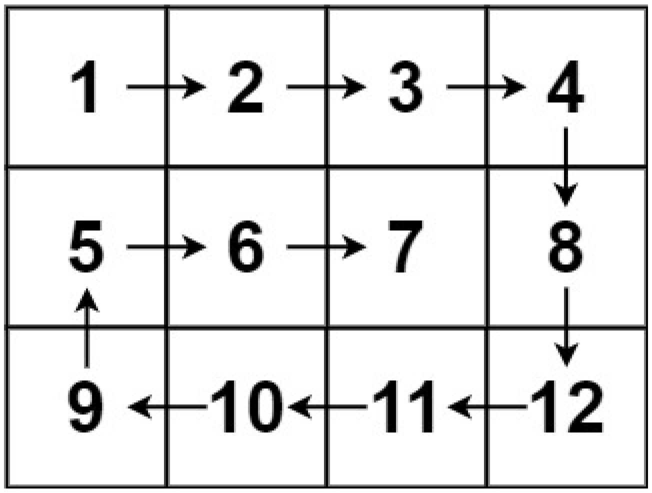
2 Problem statement: Spiral matrix (Medium)

Given an m x n matrix, return all elements of the matrix in spiral order.
Example 1:



Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [1,2,3,6,9,8,7,4,5]

Example 2:



Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]

Solutions :
Approach 1 - 2D array

When traversing the matrix in the spiral order, at any time we follow one out of the following four directions: RIGHT DOWN LEFT UP. Suppose we are working on a 5 x 3 matrix as such:

0 1 2 3 4 5
6 7 8 9 10
11 12 13 14 15

Imagine a cursor starts off at (0, -1), i.e. the position at '0', then we can achieve the spiral order by doing the following:

Go right 5 times
Go down 2 times
Go left 4 times
Go up 1 times.
Go right 3 times
Go down 0 times -> quit
Notice that the directions we choose always follow the order 'right->down->left->up', and for horizontal movements, the number of shifts follows:{5, 4, 3}, and vertical movements follows {2, 1, 0}.

Thus, we can make use of a direction matrix that records the offset for all directions, then an array of two elements that stores the number of shifts for horizontal and vertical movements, respectively. This way, we really just need one for loop instead of four.

Another good thing about this implementation is that: If later we decided to do spiral traversal on a different direction (e.g. Counterclockwise), then we only need to change the Direction matrix; the main loop does not need to be touched.

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
    vector<vector<int> > dirs{{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
    vector<int> res;
    int nr = matrix.size();     if (nr == 0) return res;
    int nc = matrix[0].size();  if (nc == 0) return res;

    vector<int> nSteps{nc, nr-1};

    int iDir = 0;   // index of direction.
    int ir = 0, ic = -1;    // initial position
    while (nSteps[iDir%2]) {
        for (int i = 0; i < nSteps[iDir%2]; ++i) {
            ir += dirs[iDir][0]; ic += dirs[iDir][1];
            res.push_back(matrix[ir][ic]);
        }
        nSteps[iDir%2]--;
        iDir = (iDir + 1) % 4;
    }
    return res;
}
```