Assignment - 01

1. Write a C program to scan two N x N matrices from the user. Perform addition of these matrices and store them in the third matrix. Use a single 3D dynamically generated array via pointer for the same.

2. Write a program to sort a stack using a temporary stack. Build a stack as input from the user.
   input: [34, 3, 31, 98, 92, 23]

```
Element taken out: 23
input: [34, 3, 31, 98, 92]
tmpStack: [23]
-----------------------------------------------
Element taken out: 92
input: [34, 3, 31, 98]
tmpStack: [23, 92]
-----------------------------------------------
Element taken out: 98
input: [34, 3, 31]
tmpStack: [23, 92, 98]
-----------------------------------------------
Element taken out: 31
input: [34, 3, 98, 92]
tmpStack: [23, 31]
-----------------------------------------------
Element taken out: 92
input: [34, 3, 98]
tmpStack: [23, 31, 92]
-----------------------------------------------
Element taken out: 98
input: [34, 3]
tmpStack: [23, 31, 92, 98]
-----------------------------------------------
Element taken out: 3
input: [34, 98, 92, 31, 23]
tmpStack: [3]
-----------------------------------------------
Element taken out: 23
input: [34, 98, 92, 31]
tmpStack: [3, 23]
-----------------------------------------------
Element taken out: 31
input: [34, 98, 92]
tmpStack: [3, 23, 31]
-----------------------------------------------
Element taken out: 92
input: [34, 98]
tmpStack: [3, 23, 31, 92]
-----------------------------------------------
Element taken out: 98
input: [34]
tmpStack: [3, 23, 31, 92, 98]
-----------------------------------------------
Element taken out: 34
input: [98, 92]
tmpStack: [3, 23, 31, 34]
-----------------------------------------------
Element taken out: 92
input: [98]
tmpStack: [3, 23, 31, 34, 92]
-----------------------------------------------
Element taken out: 98
input: []
tmpStack: [3, 23, 31, 34, 92, 98]
-----------------------------------------------
```

final sorted list: [3, 23, 31, 34, 92, 98]

3. Write a program to print the characters of the string in sorted order using stack.

   Input: str = "hello3569world12478"
   Output: 123456789dehlllloorw

   Algorithm:
   - Initialize two stacks, one main stack and other auxiliary stack.
   - Insert the first character of the string in the main stack.
   - Iterate for all the characters in the string one by one.
   - if the current character (nth character) is greater than or equal to the top element of the main stack, then push the element.
   - if the current character is not greater, then push all the greater elements of the main stack into the auxiliary stack, and then push the character into the main stack. After this, push all the greater elements of the auxiliary stack to the main stack.
   - Print all elements of the stack in reverse order when the iteration is completed.

4. Write a program to reverse a stack using recursion. (You are not allowed to use loop constructs like while, for..etc,). Use the function mentioned below.

   - `isEmpty(S)`
   - `push(S)`
   - `pop(S)`

5. Write a program to sort a stack using recursion (Use of any loop constructs like while, for etc. is not allowed.)

```
Output:
Stack elements before sorting:
-3   14   18   -5   30

Stack elements after sorting:
30   18   14   -3   -5
```

6. You are given a string S consisting of lowercase English letters. A duplicate removal consists of choosing two adjacent and equal letters and removing them. We repeatedly make duplicate removals on S until we no longer can.
   Write a program to return the final string after all such duplicate removals have been made.

```
Input: s = abbaca
Output: ca

    Explanation:
    For example, in "abbaca" we could remove "bb" since
    the letters are adjacent and equal, and this is the
    only possible move.  The result of this move is
    that the string is "aaca", of which only "aa" is
    possible, so the final string is "ca".
```

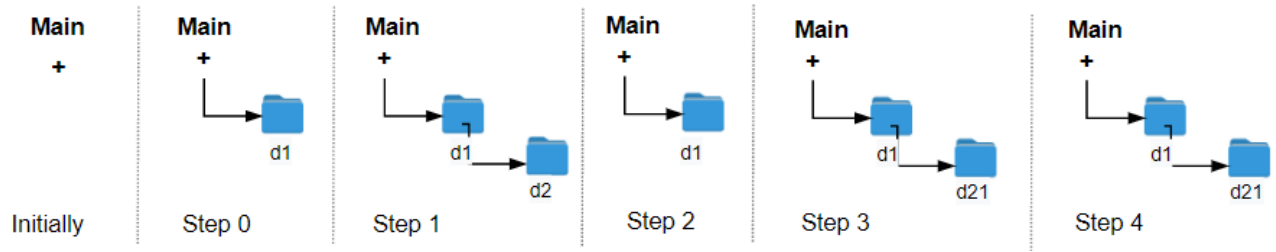
7. Write a C program to Implement Queue Using stack

8. Write a C program to create a queue where insertion and deletion of elements can be done from both the ends.

9. The file system crawler keeps a log each time some user performs a change folder operation.
   The operations are described below:
   - "../" : Move to the parent folder of the current folder. (If you are already in the main folder, remain in the same folder).
   - "./" : Remain in the same folder.
   - "x/" : Move to the child folder named x (This folder is guaranteed to always exist).

   You are given a list of string logs where logs[i] is the operation performed by the user at the i^{th} step. The file system starts in the main folder, then the operations in logs are performed. Return the minimum number of operations needed to go back to the main folder after the change folder operations.

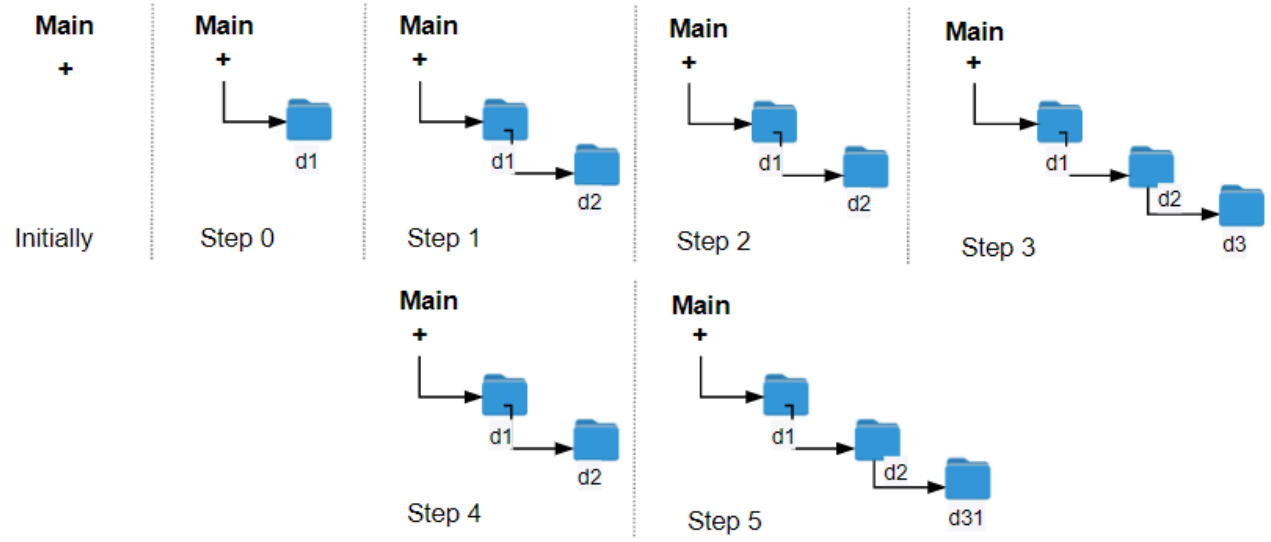
Example 1:




```
Input: logs = ["d1/","d2/","../","d21/","./"]
Output: 2

Explanation: Use this change folder operation "../" 2
times and go back to the main folder.
```

Example 2:




```
Input: logs = ["d1/","d2/","./","d3/","../","d31/"]
Output: 3
```

Example 3:

```
Input: logs = ["d1/","../","../","../"]
Output: 0
```

Constraints:

- `1 <= logs length <= 103`
- `2 <= logs[i]'s length <= 10`
- logs[i] contains lowercase English letters, digits, '.', and '/'.
- logs[i] follows the format described in the statement.
- Folder names consist of lowercase English letters and digits.
- You can also print the path at last.

10. Write a program to implement following stack operations using arrays with MAX elements. Use an Integer array. At the start of the program look for a command line argument for filename - if the file exists read file and load the stack and then start the program, else create a new stack and continue with the program.
    - · PUSH
    - · POP
    - · PEEP
    - · DISPLAY
    - · CHANGE
    - · IS_FULL
    - · IS_EMPTY
    - · SAVE IN FILE