Data Structures Fall 2023: Stacks and its Applications

Implement Templates too.

<u>Task One :</u>

Make a stack class that implements all the following functions,

1) Stack ()
   - Results: Constructor. Creates an empty stack.

2) ~Stack ()
   - Results: Destructor. Deallocates (frees) the memory used to store a stack.

3) void push (const DataItem)
   - Results: Push the element at the top of the stack.

4) Void pop ()
   - Results: Returns the element from the top of the stack.

5) element Peek ()
   - Results: Return the element at the top of the stack.

6) void clear ()
   - Results: Removes all the elements from a stack.

7) Bool isEmpty ()
   - Results: Returns true if a stack is empty. Otherwise, returns false.

<u>Task Two:</u>

Given string num representing a non-negative integer num, and an integer k, return the smallest possible integer after removing k digits from num. You have to remove the leading zeros if left any.

Example1:

Input: num = "1432219", k = 3

Output: "1219"

Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.

Example 2:

Input: num = "10200", k = 1

Output: "200"

Explanation: Remove the leading 1 and the number is 200. Note that the output must not contain leading zeroes.

Example 3:

Input: num = "10", k = 2

Output: "0"

Explanation: Remove all the digits from the number and it is left with nothing which is 0.

<u>Task Three:</u>

Given a string `path` representing an absolute path in a Unix-style file system, starting with a leading slash ('/'), the task is to transform it into a simplified canonical path.

In a Unix-style file system, a period ('.') denotes the current directory, a double period ('..') denotes the parent directory, and any sequence of consecutive slashes (e.g., '//') is treated as a single slash ('/'). In this context, any other occurrences of periods like '...' are interpreted as names of files or directories.

The desired canonical path should adhere to the following format:

1. It must begin with a single leading slash ('/').
2. Each pair of directories must be separated by a single slash ('/').
3. The path should not conclude with a trailing slash ('/').
4. It should only encompass the necessary directories from the root directory to the specified target file or directory, excluding any instances of a single period ('.') or double period ('..').

The goal is to 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.

Bonus:

Write a C++ program to solve the N-Queens problem. The N-Queens problem involves placing N queens on an NxN chessboard in such a way that no two queens threaten each other. A queen can attack horizontally, vertically, and diagonally. Your program should take an integer value N as input from the user, where N represents the size of the chessboard and the number of queens to be placed. It should then find and display all possible solutions for placing N queens on the board. Your code should have the following

1. isSafe()

   This function checks if it's safe to place a queen at a specific position on the chessboard.

   Parameters:

   board: A 2D array representing the chessboard. row: The row where you want to check for safety. col: The column where you want to check for safety.

   n: The size of the board (N).

   Purpose:

   It verifies that there are no other queens in the same row.

   It checks the upper diagonal for any queens.

   It checks the lower diagonal for any queens.

   If all three conditions are met, the function returns true, indicating that it is safe to place a queen at the given position. Otherwise, it returns false.

2. printBoard()

   This function is responsible for printing the current state of the chessboard.

   Parameters:

   board: A 2D array representing the chessboard.

   n: The size of the board (N).

   Purpose:

   It iterates through the chessboard and prints "Q" for positions with queens and "." for empty cells, effectively displaying the current board configuration.
3. solveQueens()

   This function is the main algorithm for solving the N-Queens problem using backtracking.

   Parameters:

   n: The size of the chessboard.

   Purpose:

   It initializes an empty chessboard and a stack to track queen positions.

It uses a while loop to search for solutions, starting from the leftmost column. If a solution is found (all queens placed), it prints the board and backtracks by removing the last queen.

It attempts to place a queen in the current column and continues to the next row if it's safe.

If it's not possible to place a queen in the current column and there are queens in the stack, it backtracks to the previous column and row.

If there are no queens in the stack and no more solutions are possible, it exits the loop.

Home Tasks (Non-Graded)

1. Reverse a string using stacks
2. Implement a queue using stacks
3. Sort a stack using a temporary stack
4. Remove all duplicates of adjacent characters from a string using stacks.
5. Design a stack to support an additional operation that returns the minimum element from the stack in constant time. The stack should continue supporting all other operations like push, pop, top, size, empty, etc., with no degradation in these operations' performance.

6. Implement a queue using two stacks
7. Reverse a stack using recursion
8. Given a Queue consisting of first n natural numbers (in random order). The task is to check whether the given Queue elements can be arranged in increasing order in another Queue using a stack.