/\*

3. Reverse a string using a stack.

• Test Case 1:

Input: "hello"

Output: "olleh"

• Test Case 2:

Input: "world"

Output: "dlrow"

\*/

import java.util.Scanner;

public class question3 {

private char[] stack;

private int top;

private int maxSize;

public question3(int size) {

stack = new char[size];

maxSize = size;

top = -1;

}

public void push(char c) {

if (top < maxSize - 1) {

stack[++top] = c;

}

}

public char pop() {

if (top >= 0) {

return stack[top--];

}

return '\0'; // Null character for an empty stack

}

public boolean isEmpty() {

return top == -1;

}

public static String reverse(String input) {

int size = input.length();

question3 stack = new question3(size);

// Push all characters of the string to the stack

for (int i = 0; i < size; i++) {

stack.push(input.charAt(i));

}

// Pop all characters from the stack and build the reversed string

StringBuilder reversed = new StringBuilder();

while (!stack.isEmpty()) {

reversed.append(stack.pop());

}

return reversed.toString();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Take input from user

System.out.print("Enter a string to reverse: ");

String input = scanner.nextLine();

// Reverse the string and print the output

String output = reverse(input);

System.out.println("Reversed string: \"" + output + "\"");

}

}  
  
explanation  
**Stack Initialization**:

A **ReverseString** class handles the stack operations like push, pop, and check if empty.

**Main Method**:

Prompts the user to input a string.

Calls the **reverse** method to reverse the input string using the stack.

Prints the reversed string.

**Reverse Method**:

Takes the input string and initializes a stack with its length.

Pushes each character of the input string onto the stack.

Pops each character from the stack and appends it to a **StringBuilder** to form the reversed string.

**Stack Operations**:

**push(char c)**: Adds a character **c** to the stack.

**pop()**: Removes and returns the top character from the stack.

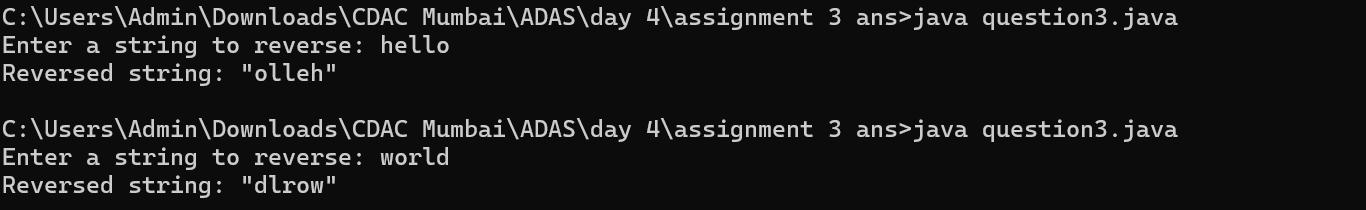
**isEmpty()**: Checks if the stack is empty.

**Time Complexity:**

**O(n)**: We iterate through the string twice (once to push and once to pop), which is linear with respect to the length of the input string.

**Space Complexity:**

**O(n)**

**Output  
**