Tutorial 07

(1) Why are stacks useful?

- Managing function calls: Stacks are coucial for managing function calls and tracking the execution flow in progoramming languages.
- Undo/Redo operations: Stacks are commonly used to implement undo/redo Functionality in applications.
- Expression evaluation: Stacks play a coucial role in evaluating arithmetic expressions. During expression parsing, operands are pushed onto the stack, and when an operator is encountered, the necessary number of operands is popped from the stack, the operation is performed, and the result is pushed back onto the stack.
- Balancing parentheses and tags: Stacks are helpful in checking and maintaining balanced parentheses, braces, or HTML/XML tags. As opening symbols are encountered, they are pushed onto the stack.

- Backtracking in algorithms: Many algorithms require backtracking, where the system needs to return to a previous state and explore alternative paths.
- Depth-first search: Stacks are integral to the implementation of depth-first search (DFS) algorithms.

Stacks offer a simple and efficient way to manage data in various scenarios. They provide a structured approach to handling elements in a LIFO manner, making them a valuable tool in programming, algorithm design, and other areas of computer science.

(2) Reverse a string using stack

```
def reverse_string(string):
    stack = []
    for char in string:
        stack.append(char) # Push each character onto the stack

    reversed_string = ""
    while len(stack)>0:
        reversed_string += stack.pop()
# Pop characters from the stack and append them to the reversed string

    return reversed_string

input_string = "Hello, World!"
reversed_string =
reverse_string(input_string)
print(reversed_string)
```

Output ⇒ !dlroW, olleH

(3) Write basic operations of queue

Enqueue - Adding an element to the queue

Dequeue - Removing an element from the queue

Front/Peek - Retrieving the Element at the front of the queue without removing it.

IsEmpty - Checking if the queue is empty

IsFull - Checking if the queue is full.

(4) What is balanced parentheses

Balanced parantheses refer to a situation where every opening parenthess has a corresponding closing paranthesis, and they are correctly nested.

For example, the following sequences are examples of balanced parantheses:

- "()"
- "((()))"
- "(()())"
- "()()()"

(5) Make stack implementation using a linked list - C program

```
#include <Stdio.h>
#include <Stdio.h>

//Define a structure for stack nodes typedef Struct Node{
    int data;
    Struct Node* next;
}Node;

//Define a structure for the stack typedef Struct Stack{
    Node* top;
}Stack;
```