Lecture Notes on Stack and Queue

Data Structures

Contents

1 Introduction **2**

2 Stack **2**

2.1 Definition . . . 2
2.2 Basic Operations . . . 2
2.3 Real-life Examples . . . 2
2.4 Types of Stack . . . 2
2.4.1 1. Static Stack . . . 2
2.4.2 2. Dynamic Stack . . . 3
2.5 Example Code (C++) . . . 3

3 Queue **3**

3.1 Definition . . . 3
3.2 Basic Operations . . . 3
3.3 Real-life Examples . . . 3
3.4 Types of Queue . . . 4
3.4.1 1. Simple Queue . . . 4
3.4.2 2. Circular Queue . . . 4
3.4.3 3. Double-ended Queue (Deque) . . . 4
3.4.4 4. Priority Queue . . . 4
3.5 Example Code (C++) . . . 4

4 Comparison of Stack and Queue **4**

5 Summary **5**

1 Introduction

In this lecture, we will study two important linear data structures:

- Stack
- Queue

Both Stack and Queue are simple yet very powerful structures. They are used to store and manage a collection of data elements in a specific order. Understanding them well helps in solving many programming and algorithmic problems.

2 Stack

2.1 Definition

A **Stack** is a linear data structure that follows the **LIFO** principle:

- **LIFO** = Last In, First Out
- The element that is inserted last is the first one to be removed.

2.2 Basic Operations

- **Push** – Add an element to the top of the stack.
- **Pop** – Remove the top element from the stack.
- **Peek / Top** – Return the top element without removing it.
- **isEmpty** – Check whether the stack is empty.

2.3 Real-life Examples

- A stack of plates in the kitchen.
- Browser history (Back button).
- Undo operation in text editors.

2.4 Types of Stack

2.4.1 1. Static Stack

- Implemented using an array.
- The size of the stack is fixed and known in advance.
- Pros: Fast access.
- Cons: Limited capacity.

2.4.2 2. Dynamic Stack

- Implemented using a linked list.
- The stack can grow and shrink dynamically as needed.
- Pros: No size limitation.
- Cons: Slightly more overhead due to pointers.

2.5 Example Code (C++)

```
stack<int> s;
s.push(10);
s.push(20);
s.pop();
int topElement = s.top();
```

3 Queue

3.1 Definition

A **Queue** is a linear data structure that follows the **FIFO** principle:

- **FIFO** = First In, First Out
- The element that is inserted first is the first one to be removed.

3.2 Basic Operations

- **Enqueue** – Add an element to the rear (back) of the queue.
- **Dequeue** – Remove an element from the front of the queue.
- **Front / Peek** – Return the front element without removing it.
- **isEmpty** – Check whether the queue is empty.

3.3 Real-life Examples

- People standing in a queue at a ticket counter.
- Print queue in a printer.
- Task scheduling in operating systems.

3.4 Types of Queue

3.4.1 1. Simple Queue

- Also known as Linear Queue.
- Insertion takes place at the rear end and deletion at the front end.
- Problem: After several dequeue operations, unused space is wasted.

3.4.2 2. Circular Queue

- The queue is treated as circular, so space is reused.
- The rear and front wrap around the array.
- Solves the space wastage problem of simple queue.

3.4.3 3. Double-ended Queue (Deque)

- Insertion and deletion can take place from both ends (front and rear).
- Two types:
  - **Input-restricted deque** – Insertion at rear only, deletion from both ends.
  - **Output-restricted deque** – Deletion at front only, insertion from both ends.

3.4.4 4. Priority Queue

- Each element is assigned a priority.
- The element with the highest priority is dequeued first.
- Not strictly FIFO.

3.5 Example Code (C++)

```
queue<int> q;
q.push(10);
q.push(20);
q.pop();
int frontElement = q.front();
```

4 Comparison of Stack and Queue

Property	Stack	Queue
Order	LIFO	FIFO
Insertion	Top	Rear
Deletion	Top	Front
Examples	Plates stack, Undo	Print queue, Call center

5 Summary

- Stack uses LIFO principle; Queue uses FIFO principle.
- Both can be implemented using arrays or linked lists.
- Variants of Stack: Static Stack, Dynamic Stack.
- Variants of Queue: Simple Queue, Circular Queue, Deque, Priority Queue.

Both Stack and Queue are foundational data structures. Mastering them will help you understand more advanced topics like recursion, graph traversal, task scheduling, and more.