UNIT 4 · LINEAR STRUCTURES

14. Circular Queue, Deque, and Priority Queue

CSC211 · Algorithms and Data Structures

COMSATS University Islamabad, Lahore Campus

Agenda

O

The Limitation, Recalled

Lecture 13's array queue permanently wastes freed slots

C

Circular Queue

Wraparound indices via the modulo operator

<>

Deque

Insertion and deletion allowed at both ends

#

Restricted Deques

Input-restricted and output-restricted variants

!

Priority Queue

Breaks FIFO on purpose — priority over arrival order

M

Min-Heap vs. Max-Heap

It's entirely a function of the comparator

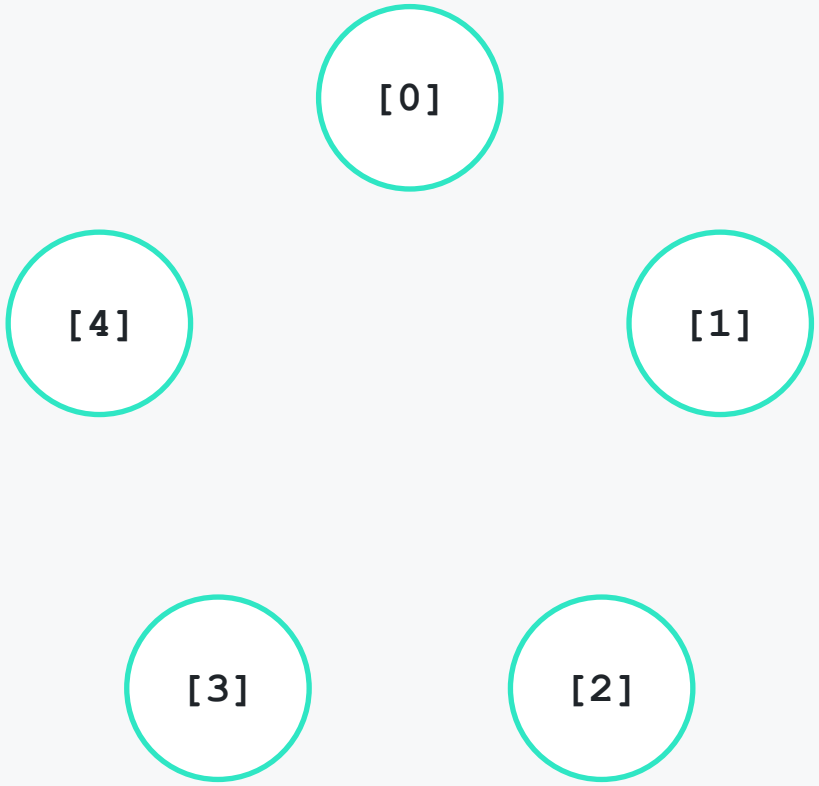
FIXING LECTURE 13'S WASTE

Circular Queue: The Array as a Ring

Both frontIndex and rearIndex advance with % CAPACITY, so they wrap around and reuse freed slots instead of abandoning them.

% 5

the entire wraparound mechanism: one modulo operator



index 4 wraps back to index 0 via % CAPACITY

IMPLEMENTATION

CircularQueue: count Replaces rearIndex

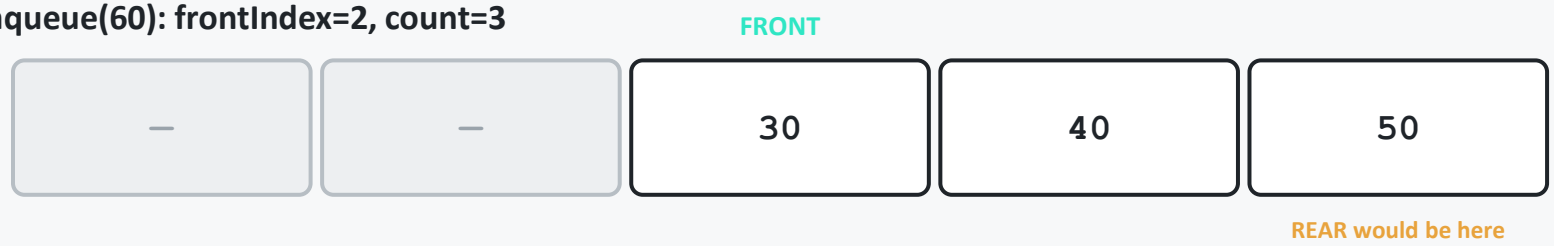
circular_queue.cpp

```
class CircularQueue {
    int data[CAPACITY];
    int frontIndex, count;   // count = elements stored right now

    bool isFull() const { return count == CAPACITY; }

    void enqueue(int value) {
        if (isFull()) throw overflow_error("Queue is full");
        int rearIndex = (frontIndex + count) % CAPACITY;
        data[rearIndex] = value;  count++;
    }

    int dequeue() {
        int value = data[frontIndex];
        frontIndex = (frontIndex + 1) % CAPACITY;  count--;
        return value;
    }
};
```

THE PART THAT TRIPS STUDENTS UP

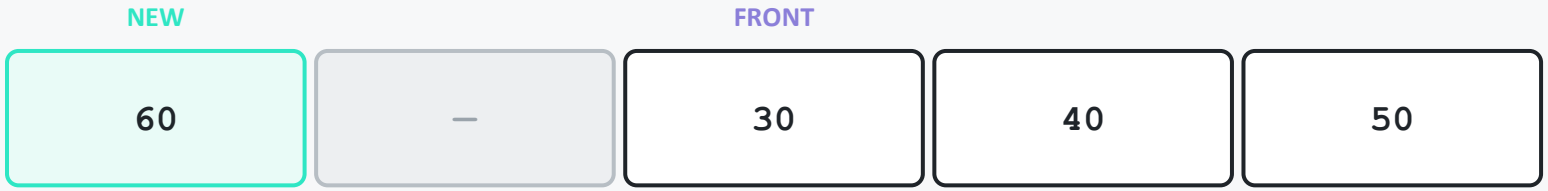
Tracing the Wraparound, Step by Step

Before enqueue(60): frontIndex=2, count=3




```
(2 + 3) % 5 = 0  ->  wraps!
```

After enqueue(60): rearIndex wraps 5 -> 0



WHY IT MATTERS

Advantages of a Circular Queue

1

Uses Memory Efficiently

No slot is ever permanently wasted — every freed slot gets reused.

2

Still O(1)

Every operation stays O(1), with none of a linked list's per-node pointer overhead.

3

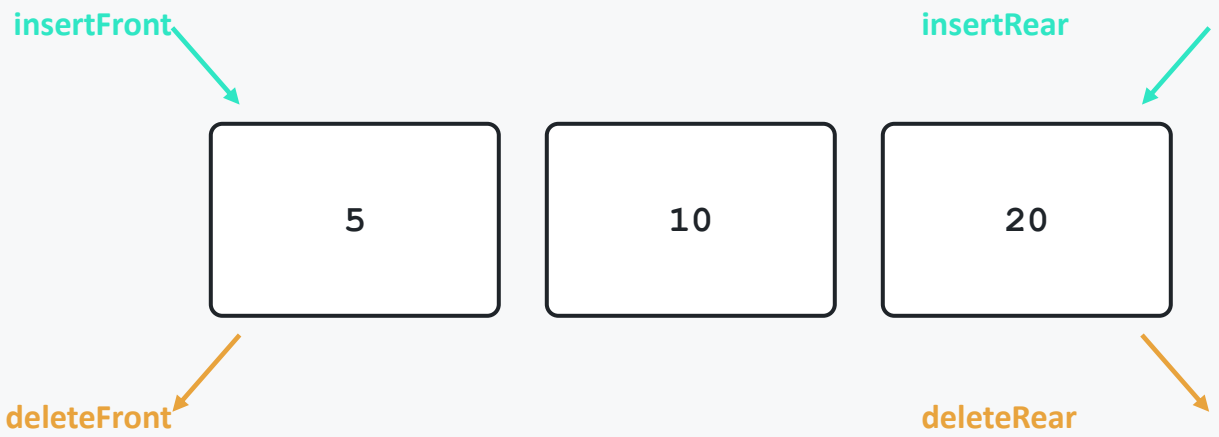
The Real-World Standard

Audio/video streaming buffers, keyboard input buffers, and network packet buffers are all circular queues under the hood.

GENERALIZING FURTHER

Deque: Double-Ended Queue

Insertion and deletion are both allowed at either end — not just one.



A queue = a deque using only push_back + pop_front. A stack = a deque using only push_back + pop_back. The deque is the more general structure both are built from.

ArrayDeque: frontIndex Moves Backward Too

array_deque.cpp

```
void insertRear(int value) {
    int rearIndex = (frontIndex + count) % CAPACITY;
    data[rearIndex] = value;  count++;
}

void insertFront(int value) {
    // moves BACKWARD, wrapping to CAPACITY - 1 instead of -1
    frontIndex = (frontIndex - 1 + CAPACITY) % CAPACITY;
    data[frontIndex] = value;  count++;
}

int deleteRear() {
    int rearIndex = (frontIndex + count - 1) % CAPACITY;
    count--;  return data[rearIndex];
}
```

ONE END LOCKED DOWN

Input-Restricted & Output-Restricted Deques

Input-Restricted Deque

Insertion allowed at only ONE end.

Deletion allowed at BOTH ends.

Output-Restricted Deque

Deletion allowed at only ONE end.

Insertion allowed at BOTH ends.

Used when a problem is mostly a plain queue or stack, but occasionally needs one extra flexibility at just one end.

BREAKING FIFO ON PURPOSE

Priority Queue: Priority Over Arrival Order

Pushed in this order:

- `[2] Send weekly report`
- `[9] Fix production outage`
- `[1] Reply to a comment`
- `[8] Patch a security bug`

Popped in this order (highest priority first):

- **`[9] Fix production outage`**
- `[8] Patch a security bug`
- `[2] Send weekly report`
- `[1] Reply to a comment`

"Fix production outage" was pushed 2nd but processed 1st — its priority (9) beats every other task's.

IT'S JUST THE COMPARATOR

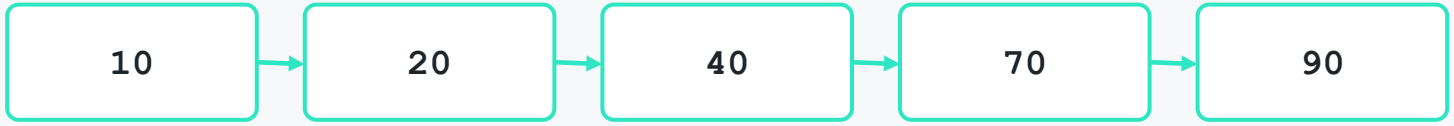
Min-Heap vs. Max-Heap

max-heap<int>

(default: less<T>)

min-heap<int>

(greater<int>)



Comparator	top() returns	Use it for...
`less<T> (default)`	The largest element	"Most urgent first" — outage tickets, leaderboard
`greater<T>`	The smallest element	"Soonest first" — nearest deadline, lowest bid

KEY TAKEAWAYS

Key Takeaways

- A circular queue reuses freed array slots by wrapping indices with the modulo operator, fixing wasted space while staying O(1).
- rearIndex = (frontIndex + count) % CAPACITY, recomputed fresh every time, is what makes the wraparound arithmetic safe to reason about.
- A deque generalizes the queue to allow insertion and deletion at both ends — stacks and queues are both special cases of a deque.
- Input-restricted and output-restricted deques lock down one end when a problem needs only a little extra flexibility.
- A priority queue breaks FIFO deliberately; whether std::priority_queue is a max-heap or min-heap is entirely a function of its comparator.