Day 23: Seating Arrangement, Circular Queues, & Dynamic Functions

1. Logical Reasoning: Seating Arrangement

These problems test positional reasoning and the ability to visualize constraints.

- **Linear Arrangement:** Objects or people are in a straight line.
  - *Tip:* Always identify "definite" positions first (e.g., "A is at the extreme left end").
- **Circular Arrangement:** Objects or people are around a circle.
  - *Facing Center:* Right is anti-clockwise, Left is clockwise.
  - *Facing Outside:* Right is clockwise, Left is anti-clockwise.
  - *Constraint Check:* If there are N people, the person opposite person i is only clearly defined if N is even.

2. Programming: Circular Queue Implementation

A Circular Queue overcomes the "waste of space" limitation of a linear queue by connecting the last position back to the first.

Key Logic:

- **Front:** Index of the first element.
- **Rear:** Index of the last element.
- **Enqueue:** `rear = (rear + 1) % size`
- **Dequeue:** `front = (front + 1) % size`
- **Full Condition:** `(rear + 1) % size == front`
- **Empty Condition:** `front == -1`

```
class CircularQueue:
    def __init__(self, k):
        self.k = k
        self.queue = [None] * k
        self.head = self.tail = -1

    def enqueue(self, value):
        if (self.tail + 1) % self.k == self.head:
            return False # Full
        if self.head == -1:
            self.head = 0
        self.tail = (self.tail + 1) % self.k
        self.queue[self.tail] = value
        return True

    def dequeue(self):
        if self.head == -1:
```

```
            return False # Empty
        if self.head == self.tail:
            self.head = self.tail = -1
        else:
            self.head = (self.head + 1) % self.k
        return True
```

3. Python Concept: Lambda Functions

Lambdas are small, one-line anonymous functions.

Syntax: `lambda arguments: expression`

Common Use Cases:

- **Sorting:** `sorted(points, key=lambda x: x[1])` (sort by second element).
- **Mapping:** `list(map(lambda x: x.upper(), names))`
- **Filtering:** `list(filter(lambda x: x > 10, numbers))`

4. C/C++ Concept: Function Pointers

Function pointers allow you to store the address of a function in a variable, enabling callback mechanisms and dynamic execution.

Syntax: `return_type (*pointer_name)(argument_types);`

Example:

```
void greet() { printf("Hello!"); }

int main() {
    void (*funcPtr)() = &greet; // Point to greet function
    funcPtr();                  // Execute via pointer
    return 0;
}
```

- **Use Case:** Passing a comparison function to `qsort()` or implementing a "Strategy" design pattern.

5. SQL: EXISTS Clause

The `EXISTS` operator is used to test for the existence of any record in a subquery.

- **Behavior:** It returns `TRUE` if the subquery returns one or more records. It is often more efficient than `IN` because it stops scanning as soon as a single match is found.
- **Syntax:**

```
SELECT customer_name
FROM customers c
```

```
WHERE EXISTS (
    SELECT 1
    FROM orders o
    WHERE o.customer_id = c.id AND o.amount > 1000
);
```

- **NOT EXISTS:** Used to find records that do *not* have a match in the related table (e.g., customers who have never placed an order).