LAB-04

Q1 Output:

Output

```
1. Produce  2. Consume  3. Exit
Enter your choice: 1

Enter the value: 2

1. Produce  2. Consume  3. Exit
Enter your choice: 4

1. Produce  2. Consume  3. Exit
Enter your choice: 2

The consumed value is 2
1. Produce  2. Consume  3. Exit
Enter your choice: 2

Buffer is Empty
1. Produce  2. Consume  3. Exit
Enter your choice: 3


=== Code Execution Successful ===
```

LAB-04

Q2 Output:

```
Produced 21 | Buffer: [21]
Consumed 21 | Buffer: []
Produced 62 | Buffer: [62]
Produced 3 | Buffer: [62, 3]
Consumed 62 | Buffer: [3]
Produced 84 | Buffer: [3, 84]
Consumed 3 | Buffer: [84]
Produced 42 | Buffer: [84, 42]
```

LAB-04

3) In producer-consumer problem what difference will it make if we utilize stack for the buffer rather than an array?

1. Default Buffer Structure (Queue - FIFO)
Typically, the producer-consumer problem is implemented using a queue (array or linked list) where the producer inserts items at the rear and the consumer removes items from the front.
This follows a First-In-First-Out (FIFO) order.
Example: If items are produced in order [A, B, C], they will be consumed in the same order: A → B → C.
2. Using a Stack Instead of a Queue
A stack follows a Last-In-First-Out (LIFO) order.
The most recently produced item is consumed first.
Example: If items are produced in order [A, B, C], they will be consumed in the reverse order: C → B → A.
3. Real-World Implications
Queue (FIFO) is better when older data needs to be processed first (e.g., task scheduling, message queues).
Stack (LIFO) is better for cases where the latest data is most relevant (e.g., browser history, function calls, or stock market trades).
4. Impact on Producer-Consumer Synchronization
Both require synchronization (mutex + semaphore) to prevent race conditions. However, with a stack, there might be less predictable behavior in scenarios where ordering matters.