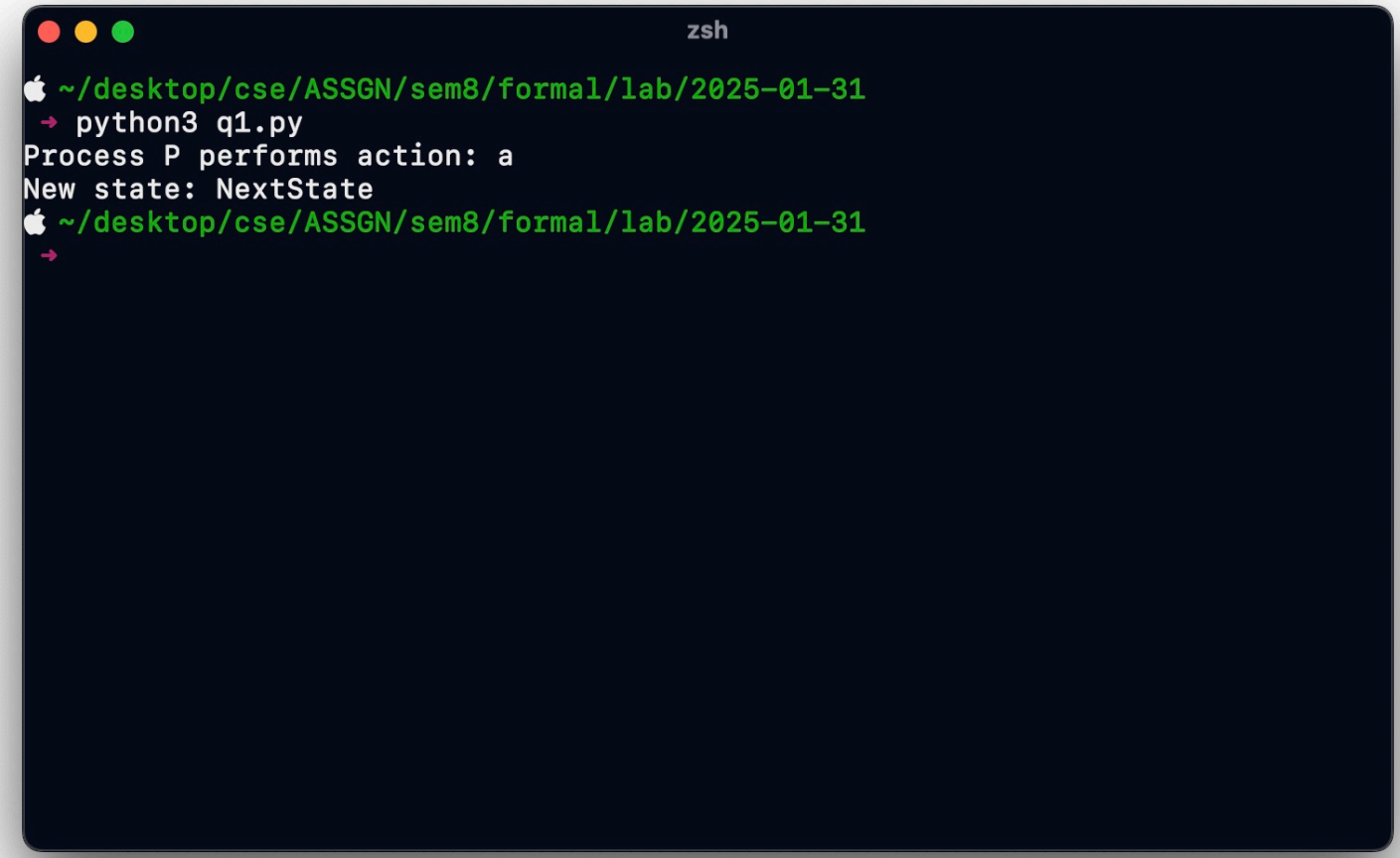
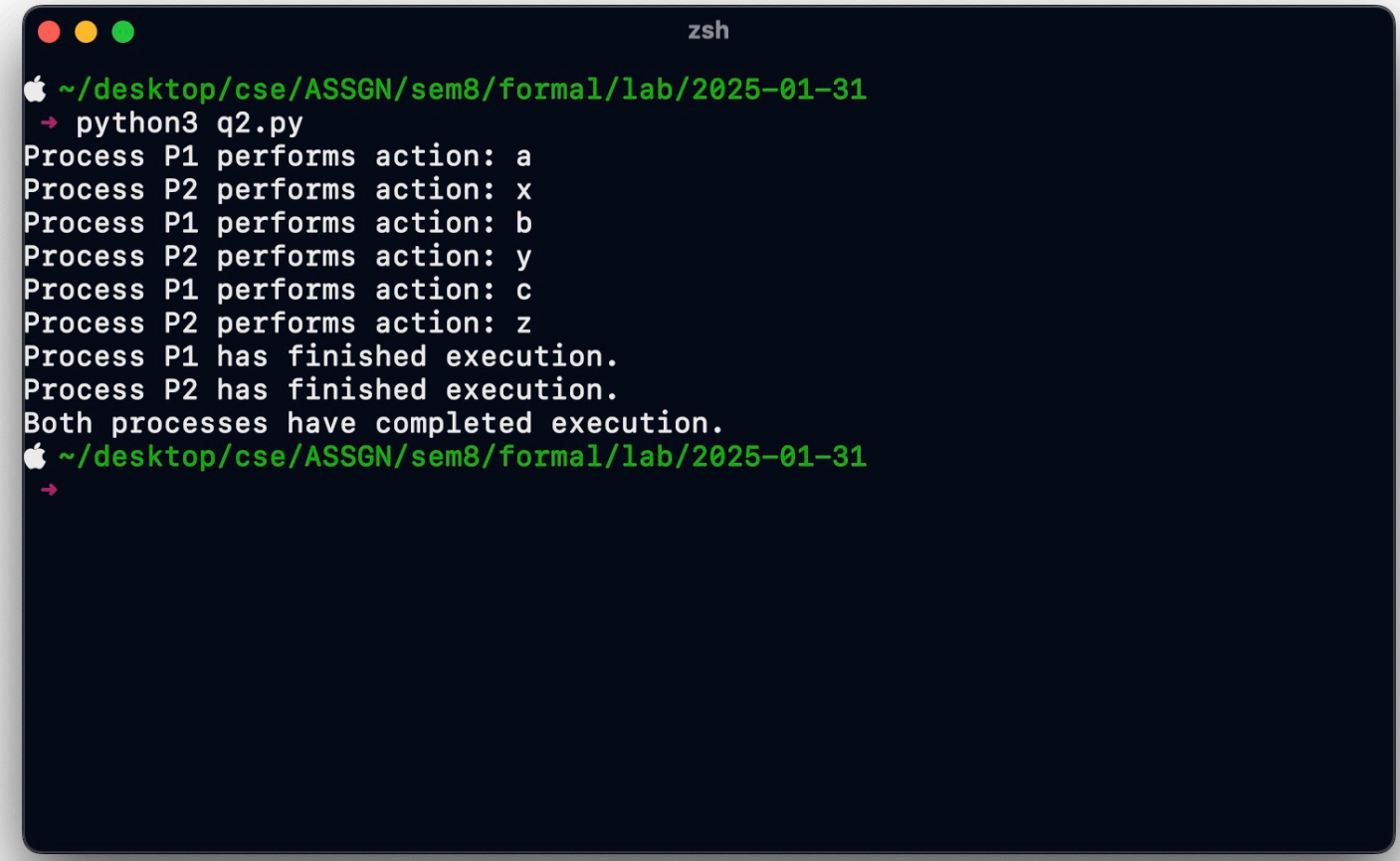
ASSIGNMENT 2
31/01/25

NAME : SHRESTH SONKAR
REGNO : 20214272
GROUP : CS8D
TOPIC : FORMAL METHOD
CODE : CS-18201

```python
# Simulate a basic CCS process in Python where one process performs an action (a) and
# transitions to the next state.
class CCSProcess:
    def __init__(self, name):
        self.name = name
        self.state = "Start"

    def transition(self, action):
        print(f"Process {self.name} performs action: {action}")
        self.state = "NextState"

    def current_state(self):
        return self.state

process = CCSProcess("P")
process.transition("a")
print(f"New state: {process.current_state()}")
```

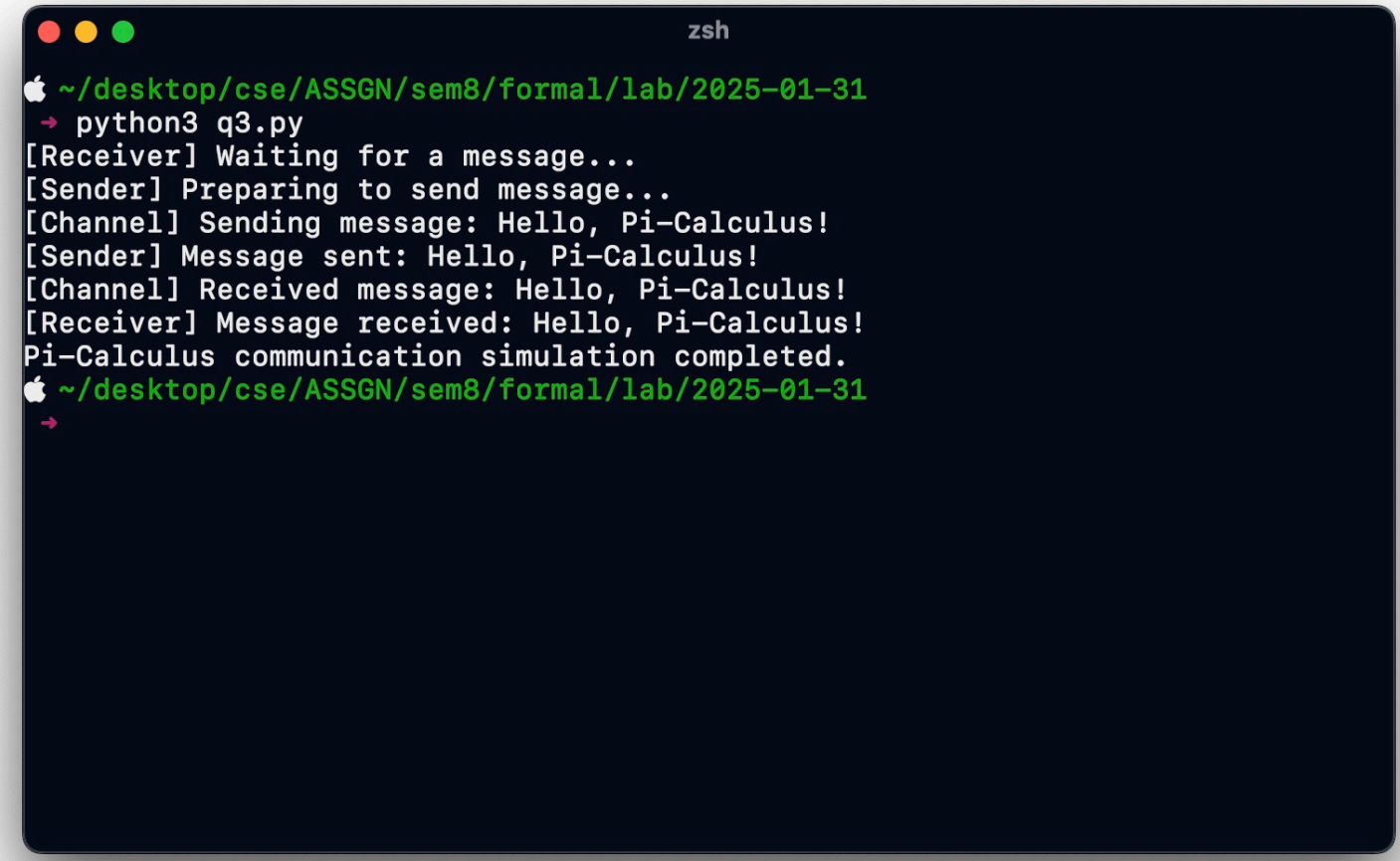


```python
# Model and simulate a parallel composition of two CCS
processes in Python, where both
# processes execute concurrently.

import threading
import time

class CCSProcess:
    def __init__(self, name, actions):
        self.name = name
        self.actions = actions
        self.current_index = 0

    def execute(self):
        while self.current_index < len(self.actions):
            action = self.actions[self.current_index]
            print(f"Process {self.name} performs
action: {action}")
            self.current_index += 1
            time.sleep(1)

        print(f"Process {self.name} has finished
execution.")

def parallel_composition(process1, process2):
    thread1 = threading.Thread(target=process1.execute)
    thread2 = threading.Thread(target=process2.execute)

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()
    print("Both processes have completed execution.")

process_A = CCSProcess("P1", ["a", "b", "c"])
process_B = CCSProcess("P2", ["x", "y", "z"])

parallel_composition(process_A, process_B)
```



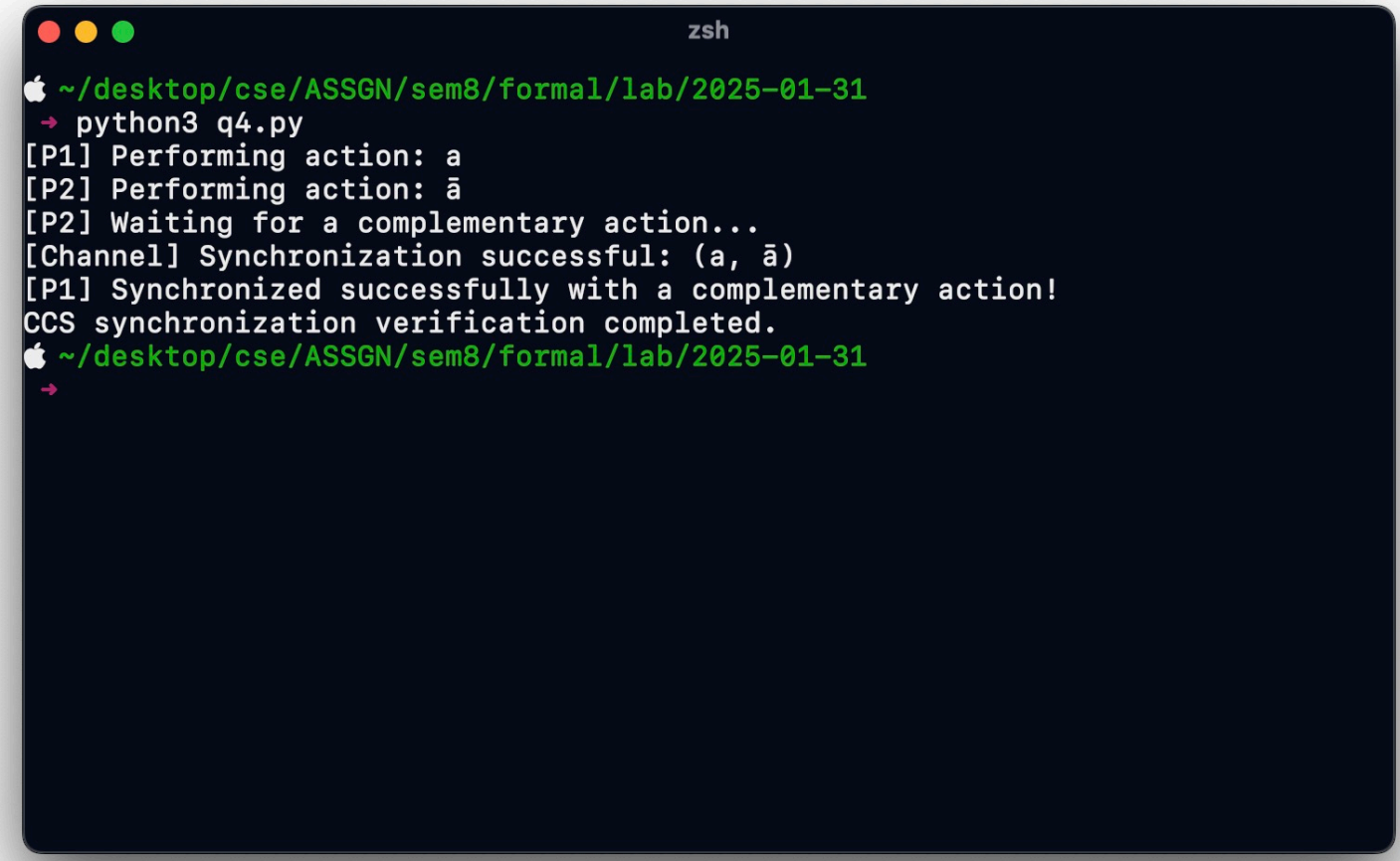

```python
# Implement Pi-Calculus communication in Python

import threading
import queue
import time

class PiChannel:
    def __init__(self):
        self.channel = queue.Queue()

    def send(self, message):
        print(f"[Channel] Sending message: {message}")
        self.channel.put(message)

    def receive(self):
        message = self.channel.get()
        print(f"[Channel] Received message: {message}")
        return message


class SenderProcess:
```

```
    def __init__(self, name, channel, message):
        self.name = name
        self.channel = channel
        self.message = message

    def send_message(self):
        print(f"[{self.name}] Preparing to send
message...")
        time.sleep(1)
        self.channel.send(self.message)
        print(f"[{self.name}] Message sent:
{self.message}")


class ReceiverProcess:
    def __init__(self, name, channel):
        self.name = name
        self.channel = channel

    def receive_message(self):
        print(f"[{self.name}] Waiting for a
message...")
        message = self.channel.receive()
        print(f"[{self.name}] Message received:
{message}")

channel = PiChannel()

sender = SenderProcess("Sender", channel, "Hello, Pi-
Calculus!")
receiver = ReceiverProcess("Receiver", channel)

sender_thread =
threading.Thread(target=sender.send_message)
receiver_thread =
threading.Thread(target=receiver.receive_message)

receiver_thread.start()
sender_thread.start()

receiver_thread.join()
sender_thread.join()

print("Pi-Calculus communication simulation
completed.")
```



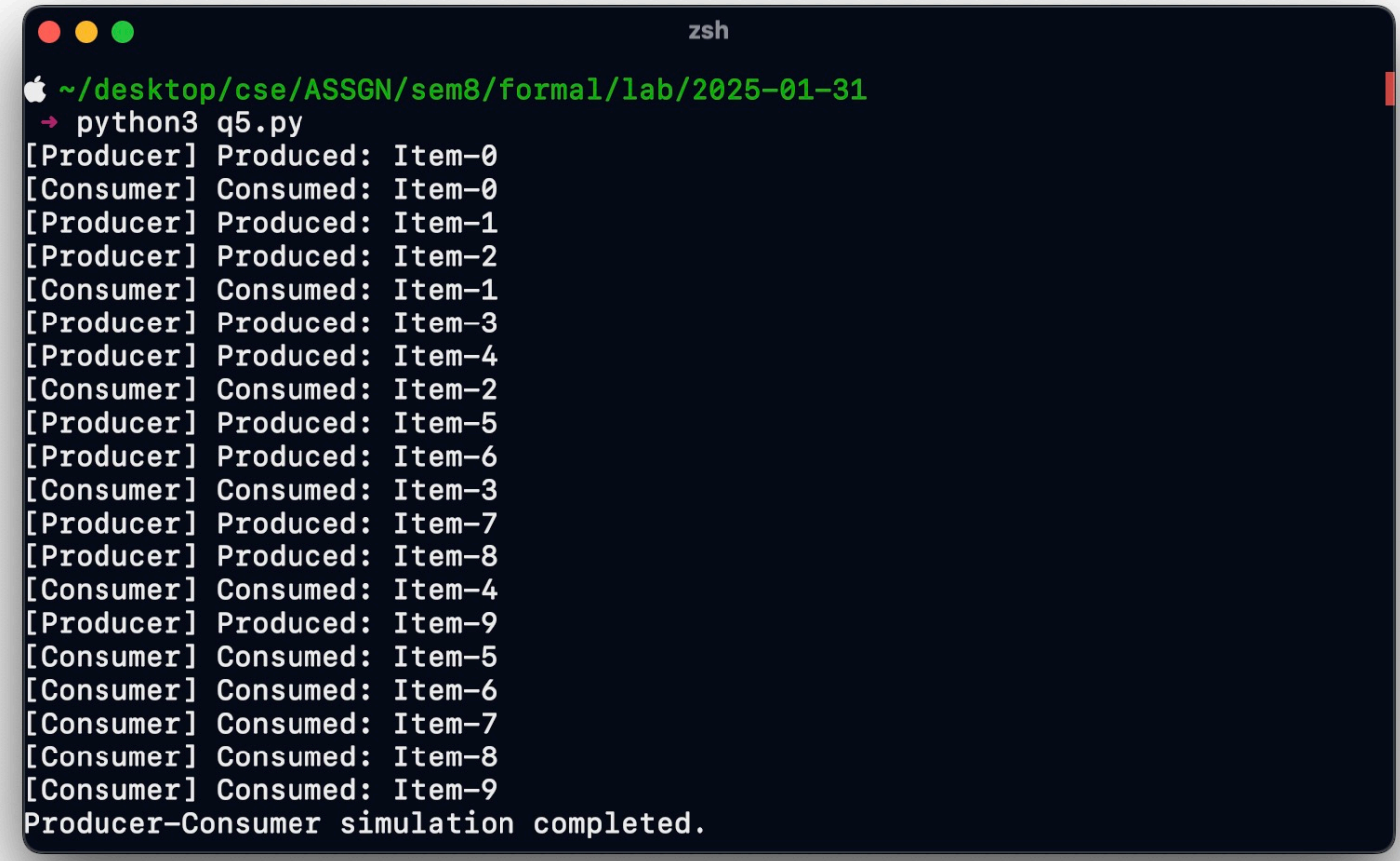

```python
# Write a Python program to verify synchronization
between two CCS processes

import threading
import queue
import time

class CCSChannel:
    def __init__(self):
        self.channel = queue.Queue()

    def synchronize(self, action):
        try:
            complement = "ā" if action == "a" else "a"
            if not self.channel.empty() and
self.channel.queue[0] == complement:
                self.channel.get()
                print(f"[Channel] Synchronization
successful: ({action}, {complement})")
                return True
            else:
```

```
                self.channel.put(action)
                return False
        except Exception as e:
            print(f"[Channel] Error: {e}")
            return False


class CCSProcess:
    def __init__(self, name, action, channel):
        self.name = name
        self.action = action
        self.channel = channel

    def execute(self):
        print(f"[{self.name}] Performing action:
{self.action}")
        time.sleep(1)

        if self.channel.synchronize(self.action):
            print(f"[{self.name}] Synchronized
successfully with a complementary action!")
        else:
            print(f"[{self.name}] Waiting for a
complementary action...")


channel = CCSChannel()

process1 = CCSProcess("P1", "a", channel)
process2 = CCSProcess("P2", "ā", channel)

thread1 = threading.Thread(target=process1.execute)
thread2 = threading.Thread(target=process2.execute)

thread1.start()
thread2.start()

thread1.join()
thread2.join()

print("CCS synchronization verification completed.")
```




```python
# Simulate a basic producer-consumer system using
Python

import threading
import queue
import time

class ProducerConsumer:
    def __init__(self, buffer_size=5):
        self.buffer = queue.Queue(maxsize=buffer_size)
        self.lock = threading.Lock()
        self.produce_count = 0
        self.consume_count = 0

    def produce(self):
        while self.produce_count < 10:
            time.sleep(1)
            item = f"Item-{self.produce_count}"

            with self.lock:
                if not self.buffer.full():
                    self.buffer.put(item)
```

```
                    print(f"[Producer] Produced:
{item}")
                    self.produce_count += 1
                else:
                    print("[Producer] Buffer full!
Waiting...")

    def consume(self):
        while self.consume_count < 10:
            time.sleep(2)

            with self.lock:
                if not self.buffer.empty():
                    item = self.buffer.get()
                    print(f"[Consumer] Consumed:
{item}")
                    self.consume_count += 1
                else:
                    print("[Consumer] Buffer empty!
Waiting...")

pc_system = ProducerConsumer(buffer_size=5)

producer_thread =
threading.Thread(target=pc_system.produce)
consumer_thread =
threading.Thread(target=pc_system.consume)

producer_thread.start()
consumer_thread.start()

producer_thread.join()
consumer_thread.join()

print("Producer-Consumer simulation completed.")
```


zsh
~/desktop/cse/ASSGN/sem8/formal/lab/2025-01-31
➜ python3 q5.py
[Producer] Produced: Item-0
[Consumer] Consumed: Item-0
[Producer] Produced: Item-1
[Producer] Produced: Item-2
[Consumer] Consumed: Item-1
[Producer] Produced: Item-3
[Producer] Produced: Item-4
[Consumer] Consumed: Item-2
[Producer] Produced: Item-5
[Producer] Produced: Item-6
[Consumer] Consumed: Item-3
[Producer] Produced: Item-7
[Producer] Produced: Item-8
[Consumer] Consumed: Item-4
[Producer] Produced: Item-9
[Consumer] Consumed: Item-5
[Consumer] Consumed: Item-6
[Consumer] Consumed: Item-7
[Consumer] Consumed: Item-8
[Consumer] Consumed: Item-9
Producer-Consumer simulation completed.