```swift
import Foundation
import Foundation

enum QueueError: Error {
case emptyQueue
case fullQueue
}

class Node<T> {
let value: T
var next: Node<T>?
init(_ value: T) {
self.value = value
}
}

class Queue<T> {
private let maxSize: Int?
private var front: Node<T>?
private var rear: Node<T>?
private var count: Int = 0
init() {
maxSize = nil
}
init(from array: [T]) {
maxSize = nil
for item in array {
do {
try enqueue(item)
} catch {
print("Error:", error)
}
}
}
init(size: Int) {
maxSize = size
}
var length: Int {
return count
}
func enqueue(_ item: T) throws {
guard maxSize == nil || count < maxSize! else {
throw QueueError.fullQueue
}
let newNode = Node(item)
if front == nil {
front = newNode
rear = newNode
} else {
rear?.next = newNode
// rear = newNode
}
count += 1
}
func dequeue() throws -> T {
if front == nil {
throw QueueError.emptyQueue
}
let dequeuedValue = front!.value
front = front!.next
if front == nil {
rear = nil // Reset rear when the queue becomes empty
}
count -= 1
return dequeuedValue
}
func printContents() {
var current = front
var contents: [T] = []
while let currentNode = current {
contents.append(currentNode.value)
current = currentNode.next
}
print(contents)
}
}

// Example usage

// Creating an empty queue
var emptyQueue = Queue<Int>()
emptyQueue.printContents() // Output: []

// Creating a queue from an array
let arrayQueue = Queue(from: [4, 5, 6])
arrayQueue.printContents() // Output: [4, 5, 6]

// Creating a fixed size queue
var fixedSizeQueue = Queue<String>(size: 2)
```

```swift
do {
try fixedSizeQueue.enqueue("A")
try fixedSizeQueue.enqueue("B")
//try fixedSizeQueue.enqueue("C") // Throws QueueError.fullQueue
} catch {
print("Error:", error)
}

print("Queue length:", fixedSizeQueue.length) // Output: 2
fixedSizeQueue.printContents() // Output: ["A", "B"]

do {
let dequeuedItem = try fixedSizeQueue.dequeue()
print("Dequeued item:", dequeuedItem) // Output: Dequeued item: A
}
catch {
print("Error:", error)
}
```