CIA-1

Date : 16/01/24

1. Stack using array

In [3]:
```python
class StackArray:
    def __init__(self):
        self.stack = []

    def is_empty(self):
        return len(self.stack) == 0

    def push(self, item):
        self.stack.append(item)

    def pop(self):
        if not self.is_empty():
            return self.stack.pop()

    def peek(self):
        if not self.is_empty():
            return self.stack[-1]

    def size(self):
        return len(self.stack)

stack_array = StackArray()
stack_array.push(1)
stack_array.push(2)
stack_array.push(3)

print(stack_array.peek())  # Output: 3
print(stack_array.pop())   # Output: 3
print(stack_array.size())  # Output: 2
```

```
3
3
2
```

2. Stack using linked list

In [4]:

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class StackLinkedList:
    def __init__(self):
        self.top = None

    def is_empty(self):
        return self.top is None

    def push(self, item):
        new_node = Node(item)
        new_node.next = self.top
        self.top = new_node

    def pop(self):
        if not self.is_empty():
            popped = self.top
            self.top = self.top.next
            return popped.data

    def peek(self):
        if not self.is_empty():
            return self.top.data

    def size(self):
        current = self.top
        count = 0
        while current:
            count += 1
            current = current.next
        return count

stack_linked_list = StackLinkedList()
stack_linked_list.push(1)
stack_linked_list.push(2)
stack_linked_list.push(3)

print(stack_linked_list.peek())  # Output: 3
print(stack_linked_list.pop())   # Output: 3
print(stack_linked_list.size())  # Output: 2
```

```
3
3
2
```

3. Queue using array

In [7]:

```python
class QueueArray:
    def __init__(self):
        self.queue = []

    def is_empty(self):
        return len(self.queue) == 0

    def enqueue(self, item):
        self.queue.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.queue.pop(0)

    def front(self):
        if not self.is_empty():
            return self.queue[0]

    def size(self):
        return len(self.queue)

queue_array = QueueArray()
queue_array.enqueue(1)
queue_array.enqueue(2)
queue_array.enqueue(3)

print(queue_array.front())  # Output: 1
print(queue_array.dequeue())  # Output: 1
print(queue_array.size())  # Output: 2

```

```
1
1
2
```

4. Queue using linked list

In [10]:

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class QueueLinkedList:
    def __init__(self):
        self.front = self.rear = None

    def is_empty(self):
        return self.front is None

    def enqueue(self, item):
        new_node = Node(item)
        if self.is_empty():
            self.front = self.rear = new_node
        else:
            self.rear.next = new_node
            self.rear = new_node

    def dequeue(self):
        if not self.is_empty():
            popped = self.front
            self.front = popped.next
            if self.front is None:
                self.rear = None
            return popped.data

    def get_front(self):  # Rename the method to get_front
        if not self.is_empty():
            return self.front.data

    def size(self):
        current = self.front
        count = 0
        while current:
            count += 1
            current = current.next
        return count

queue_linked_list = QueueLinkedList()
queue_linked_list.enqueue(1)
queue_linked_list.enqueue(2)
queue_linked_list.enqueue(3)

print(queue_linked_list.get_front())  # Output: 1
print(queue_linked_list.dequeue())    # Output: 1
print(queue_linked_list.size())       # Output: 2

```

```
1
1
2
```

5. Priority queue

In [13]:

```python
import heapq

class PriorityQueue:
    def __init__(self):
        self.heap = []

    def is_empty(self):
        return len(self.heap) == 0

    def enqueue(self, item, priority):
        heapq.heappush(self.heap, (priority, item))

    def dequeue(self):
        if not self.is_empty():
            return heapq.heappop(self.heap)[1]

    def get_front(self):
        if not self.is_empty():
            return self.heap[0][1]

    def size(self):
        return len(self.heap)

# Test PriorityQueue with task handling visualization
priority_queue = PriorityQueue()

# Enqueue tasks with priorities
priority_queue.enqueue("Task 1", 3)
priority_queue.enqueue("Task 2", 1)
priority_queue.enqueue("Task 3", 2)
priority_queue.enqueue("Task 4", 5)
priority_queue.enqueue("Task 5", 4)
priority_queue.enqueue("Task 6", 2)

# Dequeue and visualize task handling
while not priority_queue.is_empty():
    print(f"Handling: {priority_queue.dequeue()}  Remaining tasks: {prior

```

```
Handling: Task 2  Remaining tasks: 5
Handling: Task 3  Remaining tasks: 4
Handling: Task 6  Remaining tasks: 3
Handling: Task 1  Remaining tasks: 2
Handling: Task 5  Remaining tasks: 1
Handling: Task 4  Remaining tasks: 0
```

6. Circular queue

In [11]:

```python
class CircularQueue:
    def __init__(self, capacity):
        self.queue = [None] * capacity
        self.front = self.rear = -1
        self.capacity = capacity

    def is_empty(self):
        return self.front == -1

    def is_full(self):
        return (self.rear + 1) % self.capacity == self.front

    def enqueue(self, item):
        if self.is_full():
            return
        if self.is_empty():
            self.front = self.rear = 0
        else:
            self.rear = (self.rear + 1) % self.capacity
        self.queue[self.rear] = item

    def dequeue(self):
        if self.is_empty():
            return None
        removed_item = self.queue[self.front]
        if self.front == self.rear:
            self.front = self.rear = -1
        else:
            self.front = (self.front + 1) % self.capacity
        return removed_item

    def get_front(self):
        if not self.is_empty():
            return self.queue[self.front]

    def size(self):
        if self.is_empty():
            return 0
        elif self.front <= self.rear:
            return self.rear - self.front + 1
        else:
            return self.capacity - self.front + self.rear + 1

# Test CircularQueue
circular_queue = CircularQueue(3)
circular_queue.enqueue(1)
circular_queue.enqueue(2)
circular_queue.enqueue(3)

print(circular_queue.get_front())  # Output: 1
print(circular_queue.dequeue())    # Output: 1
print(circular_queue.size())       # Output: 2

```

```
1
1
2
```