

CIA-1

Date : 16/01/24

1. Stack using array

```
In [3]: 1 class StackArray:
2         def __init__(self):
3             self.stack = []
4
5         def is_empty(self):
6             return len(self.stack) == 0
7
8         def push(self, item):
9             self.stack.append(item)
10
11        def pop(self):
12            if not self.is_empty():
13                return self.stack.pop()
14
15        def peek(self):
16            if not self.is_empty():
17                return self.stack[-1]
18
19        def size(self):
20            return len(self.stack)
21
22        stack_array = StackArray()
23        stack_array.push(1)
24        stack_array.push(2)
25        stack_array.push(3)
26
27        print(stack_array.peek()) # Output: 3
28        print(stack_array.pop())  # Output: 3
29        print(stack_array.size()) # Output: 2
```

3
3
2

2. Stack using linked list

```
In [4]: 1 class Node:
2         def __init__(self, data):
3             self.data = data
4             self.next = None
5
6         class StackLinkedList:
7             def __init__(self):
8                 self.top = None
9
10            def is_empty(self):
11                return self.top is None
12
13            def push(self, item):
14                new_node = Node(item)
15                new_node.next = self.top
16                self.top = new_node
17
18            def pop(self):
19                if not self.is_empty():
20                    popped = self.top
21                    self.top = self.top.next
22                    return popped.data
23
24            def peek(self):
25                if not self.is_empty():
26                    return self.top.data
27
28            def size(self):
29                current = self.top
30                count = 0
31                while current:
32                    count += 1
33                    current = current.next
34                return count
35
36 stack_linked_list = StackLinkedList()
37 stack_linked_list.push(1)
38 stack_linked_list.push(2)
39 stack_linked_list.push(3)
40
41 print(stack_linked_list.peek()) # Output: 3
42 print(stack_linked_list.pop())  # Output: 3
43 print(stack_linked_list.size()) # Output: 2
```

3
3
2

3. Queue using array

```
In [7]: 1 class QueueArray:
2         def __init__(self):
3             self.queue = []
4
5         def is_empty(self):
6             return len(self.queue) == 0
7
8         def enqueue(self, item):
9             self.queue.append(item)
10
11        def dequeue(self):
12            if not self.is_empty():
13                return self.queue.pop(0)
14
15        def front(self):
16            if not self.is_empty():
17                return self.queue[0]
18
19        def size(self):
20            return len(self.queue)
21
22        queue_array = QueueArray()
23        queue_array.enqueue(1)
24        queue_array.enqueue(2)
25        queue_array.enqueue(3)
26
27        print(queue_array.front()) # Output: 1
28        print(queue_array.dequeue()) # Output: 1
29        print(queue_array.size()) # Output: 2
30
```

```
1
1
2
```

4. Queue using linked list

```

In [10]: 1 class Node:
2         def __init__(self, data):
3             self.data = data
4             self.next = None
5
6         class QueueLinkedList:
7             def __init__(self):
8                 self.front = self.rear = None
9
10            def is_empty(self):
11                return self.front is None
12
13            def enqueue(self, item):
14                new_node = Node(item)
15                if self.is_empty():
16                    self.front = self.rear = new_node
17                else:
18                    self.rear.next = new_node
19                    self.rear = new_node
20
21            def dequeue(self):
22                if not self.is_empty():
23                    popped = self.front
24                    self.front = popped.next
25                    if self.front is None:
26                        self.rear = None
27                    return popped.data
28
29            def get_front(self): # Rename the method to get_front
30                if not self.is_empty():
31                    return self.front.data
32
33            def size(self):
34                current = self.front
35                count = 0
36                while current:
37                    count += 1
38                    current = current.next
39                return count
40
41            queue_linked_list = QueueLinkedList()
42            queue_linked_list.enqueue(1)
43            queue_linked_list.enqueue(2)
44            queue_linked_list.enqueue(3)
45
46            print(queue_linked_list.get_front()) # Output: 1
47            print(queue_linked_list.dequeue()) # Output: 1
48            print(queue_linked_list.size()) # Output: 2
49

```

1

1

2

5. Priority queue

```

In [13]: 1 import heapq
2
3 class PriorityQueue:
4     def __init__(self):
5         self.heap = []
6
7     def is_empty(self):
8         return len(self.heap) == 0
9
10    def enqueue(self, item, priority):
11        heapq.heappush(self.heap, (priority, item))
12
13    def dequeue(self):
14        if not self.is_empty():
15            return heapq.heappop(self.heap)[1]
16
17    def get_front(self):
18        if not self.is_empty():
19            return self.heap[0][1]
20
21    def size(self):
22        return len(self.heap)
23
24    # Test PriorityQueue with task handling visualization
25    priority_queue = PriorityQueue()
26
27    # Enqueue tasks with priorities
28    priority_queue.enqueue("Task 1", 3)
29    priority_queue.enqueue("Task 2", 1)
30    priority_queue.enqueue("Task 3", 2)
31    priority_queue.enqueue("Task 4", 5)
32    priority_queue.enqueue("Task 5", 4)
33    priority_queue.enqueue("Task 6", 2)
34
35    # Dequeue and visualize task handling
36    while not priority_queue.is_empty():
37        print(f"Handling: {priority_queue.dequeue()} Remaining tasks: {prior
38

```

```

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]: 1 class CircularQueue:
2         def __init__(self, capacity):
3             self.queue = [None] * capacity
4             self.front = self.rear = -1
5             self.capacity = capacity
6
7         def is_empty(self):
8             return self.front == -1
9
10        def is_full(self):
11            return (self.rear + 1) % self.capacity == self.front
12
13        def enqueue(self, item):
14            if self.is_full():
15                return
16            if self.is_empty():
17                self.front = self.rear = 0
18            else:
19                self.rear = (self.rear + 1) % self.capacity
20            self.queue[self.rear] = item
21
22        def dequeue(self):
23            if self.is_empty():
24                return None
25            removed_item = self.queue[self.front]
26            if self.front == self.rear:
27                self.front = self.rear = -1
28            else:
29                self.front = (self.front + 1) % self.capacity
30            return removed_item
31
32        def get_front(self):
33            if not self.is_empty():
34                return self.queue[self.front]
35
36        def size(self):
37            if self.is_empty():
38                return 0
39            elif self.front <= self.rear:
40                return self.rear - self.front + 1
41            else:
42                return self.capacity - self.front + self.rear + 1
43
44        # Test CircularQueue
45        circular_queue = CircularQueue(3)
46        circular_queue.enqueue(1)
47        circular_queue.enqueue(2)
48        circular_queue.enqueue(3)
49
50        print(circular_queue.get_front()) # Output: 1
51        print(circular_queue.dequeue())  # Output: 1
52        print(circular_queue.size())     # Output: 2
53
```

1
1
2