Stack implementation using an array

In [72]:

```
class Stack():
    def __init__(self,cap):
        self.cap=cap
        self.stack=[]
        self.top=-1
    def is_empty(self):
        return self.top==-1
    def is_full(self):
        return self.top==(self.cap)-1
    def push(self,item):
        if self.is_full():
            return print("STACK overflow")
        else:
            self.top+=1
            self.stack.append(item)
            print("Pushed: ",item)
            self.print_stack()
        return item
    def pop(self):
        if self.is_empty():
            print("Stack underflow. Cannot pop element.")
            return None
        else:
            item = self.stack.pop()
            self.top -= 1
            print("Popped:", item)
            self.print_stack()
            return item
    def peek(self):
        if self.is_empty():
            print("The stack is empty !")
            return None
        else:
            return self.stack[-1]
    def size(self):
        return self.top+1
    def print_stack(self):
        print('current stack',self.stack)
stack_cap=int(input("Enter the maximum capacity that your stack can have ")
s=Stack(stack_cap)
while True:
    print('1. Push')
    print("2. Pop")
    print("3. Peek")
    print('4. size')
    print('5. Quit')
    ch=int(input("Enter the number 1-5 only"))
    if(ch==1):
        item=input("Add the element to be pushed")
        s.push(item)
    elif(ch==2):
        s.pop()
        if s.pop() is not None:
            print("The popped item is: ",s.pop())
    elif(ch==3):
        s.peek()
        if s.peek() is not None:
            print("The peeked item is: ",s.peek())
    elif(ch==4):
        print(f'The size of the stack',s.size())
    elif(ch==5):
        print("The program Exited: ")
        break
```

```
Enter the maximum capacity that your stack can have 5
1. Push
2. Pop
3. Peek
4. size
5. Quit
Enter the number 1-5 only1
Add the element to be pushed34
Pushed:  34
current stack ['34']
1. Push
2. Pop
3. Peek
4. size
5. Quit
Enter the number 1-5 only2
Popped: 34
current stack []
Stack underflow. Cannot pop element.
1. Push
2. Pop
3. Peek
4. size
5. Quit
Enter the number 1-5 only3
The stack is empty !
The stack is empty !
1. Push
2. Pop
3. Peek
4. size
5. Quit
Enter the number 1-5 only4
The size of the stack 0
1. Push
2. Pop
3. Peek
4. size
5. Quit
Enter the number 1-5 only5
The program Exited: 
```

Stack implementation using an Linked list

In [114]:

```
class IsEmptyException(Exception):
    pass
class SllStack:

    class Node:
        def __init__(self,element,_next):
            self.element=element
            self._next=_next
    def __init__(self):
        self.head=None
        self.size=0
    def __len__(self):
        return self.size
    def is_empty(self):
        return self.size==0
    def push(self,element):
        self.head=self.Node(element,self.head)
        self.size+=1
    def pop(self):
        if self.is_empty():
            raise IsEmptyException("the stack is empty and cannot pop more!
        result=self.head.element
        self.head=self.head._next
        self.size-=1
        return result
    def peek(self):
        if self.is_empty():
            raise IsEmptyException("the stack is empty and cannot pop more!
        return self.head.element
```

In [115]:

```
s1=SllStack()
```

In [116]:

```
for i in range(1,1000):
    s1.push(i)
```

In [117]:

```
print(f'Popped element: {s1.pop()}')
print(f'Peeked element:{s1.peek()}')
```

```
Popped element: 999
Peeked element:998
```

Queue implementation using array

In [118]:

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

    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():
            print("Queue overflow. Cannot enqueue element:", item)
        else:
            if self.is_empty():
                self.front = self.rear = 0
            else:
                self.rear = (self.rear + 1) % self.capacity
            self.queue[self.rear] = item
            print("Enqueued:", item)
            self.print_queue()

    def dequeue(self):
        if self.is_empty():
            print("Queue underflow. Cannot dequeue element.")
            return None
        else:
            item = self.queue[self.front]
            if self.front == self.rear:
                self.front = self.rear = -1
            else:
                self.front = (self.front + 1) % self.capacity
            print("Dequeued:", item)
            self.print_queue()
            return item

    def front_element(self):
        if self.is_empty():
            print("Queue is empty.")
            return None
        else:
            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

    def print_queue(self):
        if self.is_empty():
            print("Current queue: Empty")
        else:
            print("Current queue:", end=" ")
            if self.front <= self.rear:
                for i in range(self.front, self.rear + 1):
                    print(self.queue[i], end=" ")
            else:
                for i in range(self.front, self.capacity):
                    print(self.queue[i], end=" ")
                for i in range(0, self.rear + 1):
                    print(self.queue[i], end=" ")
            print()


# Example usage with user input:
queue_capacity = int(input("Enter the capacity of the queue: "))
queue = Queue(queue_capacity)

while True:
    print("\n1. Enqueue")
    print("2. Dequeue")
    print("3. Front element")
    print("4. Size")
    print("5. Quit")

    choice = input("Enter your choice (1-5): ")

    if choice == "1":
        item = input("Enter the element to enqueue: ")
        queue.enqueue(item)
    elif choice == "2":
        dequeued_item = queue.dequeue()
        if dequeued_item is not None:
            print("Dequeued item:", dequeued_item)
    elif choice == "3":
        front = queue.front_element()
        if front is not None:
            print("Front element:", front)
    elif choice == "4":
        print("Queue size:", queue.size())
    elif choice == "5":
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please enter a valid option.")
```

```
Enter the capacity of the queue: 100

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 1
Enter the element to enqueue: 12
Enqueued: 12
Current queue: 12

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 1
Enter the element to enqueue: 2345
Enqueued: 2345
Current queue: 12 2345

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 3
Front element: 12

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 4
Queue size: 2

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 5
Exiting the program.
```

Queue using Linked List

In [119]:

```
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
        print("Enqueued:", item)
        self.print_queue()

    def dequeue(self):
        if self.is_empty():
            print("Queue underflow. Cannot dequeue element.")
            return None
        else:
            item = self.front.data
            if self.front == self.rear:
                self.front = self.rear = None
            else:
                self.front = self.front.next
            print("Dequeued:", item)
            self.print_queue()
            return item

    def front_element(self):
        if self.is_empty():
            print("Queue is empty.")
            return None
        else:
            return self.front.data

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

    def print_queue(self):
        if self.is_empty():
            print("Current queue: Empty")
        else:
            print("Current queue:", end=" ")
            current = self.front
            while current:
                print(current.data, end=" ")
                current = current.next
            print()


# Example usage with user input:
queue = QueueLinkedList()

while True:
    print("\n1. Enqueue")
    print("2. Dequeue")
    print("3. Front element")
    print("4. Size")
    print("5. Quit")

    choice = input("Enter your choice (1-5): ")

    if choice == "1":
        item = input("Enter the element to enqueue: ")
        queue.enqueue(item)
    elif choice == "2":
        dequeued_item = queue.dequeue()
        if dequeued_item is not None:
            print("Dequeued item:", dequeued_item)
    elif choice == "3":
        front = queue.front_element()
        if front is not None:
            print("Front element:", front)
    elif choice == "4":
        print("Queue size:", queue.size())
    elif choice == "5":
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please enter a valid option.")
```

```
1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 1
Enter the element to enqueue: 12
Enqueued: 12
Current queue: 12

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 54
Invalid choice. Please enter a valid option.

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 1
Enter the element to enqueue: 23456
Enqueued: 23456
Current queue: 12 23456

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 12
Invalid choice. Please enter a valid option.

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 1
Enter the element to enqueue: 324
Enqueued: 324
Current queue: 12 23456 324

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 2
Dequeued: 12
Current queue: 23456 324
Dequeued item: 12

1. Enqueue
2. Dequeue
3. Front element
4. Size
```

```
5. Quit
Enter your choice (1-5): 32
Invalid choice. Please enter a valid option.

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 2
Dequeued: 23456
Current queue: 324
Dequeued item: 23456

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 3
Front element: 324

1. Enqueue
2. Dequeue
3. Front element
4. Size
5. Quit
Enter your choice (1-5): 5
Exiting the program.
```

Priority Queue

In [123]:

```
import heapq

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

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

    def enqueue(self, item, priority):
        heapq.heappush(self.queue, (priority, item))
        print(f"Enqueued: {item} with priority {priority}")
        self.print_queue()

    def dequeue(self):
        if not self.is_empty():
            priority, item = heapq.heappop(self.queue)
            print(f"Dequeued: {item} with priority {priority}")
            self.print_queue()
            return item
        else:
            print("Priority queue is empty.")
            return None

    def print_queue(self):
        print("Current priority queue:", self.queue)

priority_queue = PriorityQueue()

priority_queue.enqueue("Task 1", 2)
priority_queue.enqueue("Task 2", 1)
priority_queue.enqueue("Task 3", 3)
print("\n")
print(priority_queue.dequeue())
print(priority_queue.dequeue())
```

```
Enqueued: Task 1 with priority 2
Current priority queue: [(2, 'Task 1')]
Enqueued: Task 2 with priority 1
Current priority queue: [(1, 'Task 2'), (2, 'Task 1')]
Enqueued: Task 3 with priority 3
Current priority queue: [(1, 'Task 2'), (2, 'Task 1'), (3, 'Task 3')]


Dequeued: Task 2 with priority 1
Current priority queue: [(2, 'Task 1'), (3, 'Task 3')]
Task 2
Dequeued: Task 1 with priority 2
Current priority queue: [(3, 'Task 3')]
Task 1
```

Circular Queue

In [124]:

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

    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():
            print("Circular queue overflow. Cannot enqueue element:", item)
        else:
            if self.is_empty():
                self.front = self.rear = 0
            else:
                self.rear = (self.rear + 1) % self.capacity
            self.queue[self.rear] = item
            print("Enqueued:", item)
            self.print_queue()

    def dequeue(self):
        if self.is_empty():
            print("Circular queue underflow. Cannot dequeue element.")
            return None
        else:
            item = self.queue[self.front]
            if self.front == self.rear:
                self.front = self.rear = -1
            else:
                self.front = (self.front + 1) % self.capacity
            print("Dequeued:", item)
            self.print_queue()
            return item

    def print_queue(self):
        if self.is_empty():
            print("Current circular queue: Empty")
        else:
            print("Current circular queue:", end=" ")
            if self.front <= self.rear:
                for i in range(self.front, self.rear + 1):
                    print(self.queue[i], end=" ")
            else:
                for i in range(self.front, self.capacity):
                    print(self.queue[i], end=" ")
                for i in range(0, self.rear + 1):
                    print(self.queue[i], end=" ")
            print()
circular_queue = CircularQueue(5)

circular_queue.enqueue(1)
circular_queue.enqueue(2)
circular_queue.enqueue(3)
circular_queue.enqueue(4)

circular_queue.dequeue()
circular_queue.dequeue()
```

```
Enqueued: 1
Current circular queue: 1
Enqueued: 2
Current circular queue: 1 2
Enqueued: 3
Current circular queue: 1 2 3
Enqueued: 4
Current circular queue: 1 2 3 4
Dequeued: 1
Current circular queue: 2 3 4
Dequeued: 2
Current circular queue: 3 4
```

Out[124]: 2

In [ ]: