

DATA STRUCTURE PRACTICAL ASSIGNMENT 07

Suryakant Upadhyay

20220802043

LAB-07

BATCH-A1

SEM-II

1) Write a python code to perform following operation on circular linked list.

a)Creation() b)Traversal()

a) Creation()

```
In [17]: 1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.next = None
5 class CircularLinkedList:
6     def __init__(self):
7         self.last = None
8
9     def addToEmpty(self, data):
10        if (self.last != None):
11            return self.last
12
13        temp = Node(data)
14        self.last = temp
15
16        self.last.next = self.last
17        return self.last
18
19    def addBegin(self, data):
20        if (self.last == None):
21            return self.addToEmpty(data)
22
23        temp = Node(data)
24        temp.next = self.last.next
25        self.last.next = temp
26        return self.last
27
28    def addEnd(self, data):
29        if (self.last == None):
30            return self.addToEmpty(data)
31
32        temp = Node(data)
33        temp.next = self.last.next
34        self.last.next = temp
35        self.last = temp
36        return self.last
37
38    def addAfter(self, data, item):
39        if (self.last == None):
40            return None
41
42        temp = Node(data)
43        p = self.last.next
44        while p:
45            if (p.data == item):
46                temp.next = p.next
47                p.next = temp
48
49            if (p == self.last):
50                self.last = temp
51                return self.last
52            else:
53                return self.last
54        p = p.next
55        if (p == self.last.next):
56            break
57
58        print(item, " not present in the list.")
59
60    def traverse(self):
61        if (self.last == None):
62            print("List is empty")
63            return
64
65        temp = self.last.next
66        while temp:
67            print(temp.data, end=" ")
68            temp = temp.next
69            if temp == self.last.next:
70                break
71 # create a circular linked list
72 cllist = CircularLinkedList()
73
74 # add nodes to the list
75 cllist.addToEmpty(1)
76 cllist.addBegin(2)
77 cllist.addEnd(3)
78 cllist.addAfter(4, 3)
79
80 # traverse the list
81 cllist.traverse()
82
2 1 3 4
```

b) Traversal()

```
In [19]: 1 # Node class to represent a single node in the circular linked list
2 class Node:
3     def __init__(self, data=None):
4         self.data = data
5         self.next = None
6
7 # Circular linked list class
8 class CircularLinkedList:
9     def __init__(self):
10        self.head = None
11
12 # Function to traverse and print the circular linked list
13 def traversal(self):
14     if self.head is None:
15         print("Circular linked list is empty!")
16     else:
17         current = self.head
18         print("Nodes of the Circular Linked List: ")
19         # Traverse the linked list until we reach the head node again
20         while True:
21             print(current.data, end=' ')
22             current = current.next
23             if current == self.head:
24                 break
25         print()
26
27 # Example usage
28 if __name__ == '__main__':
29     cll = CircularLinkedList()
30     cll.head = Node(1)
31     second = Node(2)
32     third = Node(3)
33     fourth = Node(4)
34
35 # Linking the nodes in a circular manner
36 cll.head.next = second
37 second.next = third
38 third.next = fourth
39 fourth.next = cll.head
40
41 # Traversing the circular linked list
42 cll.traversal()
43
Nodes of the Circular Linked List:
1 2 3 4
```

2) Write a python code to implement stack to perform following operation - push,pop,peek(static and dynamic implementation).

a) Static Implementation.

```
In [22]: 1 class Stack:
2     def __init__(self, max_size):
3         self.max_size = max_size
4         self.stack = [None] * max_size
5         self.top = -1
6
7     def push(self, data):
8         if self.top == self.max_size - 1:
9             print("Stack Overflow!")
10        else:
11            self.top += 1
12            self.stack[self.top] = data
13
14    def pop(self):
15        if self.top == -1:
16            print("Stack Underflow!")
17            return None
18        else:
19            data = self.stack[self.top]
20            self.top -= 1
21            return data
22
23    def peek(self):
24        if self.top == -1:
25            print("Stack is empty!")
26            return None
27        else:
28            return self.stack[self.top]
29
30
31    # Example usage
32    if __name__ == '__main__':
33        s = Stack(5)
34        s.push(1)
35        s.push(2)
36        s.push(3)
37        print(s.peek())    # Output: 3
38        s.pop()
39        s.pop()
40        s.pop()
41        print(s.pop())    # Output: Stack Underflow!
42
```

3  
Stack Underflow!  
None

b) Dynamic Implementation.

```
In [24]: 1 class Stack:
2     def __init__(self):
3         self.stack = []
4         self.top = -1
5
6     def push(self, data):
7         self.stack.append(data)
8         self.top += 1
9
10    def pop(self):
11        if self.top == -1:
12            print("Stack Underflow!")
13            return None
14        else:
15            data = self.stack.pop()
16            self.top -= 1
17            return data
18
19    def peek(self):
20        if self.top == -1:
21            print("Stack is empty!")
22            return None
23        else:
24            return self.stack[self.top]
25
26
27    # Example usage
28    if __name__ == '__main__':
29        s = Stack()
30        s.push(1)
31        s.push(2)
32        s.push(3)
33        print(s.peek())    # Output: 3
34        s.pop()
35        s.pop()
36        s.pop()
37        print(s.pop())    # Output: Stack Underflow!
38
```

3  
Stack Underflow!  
None

3) Write a python code to implement queue to perform following operation - enqueue,dequeue,peek(static and dynamic implementation).

a) Static Implementation.

```
In [26]: 1 class Queue:
2     def __init__(self, max_size):
3         self.max_size = max_size
4         self.queue = [None] * max_size
5         self.front = -1
6         self.rear = -1
7
8     def enqueue(self, data):
9         if self.rear == self.max_size - 1:
10            print("Queue Overflow!")
11        elif self.front == -1 and self.rear == -1:
12            self.front = 0
13            self.rear = 0
14            self.queue[self.rear] = data
15        else:
16            self.rear += 1
17            self.queue[self.rear] = data
18
19    def dequeue(self):
20        if self.front == -1 or self.front > self.rear:
21            print("Queue Underflow!")
22            return None
23        else:
24            data = self.queue[self.front]
25            self.front += 1
26            return data
27
28    def peek(self):
29        if self.front == -1 or self.front > self.rear:
30            print("Queue is empty!")
31            return None
32        else:
33            return self.queue[self.front]
34
35
36    # Example usage
37    if __name__ == '__main__':
38        q = Queue(5)
39        q.enqueue(1)
40        q.enqueue(2)
41        q.enqueue(3)
42        print(q.peek())    # Output: 1
43        q.dequeue()
44        q.dequeue()
45        q.dequeue()
46        print(q.dequeue())    # Output: Queue Underflow!
47
```

1  
Queue Underflow!  
None

b) Dynamic Implementation.

In [27]:

```
1 class Queue:
2     def __init__(self):
3         self.queue = []
4         self.front = -1
5         self.rear = -1
6
7     def enqueue(self, data):
8         self.queue.append(data)
9         if self.front == -1 and self.rear == -1:
10             self.front = 0
11             self.rear = 0
12         else:
13             self.rear += 1
14
15     def dequeue(self):
16         if self.front == -1 or self.front > self.rear:
17             print("Queue Underflow!")
18             return None
19         else:
20             data = self.queue[self.front]
21             self.front += 1
22             return data
23
24     def peek(self):
25         if self.front == -1 or self.front > self.rear:
26             print("Queue is empty!")
27             return None
28         else:
29             return self.queue[self.front]
30
31
32 # Example usage
33 if __name__ == '__main__':
34     q = Queue()
35     q.enqueue(1)
36     q.enqueue(2)
37     q.enqueue(3)
38     print(q.peek())    # Output: 1
39     q.dequeue()
40     q.dequeue()
41     q.dequeue()
42     print(q.dequeue()) # Output: Queue Underflow!
43
```

1
Queue Underflow!
None