



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

## TERM PROJECT 1

```
1  from DeQueue import DeQueue
2
3  class dequeUsingStackAndQueue:  2 usages
4      def __init__(self):
5          self.stack = []
6          self.queue = DeQueue()
7
8      def __len__(self):
9          return len(self.stack) + len(self.queue)
10
11     def is_empty(self):  11 usages (6 dynamic)
12         return not self.stack and self.queue.is_empty()
13
14     def add_first(self, e):  2 usages
15         self.stack.append(e)
16
17     def add_last(self, e):  3 usages
18         self.queue.add_last(e)
19
20     def delete_first(self):  2 usages
21         if self.is_empty():
22             raise Exception("Deque is empty")
23
24         if not self.stack:
25             while not self.queue.is_empty():
```



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
26         self.stack.append(self.queue.delete_first())
27
28     return self.stack.pop()
29
30     def delete_last(self):
31         if self.is_empty():
32             raise Exception("Deque is empty")
33
34         if self.queue.is_empty():
35             while self.stack:
36                 self.queue.add_first(self.stack.pop())
37
38         return self.queue.delete_last()
39
40     def first(self): 1 usage
41         if self.is_empty():
42             raise Exception("Deque is empty")
43
44         if self.stack:
45             return self.stack[-1]
46
```

```
44         if self.stack:
45             return self.stack[-1]
46
47         return self.queue.first()
48
49     def last(self): 1 usage
50         if self.is_empty():
51             raise Exception("Deque is empty")
52
53         if self.queue.is_empty():
54             return self.stack[0]
55
56         return self.queue.last()
```



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
1  class dequeUsingStack: 2 usages
2  def __init__(self):
3      self.S_in = []
4      self.S_out = []
5
6  def __len__(self):
7      return len(self.S_in) + len(self.S_out)
8
9  def is_empty(self): 11 usages (6 dynamic)
10     return not self.S_in and not self.S_out
11
12 def display(self):
13     return self.S_in + self.S_out[::-1]
14
15 def add_first(self, e): 2 usages
16     self.S_in.append(e)
17
18 def add_last(self, e): 3 usages
19     self.S_out.append(e)
20
21 def delete_first(self): 2 usages
22     if self.is_empty():
23         raise Exception("Deque is empty")
24
25     if not self.S_in:
26         while self.S_out:
```



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
1  class dequeUsingStack: 2 usages
21      def delete_first(self): 2 usages
27          self.S_in.append(self.S_out.pop())
28
29          return self.S_in.pop()
30
31      def delete_last(self):
32          if self.is_empty():
33              raise Exception("Deque is empty")
34
35          if not self.S_out:
36              while self.S_in:
37                  self.S_out.append(self.S_in.pop())
38
39          return self.S_out.pop()
40
41      def first(self): 1 usage
42          if self.is_empty():
43              raise Exception("Deque is empty")
44
45          if not self.S_in:
46              while self.S_out:
47                  self.S_in.append(self.S_out.pop())
48
49          return self.S_in[-1]
```



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
41     def first(self): 1 usage
42         if self.is_empty():
43             raise Exception("Deque is empty")
44
45         if not self.S_in:
46             while self.S_out:
47                 self.S_in.append(self.S_out.pop())
48
49         return self.S_in[-1]
50
51     def last(self): 1 usage
52         if self.is_empty():
53             raise Exception("Deque is empty")
54
55         if not self.S_out:
56             while self.S_in:
57                 self.S_out.append(self.S_in.pop())
58
59         return self.S_out[-1]
```



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
1  class DeQueue: 3 usages
2      DEFAULT_CAPACITY = 8
3      def __init__(self):
4          self._data = [None] * DeQueue.DEFAULT_CAPACITY
5          self._size = 0
6          self._front = 0
7
8      def __len__(self):
9          return self._size
10
11     def display(self):
12         return self._data
13
14     def is_empty(self): 14 usages (6 dynamic)
15         return self._size == 0
16
17     def first(self): 1 usage
18         if self.is_empty():
19             raise Exception('Queue is empty')
20         return self._data[self._front]
21
22     def last(self): 1 usage
23         if self.is_empty():
24             raise Exception('Queue is empty')
25         back = (self._front + self._size - 1) % len(self._data)
```

SECTION:IDB2

DATE SUBMITTED:12/062024



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
25         back = (self._front + self._size - 1) % len(self._data)
26         return self._data[back]
27
28     def delete_first(self): 1 usage
29         if self.is_empty():
30             raise Exception('Queue is empty')
31         answer = self._data[self._front]
32         self._data[self._front] = None
33         self._front = (self._front + 1) % len(self._data)
34         self._size -= 1
35         return answer
36
37     def delete_last(self): 1 usage
38         if self.is_empty():
39             raise Exception("Queue is empty")
40         answer = self._data[(self._front + self._size) - 1 % len(self._data)]
41         self._data[(self._front + self._size) - 1 % len(self._data)] = None
42         self._size -= 1
43         return answer
44
45     def add_first(self, e): 1 usage
46         if self._size == len(self._data):
47             self._resize(2 * len(self._data))
48         self._front = (self._front - 1) % len(self._data)
```





UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
43         return answer
44
45     def add_first(self, e): 1 usage
46         if self._size == len(self._data):
47             self._resize(2 * len(self._data))
48         self._front = (self._front - 1) % len(self._data)
49         self._data[self._front] = e
50         self._size += 1
51
52     def add_last(self, e): 1 usage
53         if (self._size == len(self._data)):
54             self._resize(2 * len(self._data))
55         avail = (self._front + self._size) % len(self._data)
56         self._data[avail] = e
57         self._size += 1
58
59     def _resize(self, cap): 2 usages
60         old = self._data
61         self._data = [None] * cap
62         walk = self._front
63         for k in range(self._size):
64             self._data[k] = old[walk]
65             walk = (1 + walk) % len(old)
66         self._front = 0
```





UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

4 usages

1 **class** LinkedStack:

2 *'''LIFO Stack implementation using a singly linked list for storage.'''*

3 *#----- nested \_Node class -----*

4 **class** \_Node:

5 *'''Lightweight non public class for storing a singly linked node.'''*

6 *\_\_slots\_\_ = '\_element', '\_next' #streamline memory usage*

7 **def** \_\_init\_\_(self, element, next):

8 *self.\_element = element*

9 *self.\_next = next*

10 *#----- stack methods -----*

11 **def** \_\_init\_\_(self):

12 *'''Create an empty Stack'''*

13 *self.\_head = None*

14 *self.\_size = 0*

15 **def** \_\_len\_\_(self):

16 *'''Return the number of elements in the stack'''*

17 *return self.\_size*

18 19 usages (6 dynamic)

19 **def** is\_empty(self):

20 *'''Return True if the stack is empty.'''*

21 *return self.\_size == 0*

22 9 usages (2 dynamic)

23 **def** push(self, e):

24 *'''Add element e to the top of the stack.'''*

25 *self.\_head = self.\_Node(e, self.\_head)*

26 *self.\_size += 1*

27 1 usage

28 **def** top(self):

29 *'''Return but do not remove the element at the top of the stack'''*

30 *'''Raise empty exception if the stack is empty!'''*

31 *if self.is\_empty():*

32 *raise Exception('Stack is empty')*

33 *return self.\_head.\_element #top of the stack is the head of the list*



```
35     def pop(self):
36         '''Remove and return the elements from the top of the stack (LIFO)'''
37         '''Raise Empty exception if the stack is empty!'''
38         if self.is_empty():
39             raise Exception("The stack is empty!")
40         answer = self._head._element
41         self._head = self._head._next
42         self._size -= 1
43         return answer
44
45     class DequeUsingTwoStacks:
46         def __init__(self):
47             self.stack1 = LinkedStack()
48             self.stack2 = LinkedStack()
49
50             6 usages (6 dynamic)
51         def is_empty(self):
52             return self.stack1.is_empty() and self.stack2.is_empty()
53
54         def add_to_front(self, item):
55             self.stack1.push(item)
56
57         def add_to_rear(self, item):
58             self.stack2.push(item)
59
60         def remove_from_front(self):
61             if self.stack1.is_empty():
62                 while not self.stack2.is_empty():
63                     self.stack1.push(self.stack2.pop())
64             return self.stack1.pop()
65
66         def remove_from_rear(self):
67             if self.stack2.is_empty():
68                 while not self.stack1.is_empty():
69                     self.stack2.push(self.stack1.pop())
70             return self.stack2.pop()
```



```
1  class LinkedQueue:
2      '''FIFO queue implementation using a singly linked list for storage.'''
3
4      #----- nested _Node class -----
5      class _Node:
6          '''Lightweight non public class for storing a singly linked node.'''
7          __slots__ = '_element', '_next' # streamline memory usage
8
9          def __init__(self, element, next):
10             self._element = element
11             self._next = next
12
13         #----- queue methods -----
14         def __init__(self):
15             '''Create an empty queue'''
16             self._head = None
17             self._tail = None
18             self._size = 0
19
20         def __len__(self):
21             '''Return the number of elements in the queue'''
22             return self._size
23
24         15 usages (6 dynamic)
25         def is_empty(self):
26             '''Return true if the queue is empty.'''
27             return self._size == 0
28
29         1 usage
30         def first(self):
31             '''Return but do not remove the element at the front of the queue'''
32             if self.is_empty():
33                 raise Exception('Queue is empty')
34             return self._head._element #front aligned with the head of the list
35
36         5 usages (2 dynamic)
37         def dequeue(self):
38             '''Remove and return the first element of the queue (FIFO)'''
39             '''Raise empty exception if the queue is empty'''
40             if self.is_empty():
```



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
37         answer = self._head._element
38         self._head = self._head.next
39         self._size -= 1
40         if self.is_empty():#special case as queue is empty
41             self._tail = None#removed head had been the tail
42         return answer
43     5 usages (2 dynamic)
44     def enqueue(self, e):
45         '''Add an element to the back of queue.'''
46         newest = self._Node(e, next=None)#node will be new tail node
47         if self.is_empty():
48             self._head = newest#special case: previously empty
49         else:
50             self._tail._next = newest
51             self._tail = newest#update reference to tail node
52             self._size += 1
53
54     class DequeueUsingStackAndQueue:
55     def __init__(self):
56         self.stack = linkedStack()
57         self.queue = linkedQueue()
58
59     6 usages (6 dynamic)
60     def is_empty(self):
61         return self.stack.is_empty() and self.queue.is_empty()
62
63     def add_to_front(self, item):
64         self.stack.push(item)
65
66     def add_to_rear(self, item):
67         self.queue.enqueue(item)
68
69     def remove_from_front(self):
70         if self.stack.is_empty():
71             while not self.queue.is_empty():
72                 self.stack.push(self.queue.dequeue())
73             return self.stack.pop()
74
75     def remove_from_rear(self):
76         if self.queue.is_empty():
77             while not self.stack.is_empty():
78                 self.queue.enqueue(self.stack.pop())
79             return self.queue.dequeue()
```



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
1  from LinkedList import LinkedList as LinkedList
2  from LinkedStack import LinkedStack as LinkedStack
3
4  2 usages
5  class DLS:
6      def __init__(self):
7          # Create a stack for the front and a queue for the back
8          self.stack = LinkedStack()
9          self.queue = LinkedList()
10
11     def __len__(self):
12         # Total number of elements in the deque is the sum of the stack and queue sizes
13         return len(self.stack) + len(self.queue)
14
15     11 usages (6 dynamic)
16     def is_empty(self):
17         # The deque is empty if both the stack and the queue are empty
18         return self.stack.is_empty() and self.queue.is_empty()
19
20     def display(self):
21         # Combine the front (from stack) and back (from queue) for display
22         stack_elements = []
23         current = self.stack._head # Traverse stack
24         while current is not None:
25             stack_elements.append(current._element)
26             current = current._next
27         stack_elements.reverse() # Reverse stack to get front-to-back order
28
29         queue_elements = []
30         current = self.queue._head # Traverse queue
31         while current is not None:
32             queue_elements.append(current._element)
33             current = current._next
34
35         return stack_elements + queue_elements # Display front to back
36
37     4 usages
38     def add_first(self, e):
```



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
37         self.stack.push(e)
38
39     1 usage
40     def add_last(self, e):
41         # Enqueue element to the queue (back of the deque)
42         self.queue.enqueue(e)
43
44     def delete_first(self):
45         # Remove and return the front element of the deque
46         if self.is_empty():
47             raise Exception("Deque is empty")
48
49         if self.stack.is_empty():
50             # If the stack is empty, transfer elements from the queue to the stack
51             while not self.queue.is_empty():
52                 self.stack.push(self.queue.dequeue())
53
54         # Now pop from the stack (front)
55         return self.stack.pop()
56
57     def delete_last(self):
58         # Remove and return the last element of the deque
59         if self.is_empty():
60             raise Exception("Deque is empty")
61
62         if self.queue.is_empty():
63             # If the queue is empty, transfer elements from the stack to the queue
64             while not self.stack.is_empty():
65                 self.queue.enqueue(self.stack.pop())
66
67         # Now dequeue from the queue (back)
68         return self.queue.dequeue()
69
70     1 usage
71     def first(self):
72         # Access the first element of the deque without removing it
73         if self.is_empty():
74             raise Exception("Deque is empty")
```



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
73
74     if self.stack.is_empty():
75         # If the stack is empty, transfer elements from the queue to the stack
76         while not self.queue.is_empty():
77             self.stack.push(self.queue.dequeue())
78
79     return self.stack.top() # Peek at the top of the stack (front of the deque)
80
81 1 usage
82 def last(self):
83     # Access the last element of the deque without removing it
84     if self.is_empty():
85         raise Exception("Deque is empty")
86
87     if self.queue.is_empty():
88         # If the queue is empty, transfer elements from the stack to the queue
89         while not self.stack.is_empty():
90             self.queue.enqueue(self.stack.pop())
91
92     return self.queue.first()
```





UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

```
1  from dequeUsingStack import dequeUsingStack as dequeUsingStack
2  from dequeUsingStackAndQueue import dequeUsingStackAndQueue as dequeUsingStackAndQueue
3  from DLS import DLS as DLS
4
5  double = DLS()
6  deque = dequeUsingStack()
7  D = dequeUsingStackAndQueue()
8
9  print("Deque using STACK:")
10 deque.add_first(1)
11 print("First number that was added on the stack is: ", deque.delete_first())
12 deque.add_last(2)
13 print("The number that was added on the last stack is: ", deque.delete_first())
14 deque.add_first(3)
15 deque.add_last(8)
16 deque.add_last(9)
17 print("The first number in the stack is: ", deque.first())
18 print("The last number in the stack is: ", deque.last())
19 print("Checking if the stack is empty: ", deque.is_empty())
20 print()
21
22
23 print("Deque using Stack and Queue:")
24 D.add_first(5)
25 print("First number that was added is: ", D.delete_first())
26 D.add_last(2)
27 print("The number that was added is: ", D.delete_first())
28 D.add_first(1)
29 D.add_last(7)
30 D.add_last(8)
31 print("The first number is: ", D.first())
32 print("The last number is: ", D.last())
33 print("Checking if the stack is empty: ", D.is_empty())
34 print()
35
36
37 print("Deque using LinkedStack And LinkedQueue:")
38 double.add_first(6)
38 double.add_first(6)
39 double.add_last(10)
40 double.add_first(16)
41 double.add_first(75)
42 double.add_first(21)
43 print("The first number is: ", double.first())
44 print("The last number is: ", double.last())
45 print("Checking if the stack is empty: ", double.is_empty())
46 print()
```



UNIVERSITY OF  
**Baguio**

SCHOOL OF INFORMATION TECHNOLOGY

NAME: MARK LL OYD YADAO /RICHMOND BROQUEZA

DSALGO1/IDB2

DATE SUBMITTED:12/062024

## OUTPUT

```
"C:\Program Files\Python312\python.exe" "Z:\DSALGO1-IDB2\TermProject1\main (1).py"
```

```
Deque using STACK:
```

```
First number that was added on the stack is: 1
```

```
The number that was added on the last stack is: 2
```

```
The first number in the stack is: 3
```

```
The last number in the stack is: 9
```

```
Checking if the stack is empty: False
```

```
Deque using Stack and Queue:
```

```
First number that was added is: 5
```

```
The number that was added is: 2
```

```
The first number is: 1
```

```
The last number is: 8
```

```
Checking if the stack is empty: False
```

```
Deque using LinkedStack And LinkedQueue:
```

```
The first number is: 21
```

```
The last number is: 10
```

```
Checking if the stack is empty: False
```