Implementing Stacks & Queues

Chapter No.	16
Status	Completed

▼ Implementing a Stack Using a Linked List

```
#Implementing a Stack Using a Linked List
class Node:
    def __init__(self, data):
        self.data = data
        self.pointer = None #self.pointer should point to the next node

class Stack:
    def __init__(self):
        self.head = None

    def is_empty(self):
        if self.head == None:
            return True
        return False

    def size(self):
        count = 0
        current = self.head #current node - used to iterate through linked list
        while current:
            count += 1
            current = current.pointer
        return count

    def push(self, data):
        new_node = Node(data) #a new node containing the data that is to be pushed into the stack
        new_node.pointer = self.head
        self.head = new_node

    def pop(self):
        if self.head == None:
            print("Stack is empty!")
        else:
            to_return = self.head
            self.head = self.head.pointer
            return to_return.data

    def peek(self):
        if self.head == None:
            print("Stack is empty!")
        else:
            return self.head.data
```

▼ Implementing a Linear Queue Using a Linked List

```
#Implementing a Queue Using a Linked List
class Node:
    def __init__(self, data):
        self.data = data
        self.pointer = None #self.pointer should point to the next node

class Queue:
    def __init__(self):
        self.head = None
        self.tail = None

    def is_empty(self):
        if self.head == None:
            return True
        return False

    def size(self):
        count = 0
        current = self.head #current node - used to iterate through linked list
        while current:
            count += 1
            current = current.pointer
        return count

    def enqueue(self, data):
        new_node = Node(data)
        if self.head == None:
            self.head = new_node
            self.tail = new_node
        else:
            self.tail.pointer = new_node
            self.tail = new_node

    def dequeue(self):
        if self.head == None:
            print("Queue is empty!")
        else:
            to_return = self.head
            self.head = self.head.pointer
            return to_return.data

    def qhead(self):
        if self.head == None:
            print("Queue is empty!")
        else:
            return self.head.data

    def qtail(self):
        if self.head == None:
            print("Queue is empty!")
        else:
            return self.tail.data
```

▼ Memory Allocation

- Static memory allocation is the allocation of memory before a program is run
- Dynamic memory allocation is the allocation of memory when it is required when a program is running

▼ Comparing Static & Dynamic Memory Allocation

Static VS Dynamic Memory Allocation

Aa Memory Allocation	≡ Advantages	≡ Disadvantages
Static	- Easier to implement - Quicker access to memory	- Not possible to change the amount of memory allocated after the initial allocation
Dynamic	- Efficient use of memory as only the required amount of memory is used	- Difficult to implement - Slower access to memory