**NAME:** **Meet Raut**

**DIV:** **S2-1**

**ROLL NO:** **2201084**

## EXPERIMENT – 14:

- ***AIM:*** **To study and implement Menu Driven program for data structure using built in function for Link List, Stack and Queue.**

- ***THEORY:***

Data structure organizes the storage in computers so that we can easily access and change data. Stacks and Queues are the earliest data structure defined in computer science. A simple Python list can act as a queue and stack as well. A queue follows FIFO rule (First In First Out) and used in programming for sorting. It is common for stacks and queues to be implemented with an array or linked list.

Python offers various ways to implement the stack. Such as

- **List**
- **Collection.deque**
- **LifeQueue**

Here, **Python List** can be used to implement the stack. For those following, in-build methods are used in the below program.

- **append() method -** To PUSH or insert elements into the stack.
- **pop() method -** To POP out or remove the element from the stack in LIFO order.
- **len(stack) -** To find out the size of the stack.

## LINK LIST:

In Python, a linked list can be implemented using a class, with each instance of the class representing a node in the list. Each node has a reference (pointer) to the next node in the list, as well as a value. The last node in the list has a reference to None, indicating the end of the list. Linked lists have advantages over arrays when it comes to inserting and deleting elements, as the elements do not have to be shifted in memory. However, they have slower random-access times.

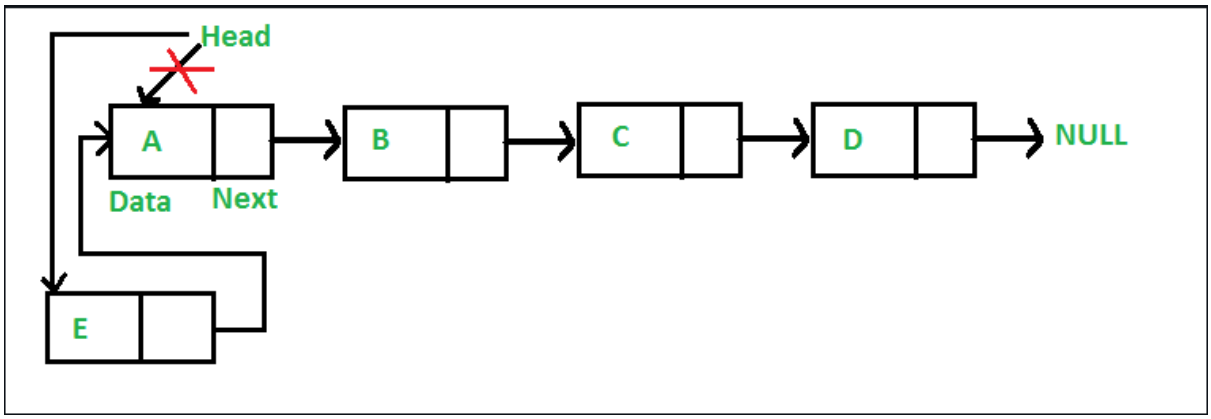
**Nodes**

These nodes generally have two attributes:

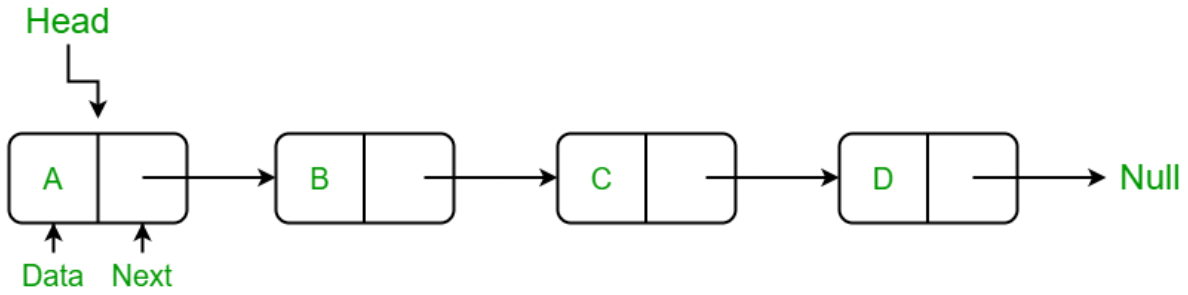
1. One that holds data. This data can be a character, string, stack, queue, another object, or another linked list. It can be anything.
2. A reference to another node, the "next" node, often in the form of the actual address in memory.
3. In doubly linked list, they also have a reference to the "previous" node.

The linked list

These nodes make up a Linked List which can be added to, or removed from




A linked list is a type of linear data structure similar to arrays. It is a collection of nodes that are linked with each other. A node contains two things first is data and second is a link that connects it with another node. Below is an example of a linked list with four nodes and each node contains character data and a link to another node. Our first node is where **head** points and we can access all the elements of the linked list using the **head.**




## Creating a linked list in Python

In this LinkedList class, we will use the Node class to create a linked list. In this class, we have an **__init__** method that initializes the linked list with an empty head. Next, we have created an **insertAtBegin()** method to insert a node at the beginning of the linked list, an **insertAtIndex()** method to insert a node at the given index of the linked list,

and **insertAtEnd()** method inserts a node at the end of the linked list. After that, we have the **remove_node()** method which takes the data as an argument to delete that node. In the **remove_node()** method we traverse the linked list if a node is present equal to data then we delete that node from the linked list. Then we have the **sizeOfLL()** method to get the current size of the linked list and the last method of the LinkedList class is **printLL()** which traverses the linked list and prints the data of each node.

## Creating a Node Class

We have created a Node class in which we have defined a **__init__** function to initialize the node with the data passed as an argument and a reference with None because if we have only one node then there is nothing in its reference.

## Insertion in Linked List:

### Insertion at Beginning in Linked List

This method inserts the node at the beginning of the linked list. In this method, we create a **new_node** with the given **data** and check if the head is an empty node or not if the head is empty then we make the **new_node** as **head** and **return** else we insert the head at the next **new_node** and make the **head** equal to **new_node.**

### Insert a Node at a Specific Position in a Linked List

This method inserts the node at the given index in the linked list. In this method, we create a **new_node** with given data , a current_node that equals to the head, and a counter **'position'** initializes with **0.** Now, if the index is equal to zero it means the node is to be inserted at begin so we called **insertAtBegin() method** else we run a while loop until the **current_node** is not equal to **None** or **(position+1)** is not equal to the index we have to at the one position back to insert at a given position to make the linking of nodes and in each iteration, we increment the position by 1 and make the **current_node** next of it. When the loop breaks and if **current_node** is not equal to **None** we insert new_node at after to the **current_node.** If **current_node** is equal to **None** it means that the index is not present in the list and we print **"Index not present".**

### Insertion in Linked List at End

This method inserts the node at the end of the linked list. In this method, we create a **new_node** with the given data and check if the **head** is an empty node or not if the **head** is empty then we make the **new_node** as **head** and return **else** we make a **current_node equal** to **the head** traverse to the last **node** of the linked list and when we get **None** after the current_node the while loop breaks and insert the **new_node** in the next of **current_node** which is the last node of linked list.

## Delete Node in a Linked List:

### Remove First Node from Linked List

This method removes the first node of the linked list simply by making the second node **head** of the linked list.

**Remove Last Node from Linked List**

In this method, we will delete the last node. First, we traverse to the second last node using the while loop, and then we make the next of that node **None** and last node will be removed.

**Delete a Linked List Node at a given Position**

In this method, we will remove the node at the given index, this method is similar to the **insert_at_inded()** method. In this method, if the **head** is **None** we simply **return** else we initialize a **current_node** with **self.head** and **position** with **0.** If the position is equal to the index we called the **remove_first_node()** method else we traverse to the one node before that we want to remove using the while loop. After that when we out of the while loop we check **that current_node** is equal to **None** if not then we make the next of current_node equal to the next of node that we want to remove else we print the message **"Index not present"** because **current_node** is equal to **None.**

**Delete a Linked List Node of a given Data**

This method removes the node with the given data from the linked list. In this method, firstly we made a **current_node** equal to the **head** and run a **while loop** to traverse the linked list. This while loop breaks when **current_node** becomes **None** or the data next to the current node is equal to the data given in the argument. Now, After coming out of the loop if the **current_node** is equal to **None** it means that the node is not present in the data and we just return, and if the data next to the **current_node** is equal to the data given then we remove that node by making next of that removed_node to the next of current_node. And this is implemented using the if else condition.

## Linked List Traversal in Python:

This method traverses the linked list and prints the data of each node. In this method, we made a **current_node** equal to the **head** and iterate through the linked list using a **while loop** until the **current_node** become None and print the data of **current_node** in each iteration and make the **current_node** next to it.

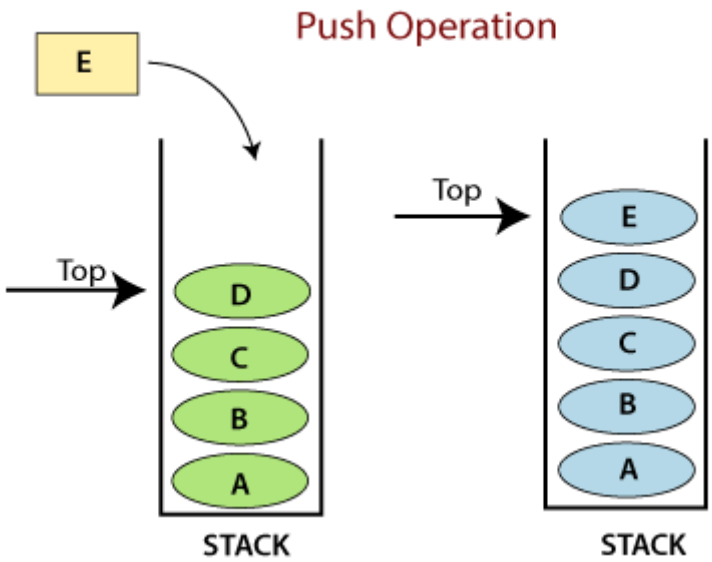
## Get Length of a Linked List in Python:

This method returns the size of the linked list. In this method, we have initialized a counter **'size'** with 0, and then if the head is not equal to None we traverse the linked list using a **while loop** and increment the **size** with 1 in each iteration and return the size when **current_node** becomes **None else** we return 0.

## Stack:

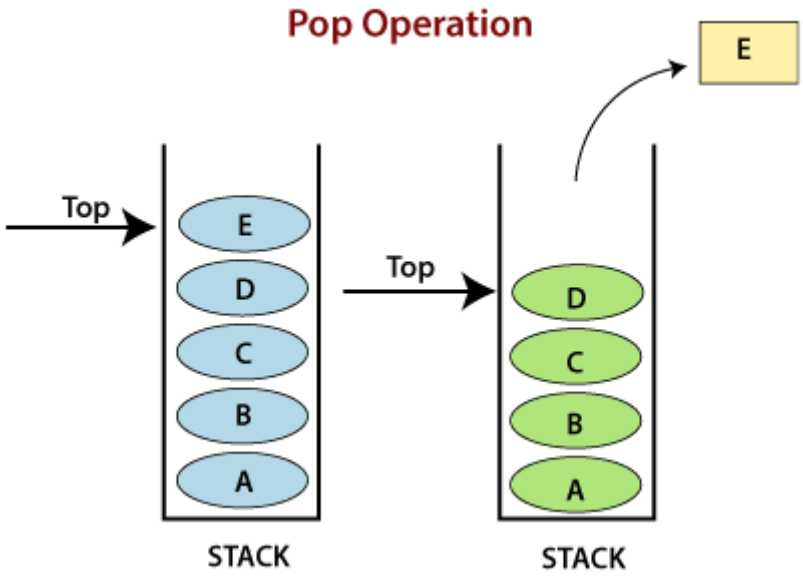
A Stack is a data structure that follows the LIFO(Last In First Out) principle. To implement a stack, we need two simple operations:

- **push -** It adds an element to the top of the stack.
- **pop -** It removes an element from the top of the stack.




**Operations:**

- **Adding -** It adds the items in the stack and increases the stack size. The addition takes place at the top of the stack.
- **Deletion -** It consists of two conditions, first, if no element is present in the stack, then underflow occurs in the stack, and second, if a stack contains some elements, then the topmost element gets removed. It reduces the stack size.
- **Traversing -** It involves visiting each element of the stack.




**Characteristics:**

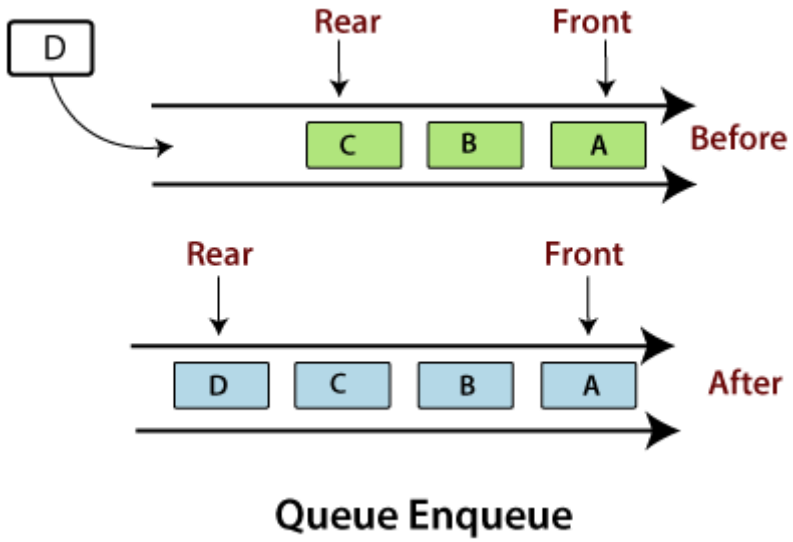
- Insertion order of the stack is preserved.
- Useful for parsing the operations.
- Duplicacy is allowed.

## **QUEUE**:

A Queue follows the First-in-First-Out (FIFO) principle. It is opened from both the ends hence we can easily add elements to the back and can remove elements from the front.

To implement a queue, we need two simple operations:

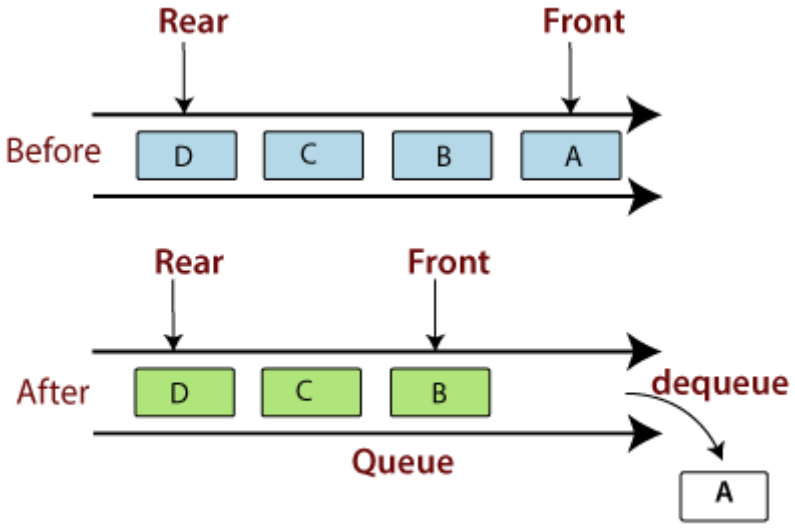
- **enqueue -** It adds an element to the end of the queue.
- **dequeue -** It removes the element from the beginning of the queue.




Queue Enqueue

**Operations on Queue**

- **Addition -** It adds the element in a queue and takes place at the rear end, i.e., at the back of the queue.
- **Deletion -** It consists of two conditions - If no element is present in the queue, Underflow occurs in the queue, or if a stack contains some elements then element present at the front gets deleted.
- **Traversing -** It involves to visit each element of the queue.




Queue Dequeue

### Characteristics

- Insertion order of the queue is preserved.
- Duplicacy is allowed.
- Useful for parsing CPU task operations.

Note: **The implementation of a queue is a little bit different. A queue follows the "First-In-First-Out". Time plays an important factor here. The Stack is fast because we insert and pop the elements from the end of the list, whereas in the queue, the insertion and pops are made from the beginning of the list, so it becomes slow. The cause of this time difference is due to the properties of the list, which is fast in the end operation but slow at the beginning operations because all other elements have to be shifted one by one.**

- **PROGRAM:**

```
from collections import deque

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    def insert_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
```

```
        self.head = new_node

    def insert_at_end(self, data):
        if not self.head:
            self.head = Node(data)
            return
        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = Node(data)

    def insert_at_position(self, data, position):
        if position == 0:
            self.insert_at_beginning(data)
            return
        new_node = Node(data)
        temp = self.head
        for _ in range(position - 1):
            if temp is None:
                print("Position out of range")
                return
            temp = temp.next
        if temp is None:
            print("Position out of range")
            return
        new_node.next = temp.next
        temp.next = new_node

    def delete_at_position(self, position):
```

```
    if self.head is None:
        print("Linked list is empty")
        return
    temp = self.head
    if position == 0:
        self.head = temp.next
        temp = None
        return
    for _ in range(position - 1):
        temp = temp.next
        if temp is None:
            break
    if temp is None or temp.next is None:
        print("Position out of range")
        return
    next_node = temp.next.next
    temp.next = None
    temp.next = next_node

def delete_at_beginning(self):
    if not self.head:
        print("Linked list is empty")
        return
    self.head = self.head.next

def delete_at_end(self):
    if not self.head:
        print("Linked list is empty")
        return
```

```
        if not self.head.next:
            self.head = None
            return
        temp = self.head
        while temp.next.next:
            temp = temp.next
        temp.next = None

    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" ")
            temp = temp.next
        print()

class Stack:
    def __init__(self):
        self.stack = []

    def push(self, data):
        self.stack.append(data)

    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        else:
            print("Stack is empty")

    def peek(self):
```

```
        if not self.is_empty():
            return self.stack[-1]
        else:
            print("Stack is empty")

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

class Queue:
    def __init__(self):
        self.queue = deque()

    def enqueue(self, data):
        self.queue.append(data)

    def dequeue(self):
        if not self.is_empty():
            return self.queue.popleft()
        else:
            print("Queue is empty")

    def view(self):
        return list(self.queue)

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

def linked_list_menu(ll):
    while True:
```

```
print("\nLinked List Menu:")
print("1. Insert at beginning")
print("2. Insert at end")
print("3. Insert at specific position")
print("4. Delete at specific position")
print("5. Delete at beginning")
print("6. Delete at end")
print("7. Display")
print("8. Back to main menu")
choice = int(input("Enter your choice: "))
if choice == 1:
    data = int(input("Enter data to insert at beginning: "))
    ll.insert_at_beginning(data)
elif choice == 2:
    data = int(input("Enter data to insert at end: "))
    ll.insert_at_end(data)
elif choice == 3:
    data = int(input("Enter data to insert: "))
    position = int(input("Enter position to insert at: "))
    ll.insert_at_position(data, position)
elif choice == 4:
    position = int(input("Enter position to delete: "))
    ll.delete_at_position(position)
elif choice == 5:
    ll.delete_at_beginning()
elif choice == 6:
    ll.delete_at_end()
elif choice == 7:
    print("Linked List:")
```

```
            ll.display()
        elif choice == 8:
            break
        else:
            print("Invalid choice")

def stack_menu(stack):
    while True:
        print("\nStack Menu:")
        print("1. Push")
        print("2. Pop")
        print("3. Peek")
        print("4. Back to main menu")
        choice = int(input("Enter your choice: "))
        if choice == 1:
            data = int(input("Enter data to push: "))
            stack.push(data)
        elif choice == 2:
            print("Popped element:", stack.pop())
        elif choice == 3:
            print("Top element:", stack.peek())
        elif choice == 4:
            break
        else:
            print("Invalid choice")

def queue_menu(queue):
    while True:
        print("\nQueue Menu:")
```

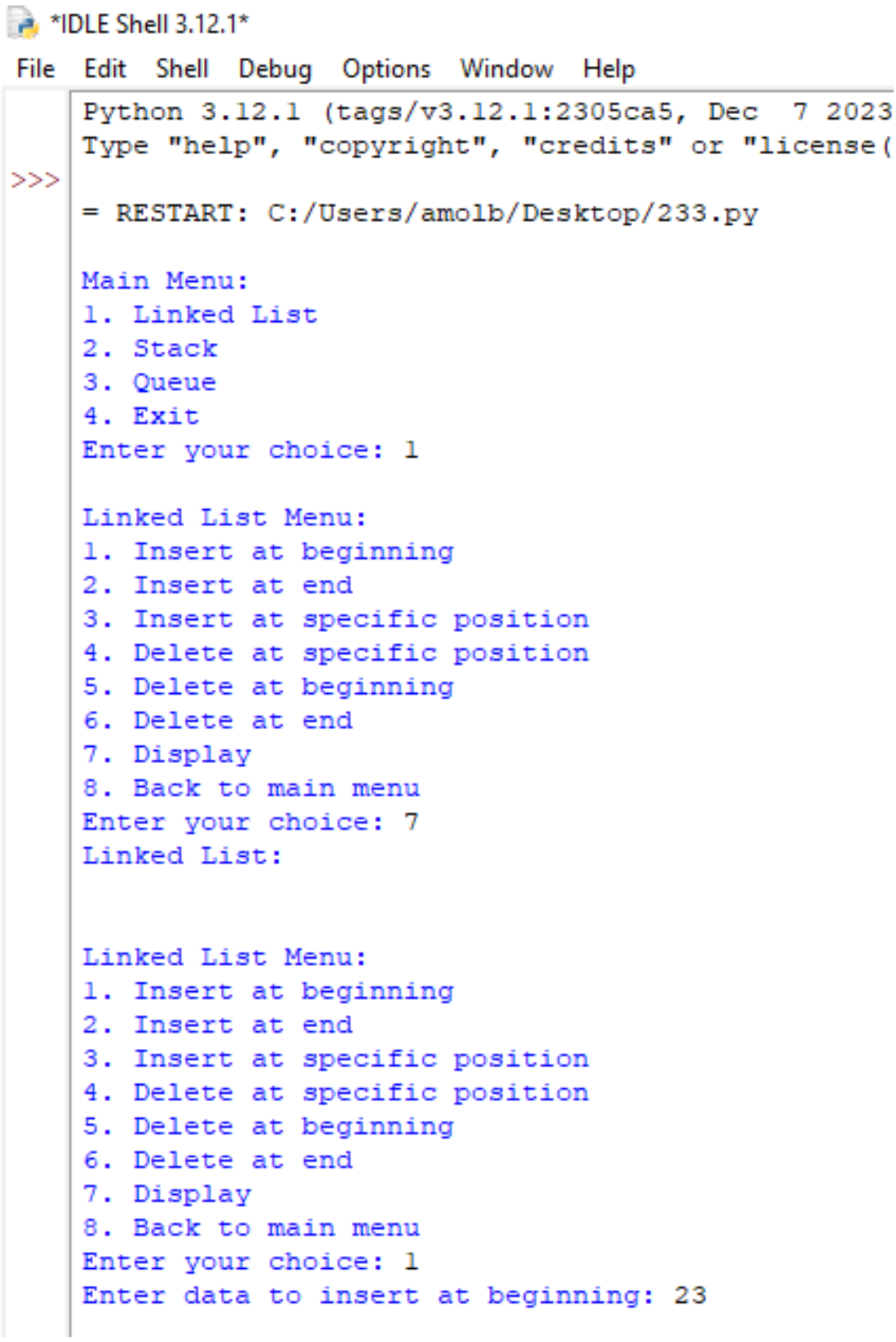
```
        print("1. Enqueue")
        print("2. Dequeue")
        print("3. View Queue")
        print("4. Back to main menu")
        choice = int(input("Enter your choice: "))
        if choice == 1:
            data = int(input("Enter data to enqueue: "))
            queue.enqueue(data)
        elif choice == 2:
            print("Dequeued element:", queue.dequeue())
        elif choice == 3:
            print("Queue:", queue.view())
        elif choice == 4:
            break
        else:
            print("Invalid choice")

if __name__ == "__main__":
    ll = LinkedList()
    stack = Stack()
    queue = Queue()

    while True:
        print("\nMain Menu:")
        print("1. Linked List")
        print("2. Stack")
        print("3. Queue")
        print("4. Exit")
        choice = int(input("Enter your choice: "))
```

```
if choice == 1:
    linked_list_menu(ll)
elif choice == 2:
    stack_menu(stack)
elif choice == 3:
    queue_menu(queue)
elif choice == 4:
    print("Exiting program")
    break
else:
    print("Invalid choice")
```

- **OUTPUT:**


*IDLE Shell 3.12.1*
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023
Type "help", "copyright", "credits" or "license(
>>>
= RESTART: C:/Users/amolb/Desktop/233.py
Main Menu:
1. Linked List
2. Stack
3. Queue
4. Exit
Enter your choice: 1
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 7
Linked List:
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 1
Enter data to insert at beginning: 23

```
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 1
Enter data to insert at beginning: 12

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 2
Enter data to insert at end: 34

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 3
Enter data to insert: 18
Enter position to insert at: 1
```

```
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 7
Linked List:
12 18 23 34

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 4
Enter position to delete: 1

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 7
Linked List:
12 23 34
```

```
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 5

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 6

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 7
Linked List:
23
```

```
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 8

Main Menu:
1. Linked List
2. Stack
3. Queue
4. Exit
Enter your choice: 2

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 1
Enter data to push: 34

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 1
Enter data to push: 44
```

```
Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 1
Enter data to push: 55

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 3
Top element: 55

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 2
Popped element: 55

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 2
Popped element: 44

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 2
Popped element: 34
```

```
Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 2
Stack is empty
Popped element: None

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 3
Stack is empty
Top element: None

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 4
```

```
Main Menu:
1. Linked List
2. Stack
3. Queue
4. Exit
Enter your choice: 3

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 3
Queue: []

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 1
Enter data to enqueue: 23

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 1
Enter data to enqueue: 34

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 1
Enter data to enqueue: 45
```

```
Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 3
Queue: [23, 34, 45]

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 2
Dequeued element: 23

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 2
Dequeued element: 34

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 2
Dequeued element: 45
```

```
Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 2
Queue is empty
Dequeued element: None

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 4

Main Menu:
1. Linked List
2. Stack
3. Queue
4. Exit
Enter your choice: 4
Exiting program
```

- ***CONCLUSION:*** **Hence, we have successfully implemented Menu driven program for data structure using built in function for Link List, Stack and Queue; LO 3.**