Experiment No. 7
Program for data structure using built in function for link list, stack and queues
Date of Performance:
Date of Submission:

Experiment No. 7

Title: Program for data structure using built in function for link list, stack and queues

Aim: To study and implement data structure using built in function for link list, stack and queues

Objective: To introduce data structures in python

Theory:

Stacks -the simplest of all data structures, but also the most important. A stack is a collection of objects that are inserted and removed using the LIFO principle. LIFO stands for "Last In First Out". Because of the way stacks are structured, the last item added is the first to be removed, and vice-versa: the first item added is the last to be removed.

Queues – essentially a modified stack. It is a collection of objects that are inserted and removed according to the FIFO (First In First Out) principle. Queues are analogous to a line at the grocery store: people are added to the line from the back, and the first in line is the first that gets checked out – BOOM, FIFO!

Linked Lists

The Stack and Queue representations I just shared with you employ the python-based list to store their elements. A python list is nothing more than a dynamic array, which has some disadvantages.

The length of the dynamic array may be longer than the number of elements it stores, taking up precious free space.

Insertion and deletion from arrays are expensive since you must move the items next to them over

Using Linked Lists to implement a stack and a queue (instead of a dynamic array) solve both of these issues; addition and removal from both of these data structures (when implemented with a linked list) can be accomplished in constant O(1) time. This is a HUGE advantage when dealing with lists of millions of items.

Linked Lists – comprised of 'Nodes'. Each node stores a piece of data and a reference to its next and/or previous node. This builds a linear sequence of nodes. All Linked Lists store a head, which is a reference to the first node. Some Linked Lists also store a tail, a reference to the last node in the list.

1] Stack Program:

```
# Stack

st = [3, 2, 21, 4, 56]

n = len(st)

print("Original Stack:", st)


# Functions on Stack

st.append(32)  # Inserting element 32

print("Element added to the stack:", st)


popped_element = st.pop()

print("Popped element from the stack:", popped_element)


top_element = st[n-1]

print("Topmost element of the stack:", top_element)


element_index = st.index(21)

print("Index of element 21 in the stack:", element_index)
```

```
# Checking if stack is empty or not
if st == []:
    print("Stack is empty")
else:
    print("Stack is not empty")
```

Output:

```
Original Stack: [3, 2, 21, 4, 56]
Element added to the stack: [3, 2, 21, 4, 56, 32]
Popped element from the stack: 32
Topmost element of the stack: 56
Index of element 21 in the stack: 2
Stack is not empty
```

2] Linked List Program:

```
# Linked List
ls = [4, 3, 1, 5, 21]
print("Original Linked List:", ls)

# Operations on Linked List

# Traversing
```

```
print("Elements in the Linked List:")
for element in ls:
    print(element)

ls.append(227)  # Appending 227 at the end
print("After appending 227 at the end:", ls)

ls.insert(3, 31)  # Inserting 31 at position 3
print("After inserting 31 at position 3:", ls)

ls.remove(4)  # Removing element 4
print("After removing element 4:", ls)

# Replacing an element
ls.remove(1)
ls.insert(2, 121)
print("After replacing 1 with 121 at index 2:", ls)

print("Index of element 5:", ls.index(5))  # Searching for element 5

print("Size of the list:", len(ls))  # Size of the list
```

Output:

Original Linked List: [4, 3, 1, 5, 21]

Elements in the Linked List:

4

3

1

5

21

After appending 227 at the end: [4, 3, 1, 5, 21, 227]

After inserting 31 at position 3: [4, 3, 1, 31, 5, 21, 227]

After removing element 4: [3, 1, 31, 5, 21, 227]

After replacing 1 with 121 at index 2: [3, 31, 121, 5, 21, 227]

Index of element 5: 3

Size of the list: 6

3] Queue Program:

```
class Queue:
    def __init__(self):
        # Initialize an empty list to store the elements of the queue
        self.que = []

    def isempty(self):
        # Check if the queue is empty
```

```
    return self.que == []


def add(self, element):
    # Add an element to the end of the queue
    self.que.append(element)


def delete(self):
    # Delete and return the element at the front of the queue
    if self.isempty():
        # If the queue is empty, return -1 indicating underflow
        return -1
    else:
        # Otherwise, remove and return the first element
        return self.que.pop(0)


def search(self, element):
    # Search for an element in the queue and return its position
    if self.isempty():
        # If the queue is empty, return -1 indicating underflow
        return -1
    else:
        try:
            # Attempt to find the index of the element
```

```
            n = self.que.index(element)
            # If found, return its position (1-indexed)
            return n + 1
        except ValueError:
            # If element not found, return -2 indicating element not found
            return -2

    def display(self):
        # Display the elements of the queue
        return self.que


# Test the Queue class
q = Queue()
q.add(1)
q.add(2)
q.add(3)
q.add(4)
print("Queue:", q.display())
print("Deleting element:", q.delete())
print("Queue after deletion:", q.display())
print("Search 3:", q.search(3))
```

Output:

Queue: [1, 2, 3, 4]

Deleting element: 1

Queue after deletion: [2, 3, 4]

Search 3: 2

Conclusion:

Through this experiment, various fundamental data structures such as stacks, queues, and linked lists were explored and implemented using built-in functions in Python. These structures follow distinct principles - LIFO for stacks, FIFO for queues, and node-based sequencing for linked lists, showcasing their diverse applications. By understanding and utilizing these structures, efficient data organization and manipulation can be achieved, enhancing programming proficiency and problem-solving skills in Python.