Experiment No. 7
Program for data structure using built in function for link list, stack and queues
Name : Vaidehi D. Gadag
Branch/Div.: Comps-1 (C47)
Date of Performance: 28/02/2024
Date of Submission: 20/03/2024

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.

Code:

```
#Creating an Empty list
lst = []

#Number of elements in a list
n = int(input("Enter number of elements : "))
print("The elements are : ")
for i in range(0, n):
        element = int(input())

        #Append the elements
        lst.append(element)
print("List : ",lst)

#Traversaling the element
print("The elements in the list are : ")
for element in lst:
   print(element)

#Insert the element
a = int(input("Enter the element to insert : "))
b = int(input("Enter the index to insert at : "))
lst.insert(b,a)
print("After insertion : ",lst)

#Remove the element
c = int(input("Enter the element to remove : "))
lst.remove(c)
print("After removing : ",lst)
#Search location of element
d = int(input("Enter the element to search : "))
if(lst.index(d)):
```

```
    print("Element is present at index : ",lst.index(d))
else:
    print("Element is not present")

#Replace the Element
e = int(input("Enter the element to remove : "))
lst.remove(e)
f = int(input("Enter the element to insert : "))
g = int(input("Enter the index to insert at : "))
lst.insert(g,f)
print("After replacing : ",lst)

#Size of the list
print("The size of the list : ",len(lst))
```

Output:

```
Enter number of elements : 3
The elements are :
45
23
87
List :  [45, 23, 87]
The elements in the list are :
45
23
87
Enter the element to insert : 95
Enter the index to insert at : 1
After insertion :  [45, 95, 23, 87]
Enter the element to remove : 23
After removing :  [45, 95, 87]
Enter the element to search : 87
Element is present at index :  2
Enter the element to remove : 45
Enter the element to insert : 13
Enter the index to insert at : 0
After replacing :  [13, 95, 87]
The size of the list :  3
```

Conclusion:

Python's linked list data structures offer several advantages over arrays. They allow for dynamic memory allocation, making them ideal for large lists and reducing the need for contiguous memory. Moreover, insertion and deletion operations are faster in linked lists due to the minimal number of references needed. Python provides built-in support for linked lists through the dynamic array data type, but custom linked list classes offer more flexibility. It's crucial to choose the appropriate type of linked list, such as singly linked list or doubly linked list, for optimal performance. Overall, linked lists are a powerful data structure in Python.