Experiment No. 7
Program for data structure using built in function for link list, stack and queues
Date of Performance: 05/03/2024
Date of Submission: 12/03/2024

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.

Program :

```
# Linked List

list = [2,3,4,5]

print("Traverse Function: ")

print("The result after traversing the list: ");

for element in list :

    print(element)


print("Append function: ")

val = int(input("Enter the element to append : "))

list.append(val)

print("Linked List : ",list)


print("Insert function :")

elemt = int(input("Enter the element to insert: "))

pos = int(input("Enter the position where you want to insert: "))

list.insert(pos,elemt)

print("Linked List : ",list)
```

```
print("Remove function :")

value = int(input("Enter the element you want to delete: "))

list.remove(value)

print("Linked List : ",list)


print("Search function: ")

num = int(input("Enter the element of which you to find index: "))

ind = list.index(num)

print("Index of given element: ",ind)


print("Replace function: ")

old = int(input("Enter the element value where you want to replace: "))

new = int(input("Enter the element you want to replace: "))

pos1 = list.index(old)

list.remove(old)

list.insert(pos1,new)

print("Linked List : ",list)


print("Length function :")

print("Length of given list: ",len(list))
```

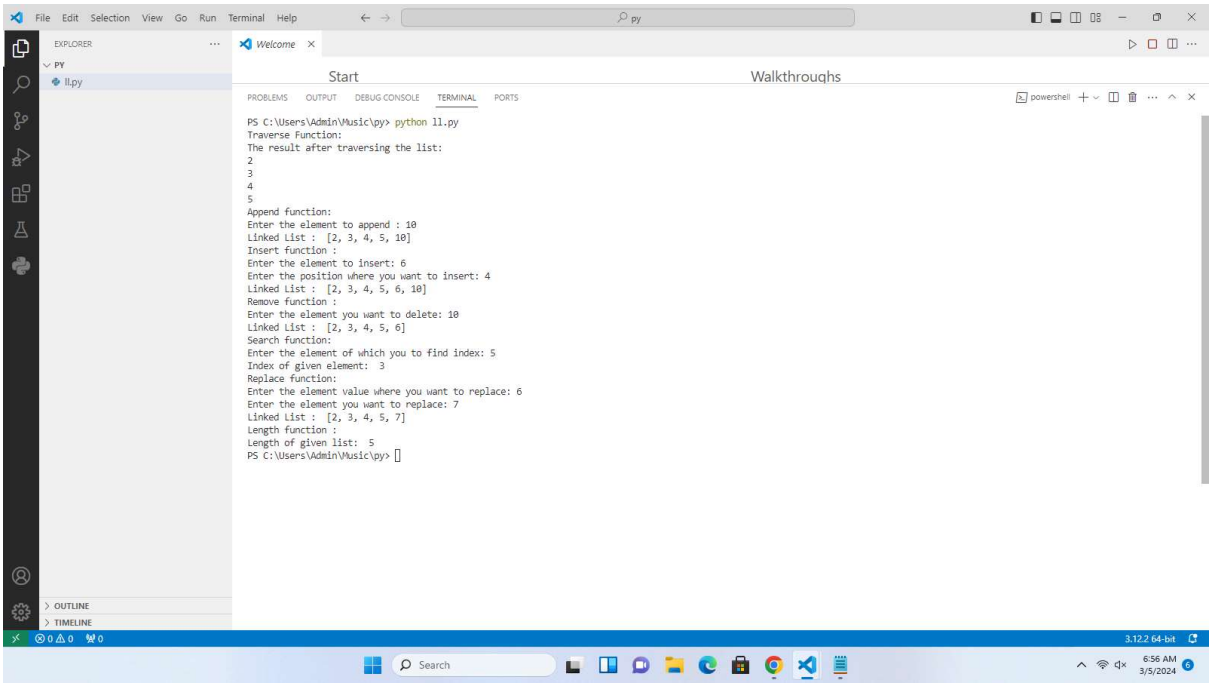
```
#Stack

stack = [1,2]


print("Insert Element in Stack")

value = int(input("Enter the element: "))

stack.append(value)

print("Element ",value," is inserted into stack")

print("Stack: ",stack)

print()


print("Delete Element from Stack")

val = stack[len(stack)-1]

stack.pop()

print("Element ",val," is deleted from stack")

print("Stack: ",stack)

print()


print("Peek TopMost Element of Stack: ")

top = stack[len(stack)-1]

print("Topmost element of stack : ",top)

print()


print("Searching an element in Stack")
```

```
key = int(input("Enter the element to search: "))
index = stack.index(key)
print("The element ",key," is at index ",index)
print()

print("To Check whether Stack is Empty : ")
if(stack==[]):
    print("Stack is Empty")

else :
    print("Stack is Not Empty")
print()

#Queue

queue = [2,3,4]
print("Insert element into Queue")
elem = int(input("Enter the element to insert: "))
queue.append(elem)
print("Element ",elem," is inserted in the Queue")
print("Queue : ",queue)
print()
print("Delete element from Queue")
```

```
val = queue[0]

queue.pop(0)

print("Element ",val," deleted from queue")

print("Queue : ",queue)

print()

print("Search element in Queue")

key = int(input("Enter the element to search : "))

index = queue.index(key)

print("Element ",key," is at index ",index)
```

Output :

#Linked List



#Stack

```
C:\Users\Student\Documents\Vivek>C:/Users/Student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Student/Documents/Vivek/stack.py
Insert Element in Stack
Enter the element: 5
Element  5  is inserted into stack
Stack:  [1, 2, 5]

Delete Element from Stack
Element  5  is deleted from stack
Stack:  [1, 2]

Peek TopMost Element of Stack:
Topmost element of stack :  2

Searching an element in Stack
Enter the element to search: 2
The element  2  is at index  1

To Check whether Stack is Empty :
Stack is Not Empty
```

#Queue

```
C:\Users\Student\Documents\Vivek>C:/Users/Student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Student/Documents/Vivek/queue.py
Insert element into Queue
Enter the element to insert: 5
Element  5  is inserted in the Queue
Queue :  [2, 3, 4, 5]

Delete element from Queue
Element  2  deleted from queue
Queue :  [3, 4, 5]

Search element in Queue
Enter the element to search : 5
Element  5  is at index  2
```

Conclusion:

In summary, linked lists, stacks, and queues are fundamental data structures in Python, each serving distinct purposes. Linked lists offer dynamic memory allocation and efficient insertion/deletion operations. Stacks, operating on the Last-In-First-Out (LIFO) principle, are useful for tracking function calls, expression evaluation, and managing undo operations. On the other hand, queues, following the First-In-First-Out (FIFO) principle, are employed in tasks such as scheduling, breadth-first search, and managing resource access. These data structures form the backbone of many algorithms and software applications, providing efficient solutions to a wide range of problems in computer science and beyond.