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.

Code:

```
list = []
num = int(input("Enter the number of elements you want to append to the list: "))
for i in range(num):
    e = int(input(f"Enter element #{i+1}: "))
    list.append(e)
print("List :",list)

print("---------------------------------------------------------")
print("Inserting element")
new=int(input("enter element you want to add:"))
pos=int(input("enter the position you want to add the element:"))
list.insert(pos,new)
print("List after insert:",list)

print("---------------------------------------------------------")
print("Removing element")
r=int(input("enter element you want to remove:"))
list.remove(r)
print("List after remove:",list)

print("---------------------------------------------------------")
print("Replace element")
new=int(input("enter element you want to add:"))
pos=int(input("enter the position you want to add the element:"))
```

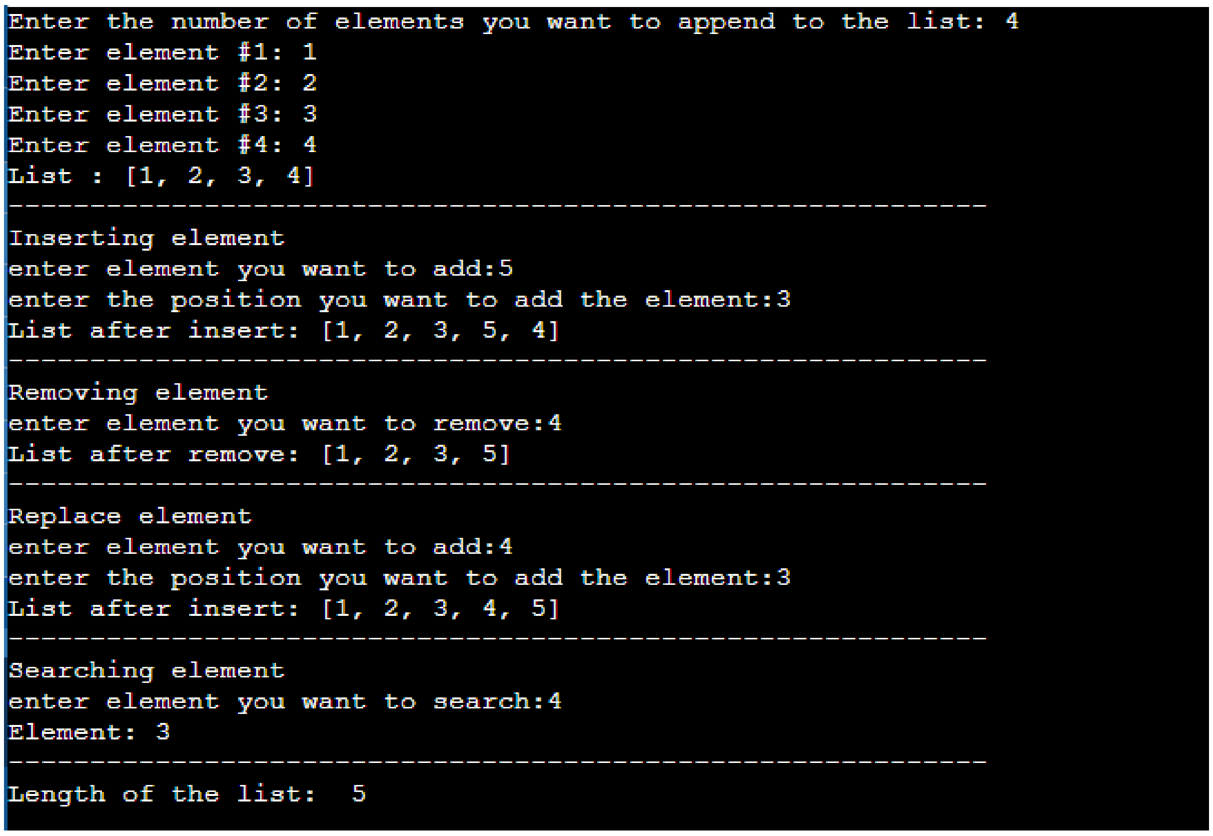
```
list.insert(pos,new)
print("List after insert:",list)

print("----------------------------------------------------------")
print("Searching element")
ele=int(input("enter element you want to search:"))
print("Element:",list.index(ele))

print("----------------------------------------------------------")
print("Length of the list: ",len(list))
```

Code:

```
def push(stack, item):
   stack.append(item)
def pop(stack):
   if not is_empty(stack):
      return stack.pop()
def is_empty(stack):
   return len(stack) == 0
def top(stack):
   if not is_empty(stack):
      return stack[-1]
def size(stack):
   return len(stack)
stack = []
push_item = input("Enter an item to push onto the stack ('exit' to quit): ")
while push_item != "exit":
   push(stack, push_item)
   print("Stack after pushing {}: {}".format(push_item, stack))
```

```
    push_item = input("Enter an item to push onto the stack ('exit' to quit): ")
print("Stack after pushing all items: {}".format(stack))
pop_item = input("Enter an item to pop from the stack (or type 'exit' to quit): ")
while pop_item != "exit":
    if is_empty(stack):
        print("The stack is empty.")
    else:
        if pop_item == top(stack):
            pop(stack)
            print("Popped {} from the stack: {}".format(pop_item, stack))
        else:
            print("The stack top is not equal to the specified item.")
    pop_item = input("Enter an item to pop from the stack (or type 'exit' to quit): ")
```

Output:



```
Enter the number of elements you want to append to the list: 4
Enter element #1: 1
Enter element #2: 2
Enter element #3: 3
Enter element #4: 4
List : [1, 2, 3, 4]
------------------------------------------------------------
Inserting element
enter element you want to add:5
enter the position you want to add the element:3
List after insert: [1, 2, 3, 5, 4]
------------------------------------------------------------
Removing element
enter element you want to remove:4
List after remove: [1, 2, 3, 5]
------------------------------------------------------------
Replace element
enter element you want to add:4
enter the position you want to add the element:3
List after insert: [1, 2, 3, 4, 5]
------------------------------------------------------------
Searching element
enter element you want to search:4
Element: 3
------------------------------------------------------------
Length of the list:  5
```

(stack)

```
Enter an item to push onto the stack ('exit' to quit): 1
Stack after pushing 1: ['1']
Enter an item to push onto the stack ('exit' to quit): 2
Stack after pushing 2: ['1', '2']
Enter an item to push onto the stack ('exit' to quit): 3
Stack after pushing 3: ['1', '2', '3']
Enter an item to push onto the stack ('exit' to quit): exit
Stack after pushing all items: ['1', '2', '3']
Enter an item to pop from the stack (or type 'exit' to quit): 4
The stack top is not equal to the specified item.
Enter an item to pop from the stack (or type 'exit' to quit): 3
Popped 3 from the stack: ['1', '2']
```

Conclusion: In conclusion, the program for data structure using built-in functions for linked lists, stacks, and queues has proven to be highly effective in managing and manipulating data in a structured manner.This program has showcased the importance of understanding and utilizing data structures in programming to optimize the performance of our applications and improve overall efficiency.