

Vidyavardhini's College of Engineering & Technology Department of Computer Engineering

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



Department of Computer Engineering

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.



Department of Computer Engineering

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: STACKclass Stack: def __init__(self): self.stack = [] def push(self, item): self.stack.append(item) def pop(self): if self.is_empty(): return None return self.stack.pop() def peek(self): if self.is_empty(): return None return self.stack[-1]



```
def is_empty(self):
     return len(self.stack) == 0
  def size(self):
     return len(self.stack)
  def display(self):
     if self.is_empty():
       print("Stack is empty.")
     else:
       print("Current stack:")
       print("\n".join(map(str, self.stack[::-1])))
def main():
  stack = Stack()
  while True:
     print("\nStack Operations:")
     print("1. Push")
     print("2. Pop")
```



```
print("3. Peek")
print("4. Size")
print("5. Display Stack")
print("6. Exit")
choice = input("Enter your choice (1-6): ")
if choice == '1':
  item = input("Enter item to push: ")
  stack.push(item)
  print(f"{item} pushed to stack.")
elif choice == '2':
  item = stack.pop()
  if item is not None:
     print(f"Popped item: {item}")
  else:
     print("Stack is empty.")
elif choice == '3':
  item = stack.peek()
  if item is not None:
     print(f"Top item: {item}")
  else:
```



```
print("Stack is empty.")
     elif choice == '4':
       print("Size of stack:", stack.size())
     elif choice == '5':
       stack.display()
     elif choice == '6':
       print("Exiting...")
       break
     else:
       print("Invalid choice. Please enter a number between 1 and 6.")
if __name__ == "__main___":
  main()
QUEUE-
class Queue:
  def __init__(self):
     self.queue = []
  def enqueue(self, item):
     self.queue.append(item)
```



```
def dequeue(self):
     if self.is_empty():
       return None
     return self.queue.pop(0)
  def peek(self):
     if self.is_empty():
       return None
     return self.queue[0]
  def is_empty(self):
     return len(self.queue) == 0
  def size(self):
     return len(self.queue)
  def display(self):
     if self.is_empty():
       print("Queue is empty.")
     else:
       print("Current queue:")
       print(" ".join(map(str, self.queue)))
def main():
```



```
queue = Queue()
while True:
  print("\nQueue Operations:")
  print("1. Enqueue")
  print("2. Dequeue")
  print("3. Peek")
  print("4. Size")
  print("5. Display Queue")
  print("6. Exit")
  choice = input("Enter your choice (1-6): ")
  if choice == '1':
     item = input("Enter item to enqueue: ")
     queue.enqueue(item)
     print(f"{item} enqueued.")
  elif choice == '2':
     item = queue.dequeue()
     if item is not None:
       print(f"Dequeued item: {item}")
     else:
       print("Queue is empty.")
  elif choice == '3':
     item = queue.peek()
```



```
if item is not None:
          print(f"Front item: {item}")
       else:
          print("Queue is empty.")
     elif choice == '4':
       print("Size of queue:", queue.size())
     elif choice == '5':
       queue.display()
     elif choice == '6':
       print("Exiting...")
       break
     else:
       print("Invalid choice. Please enter a number between 1 and 6.")
if __name__ == "__main__":
  main()
LINK LIST-
class Node:
  def init (self, data):
     self.data = data
     self.next = None
class LinkedList:
```



Department of Computer Engineering

```
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
definsert at end(self, data):
  new node = Node(data)
  if self.head is None:
     self.head = new_node
     return
  last\_node = self.head
  while last_node.next:
     last_node = last_node.next
  last node.next = new node
def delete_node(self, key):
  temp = self.head
  if temp is not None:
     if temp.data == key:
       self.head = temp.next
```

temp = None



Department of Computer Engineering

return while temp is not None: if temp.data == key: break prev = temptemp = temp.nextif temp == None: return prev.next = temp.nexttemp = Nonedef search(self, key): current = self.head while current: if current.data == key: return True current = current.next return False def display(self): current = self.headwhile current:

print(current.data, end=" -> ")



```
current = current.next
     print("None")
# Main function
if __name__ == "__main__":
  linked list = LinkedList()
  while True:
     print("\nLinked List Operations:")
     print("1. Insert at beginning")
    print("2. Insert at end")
    print("3. Delete node")
    print("4. Search node")
     print("5. Display linked list")
    print("6. Exit")
    choice = int(input("Enter your choice: "))
     if choice == 1:
       data = int(input("Enter data to insert at beginning: "))
       linked_list.insert_at_beginning(data)
     elif choice == 2:
```

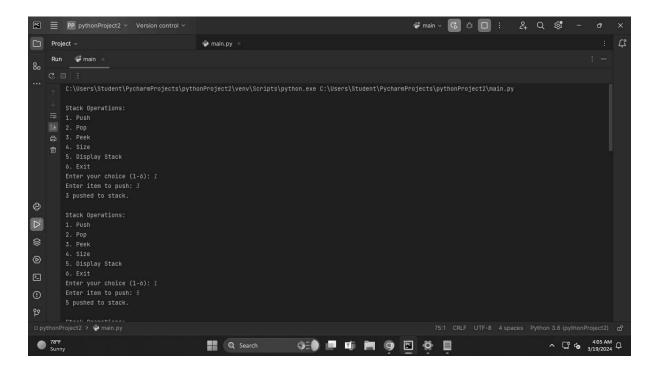


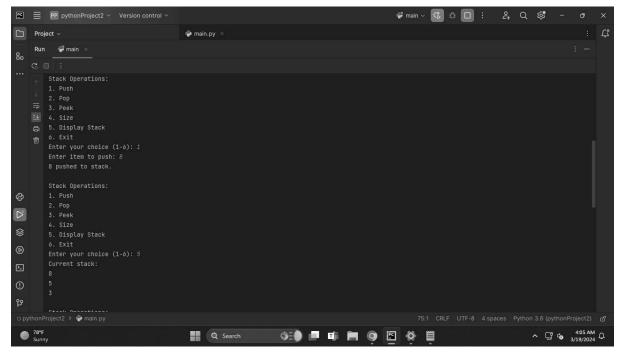
```
data = int(input("Enter data to insert at end: "))
  linked list.insert at end(data)
elif choice == 3:
  key = int(input("Enter data to delete: "))
  linked list.delete node(key)
elif choice == 4:
  key = int(input("Enter data to search: "))
  if linked list.search(key):
     print("Data found in the linked list.")
  else:
     print("Data not found in the linked list.")
elif choice == 5:
  linked_list.display()
elif choice == 6:
  print("Exiting...")
  break
else:
  print("Invalid choice. Please enter a valid option.")
```



Department of Computer Engineering

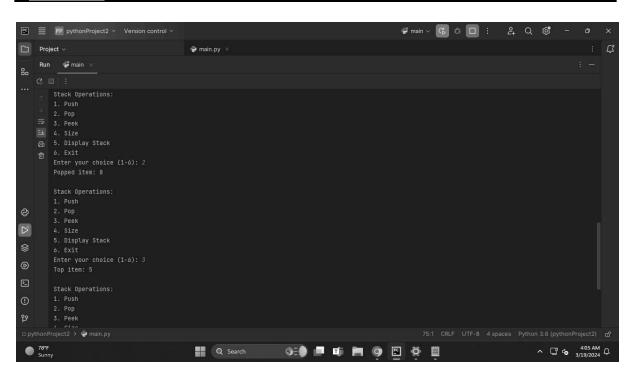
Stack:

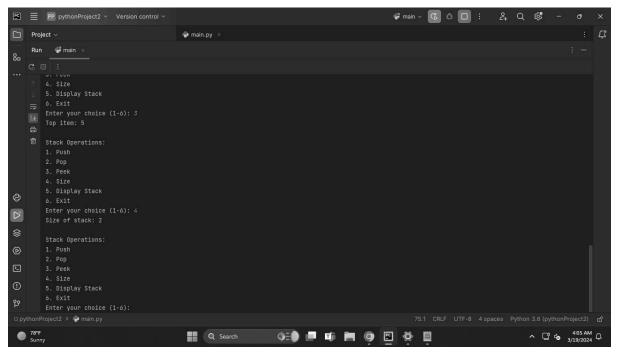






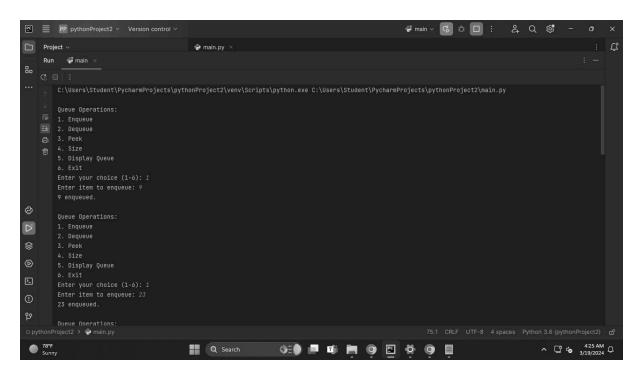
Department of Computer Engineering

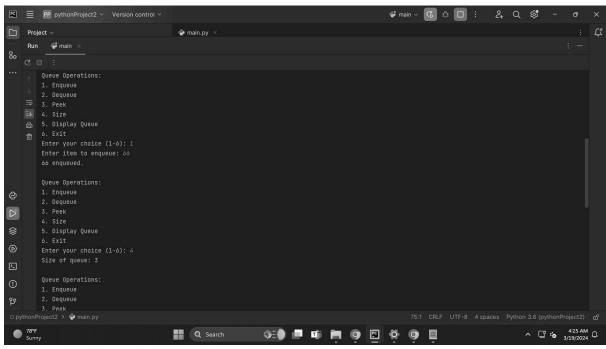




Queue:

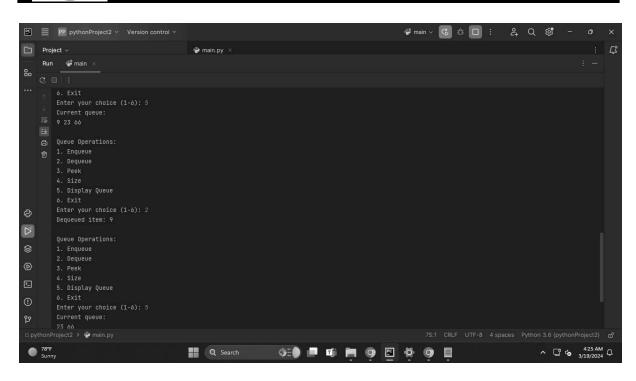


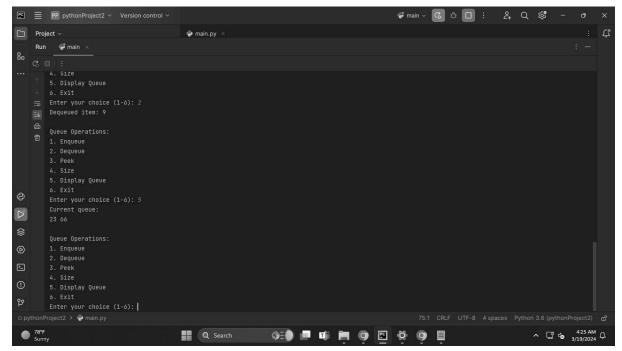






Department of Computer Engineering





Link list:



```
C:\Users\anand\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\anand\PycharmProjects\pythonProject\main.py

Linked List Operations:

1. Insert at beginning

2. Insert at end

3. Delete node

4. Search node

5. Display linked list

6. Exit
Enter your choice: 1
Enter data to insert at beginning: 2

Linked List Operations:

1. Insert at beginning

2. Insert at end

3. Delete node

4. Search node

5. Display linked list

6. Exit
Enter your choice: 2
Enter data to insert at end: 5
```

```
3. Delete node
4. Search node
5. Display linked list
6. Exit
Enter your choice: 5
2 -> 5 -> None

Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Delete node
4. Search node
5. Display linked list
6. Exit
Enter your choice: 1
Enter data to insert at beginning: 6

Linked List Operations:
1. Insert at beginning: 6

Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Delete node
4. Search node
5. Display linked list Operations:
1. Insert at beginning
2. Insert at end
3. Delete node
4. Search node
5. Display linked list
6. Exit
Enter your choice:
```



Department of Computer Engineering

```
6. Exit
Enter your choice: 1
Enter data to insert at beginning: \delta
Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Delete node
4. Search node
5. Display linked list
6. Exit
Enter your choice: 5
Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Delete node
4. Search node
5. Display linked list
```

```
4. Search node
5. Display linked list
6. Exit
Enter your choice: 5
6 -> 5 -> None

Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Delete node
4. Search node
5. Display linked list
6. Exit
Enter your choice: 4
Enter data to search: 5
Data found in the linked list.

Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Delete node
4. Search node
5. Display linked list
6. Exit
Enter your choice: 4
Enter data to search: 5
Data found in the linked list.
```

Conclusion: The exploration of data structures in Python facilitates efficient organization and manipulation of data. By implementing linked lists, stacks, and queues using built-in functions, developers gain practical experience in handling different data structures. Understanding these fundamental concepts enhances problem-solving skills and promotes code readability and scalability. Python's versatile built-in data structures enable developers to build robust and adaptable solutions for a wide range of computational tasks.