



K.R. MANGALAM UNIVERSITY

DATA STRUCTURES LAB

ENCS253

LAB PRACTICALS

SUBMITTED BY :

NAME : UTSAV RATAN

BTECH CSE CORE

SECTION : B

ROLL NUMBER : 2401010046

SEMESTER : 3rd

SUBMITTED TO :

Dr. SWATI GUPTA

INDEX

S.No.	Name of Experiment
1.	Implement Inventory Management System using Array List.
2.	Implement Inventory Stock Manager to Process Sales and Identify Zero Stock .
3.	Implement Linear Search and analyse time complexity.
4.	Implement Insertion in Circular Linked List (Beginning and End) .
5.	Implement Deletion in Circular Linked List (Beginning and End) .
6.	Implement Deletion from Singly Linked List (Beginning and End) .
7.	Implement Circular Queue using Array
8.	Implement Stack using Array (Push, Pop, Peek, Display).
9.	Evaluate Postfix Expression using Stack .
10.	Simulate Browser Back Button using Stack .
11.	Implement Bubble Sort and analyse time complexity.
12.	Implement Binary Search and analyse time complexity.

EXPERIMENT – 1

INVENTORY MANAGEMENT SYSTEM

AIM:

To implement an inventory management system in Python that allows insertion of new products and display of all products using ArrayList.

Question:

Write a menu-driven program to store product details (SKU, name, quantity) in an inventory. The program should allow the user to insert new products after validating quantity and to display the complete inventory in tabular form.

Introduction:

Inventory management keeps track of items available in stock. Using a list in Python, we can store a dynamic list of products where each product is represented as an object containing SKU, name and quantity.

Algorithm:

- Start.
- Create a Product class with fields sku, name and quantity.
- Use a Python list to store all product records.
- Display menu with options:
 1. Insert product
 2. Display inventory
 3. Exit
- For insertion: read sku, name and quantity; validate that quantity is positive and sku does not already exist; then add new Product object to list.

- For display: if list is empty, show message; otherwise print all products in table form.
- Repeat menu until user selects Exit.
- Stop.

Python Code:

```

class Product:
def __init__(self, sku, name, quantity):
    self.sku = sku
    self.name = name
    self.quantity = quantity
    inventory = []
def
    insert_product():
        sku = input("Enter SKU: ").strip()
        for p in inventory:
            if p.sku.lower() == sku.lower(): print("Product
with this SKU already exists!")
        return
        name = input("Enter Product Name:
").strip() if not name:
            print("Product name cannot be empty.")
        return
        try:
            qty = int(input("Enter Quantity: ").strip())
            if qty <= 0: print("Quantity must be
positive.") return except
            ValueError:
            print("Invalid quantity.")
        return
        inventory.append(Product(sku, name,
        qty)) print("Product inserted
successfully.")
def
    display_inventory():

```

```
if not inventory:
    print("Inventory is empty.")
    return
print("SKU\t\tName\t\tQuantity") print("-----"
-----
for p in
inventory:
    print(f"{p.sku}\t\t{p.name}\t\t{p.quantity}")
def main():
    while True:
        print("\nInventory Management System")
        print("1. Insert Product") print("2.
Display Inventory") print("3. Exit")
        ch = input("Enter your choice: ").strip()
        if ch ==
        "1":
            insert_product()
        elif ch == "2":
            display_inventory()
        elif ch == "3":
            print("Exiting...")
            break
        else:
            print("Invalid choice.")
    if __name__ ==
    '__main__': main()
```

EXPERIMENT – 2

Inventory Stock Manager – Process Sales &Zero Stock

AIM:

To implement an inventory stock manager that processes sales for a given SKU and identifies items with zero stock.

Question:

Write a program that maintains a list of items (SKU and quantity). Implement a method to process a sale given SKU and quantity sold, updating stock if available or showing appropriate error messages. Also implement a method to list all SKUs whose quantity becomes zero.

Algorithm:

- Use a class `Item` with fields `sku` and `quantity`.
- Store all items in a list.
- For `processSale(sku, qtySold)`: search the list for given `sku`.
- If `sku` not found: display message.
- If found and $quantity \geq qtySold$: reduce `quantity` and show success message.
- If found and $quantity < qtySold$: do not update `quantity` and show insufficient stock message.
- For `identifyZeroStock()`: traverse list and collect all items whose `quantity` is 0 and display them.

Python Code:

```
class Item: def __init__(self,  
sku, quantity):  
self.sku = sku
```

```
self.quantity = quantity
def process_sale(inventory, sku,
qty_sold):
    found = False
    for item in inventory:
        if item.sku == sku:
            found = True
            if item.quantity >= qty_sold:
                item.quantity -= qty_sold
                print(f"Sale processed: {qty_sold} units of SKU {sku}")
            else:
                print(f"Insufficient stock for SKU {sku}. Available: {item.quantity}")
                break
    if not found:
        print(f"SKU {sku} not found in inventory.")
def identify_zero_stock(inventory):
    zero_list = [item.sku for item in inventory if item.quantity == 0]
    if not zero_list:
        print("No zero stock items found.")
    else:
        print("Zero stock SKUs:", zero_list)
    return zero_list
def main():
    inventory = [
        Item(101, 50),
        Item(102, 20),
        Item(103, 0)
    ]
    process_sale(inventory, 101, 30) # normal sale
    process_sale(inventory, 102, 25) # insufficient stock
```

```
process_sale(inventory, 104, 10) # SKU not found
identify_zero_stock(inventory)
print("Updated Inventory:", [(item.sku,
item.quantity) for item in
inventory])
if __name__ ==
'__main__':
main()
```

EXPERIMENT – 3

Linear Search

AIM:

To implement linear search and analyse its best and worst case time complexity.

Question:

Write a program to perform linear search on an array of integers.
Also state the best and worst case time complexities of linear search.

Introduction:

Linear search scans the array sequentially from left to right and compares each element with the key. It works on both sorted and unsorted arrays but it is inefficient for large datasets.

Algorithm:

- Input array A of n elements and key K.
- Set i = 0.
- While $i < n$, compare $A[i]$ with K.
- If $A[i] == K$, return index i.
- Else increment i and continue loop.
- If no element matches, return -1 meaning key not found.

Python Code:

```
def linear_search(arr, key):  
    for i, val in enumerate(arr):  
        if val == key: return i  
    return -1  
def main(): n = int(input("Enter number of  
elements: ")) arr = []  
print(f"Enter {n}")
```

```
elements:") for _ in
range(n):
arr.append(int(input()))
key = int(input("Enter element to search: "))
index = linear_search(arr,
key)
if index == -
1:
print("Element not found.")
else:
print("Element found at index:", index) if
__name__ == "__main__":
main()
```

EXPERIMENT – 4

Insertion in Circular Linked List (Beginning & End)

AIM:

To implement insertion at beginning and at end in a circular linked list.

Question:

Write a program to create a circular linked list and perform insertion of nodes at the beginning and at the end, displaying the list after each operation.

Python Code:

```
class CNode: def
    __init__(self, data):
        self.data = data
        self.next = None
class
CircularLinkedListInsert:
def __init__(self):
    self.head = None
def insert_at_end(self, data):
    new_node = CNode(data) if
    self.head is None:
        self.head = new_node
    new_node.next = self.head
    return
    temp = self.head
    while temp.next != self.head:
        temp = temp.next
        temp.next = new_node
        new_node.next = self.head
def insert_at_beginning(self,
data):
```

```
new_node = CNode(data)
if self.head is None:
    self.head = new_node
    new_node.next = self.head
    return
temp = self.head
while temp.next != self.head:
    temp = temp.next
new_node.next = self.head
temp.next = new_node
self.head = new_node
def display(self):
    if self.head is None:
        print("List is empty")
        return
    temp = self.head
    print("Circular List:", end=" ")
    while True:
        print(f"{temp.data} ->", end=" ")
        if temp == self.head:
            break
    print("(back to head)")
def main():
    cll = CircularLinkedListInsert()
    cll.insert_at_end(10)
    cll.insert_at_end(20)
    cll.insert_at_end(30)
    print("Original list:")
    cll.display()
    cll.insert_at_beginning(5)
    print("After inserting 5 at beginning:")
    cll.display()
    cll.insert_at_end(40)
    print("After inserting 40 at end:")
    cll.display()
```

```
if __name__ ==  
"__main__": main()
```

EXPERIMENT – 5

Deletion in Circular Linked List (Beginning & End)

AIM:

To delete a node from the beginning and from the end of a circular linked list.

Python Code:

```
class CNode: def
    __init__(self, data):
        self.data = data
        self.next = None
class
CircularLinkedListDelete:
def __init__(self):
    self.head = None
def insert(self, data):
    new_node = CNode(data) if
    self.head is None:
        self.head = new_node
    new_node.next = self.head
    return
    temp = self.head
    while temp.next != self.head:
        temp = temp.next
    temp.next = new_node
    new_node.next = self.head
def
delete_from_beginning(self):
    if self.head is None:
        print("List is empty, nothing to delete.")
```

```
return
if self.head.next ==
self.head:
    self.head = None
    print("Deleted the only node in the list.")
return
last = self.head
while last.next != self.head:
    last = last.next
    self.head =
    self.head.next last.next =
    self.head
    print("Node deleted from beginning.")
def
delete_from_end(self):
if self.head is None:
    print("List is empty, nothing to delete.")
return
if self.head.next ==
self.head: self.head = None
    print("Deleted the only node in the list.")
return
prev = None
temp = self.head
while temp.next != self.head:
    prev = temp
    temp = temp.next
    prev.next = self.head
    print("Node deleted from end.")
def
display(self):
if self.head is None:
    print("List is empty") return
temp = self.head
print("Circular List:", end=" ")
```

```
while True:
    print(f"{temp.data} ->", end=" ")
    if temp == self.head:
        temp = temp.next
        break
    print("(back to head)")
def main():
    cll = CircularLinkedListDelete()
    cll.insert(10)
    cll.insert(20)
    cll.insert(30)
    cll.insert(40)
    print("Initial
list:") cll.display()
    cll.delete_from_beginning()
    cll.display()
    cll.delete_from_end()
    cll.display()
    if __name__ == "__main__":
        main()
```

EXPERIMENT – 6

Deletion from Singly Linked List (Beginning & End)

AIM:

To implement deletion of nodes from the beginning and end of a singly linked list.

Python Code:

```
class SNode: def
    __init__(self, data):
        self.data = data
        self.next = None
class
SinglyLinkedListDelete:
def __init__(self):
    self.head = None
def insert(self, data):
    new_node = SNode(data)
    if self.head is None:
        self.head = new_node
    return
    temp = self.head
    while temp.next is not None:
        temp = temp.next
    temp.next = new_node
def
delete_from_beginning(self):
    if self.head is None:
        print("List is empty!")
    return
    self.head = self.head.next
    print("Node deleted from beginning.")
def
delete_from_end(self):
```

```
if self.head is None:
    print("List is empty!")
    return
if self.head.next is
None:
    self.head = None
    print("Last node deleted.")
    return
temp = self.head while
temp.next.next is not None:
    temp = temp.next
    temp.next = None
    print("Node deleted from end.")def display(self):
if
self.head is None:
    print("List is empty!")
    return
temp = self.head while temp
is not None: print(f"{temp.data}
->", end=" ") temp = temp.next
print("null")
def
main():
lst = SinglyLinkedListDelete()
lst.insert(10)
lst.insert(20)
lst.insert(30)
lst.insert(40)
print("Original
List:") lst.display()
lst.delete_from_beginning()
print("After deleting from beginning:")
lst.display()
lst.delete_from_end()
print("After deleting from end:")
lst.display()
```

```
if __name__ ==  
"__main__":  
    main()
```

EXPERIMENT – 7

Circular Queue using Array

AIM:

To implement a circular queue using array with enqueue, dequeue and display operations.

Python Code:

```
class CircularQueue: def
    __init__(self, capacity):
        self.arr = [None] * capacity
        self.front = -1 self.rear =
        -1 self.size = capacity
    def
    is_empty(self):
        return self.front == -1
    def
    is_full(self):
        return (self.front == 0 and self.rear == self.size
        - 1) or \
        (self.rear + 1 == self.front)
    def enqueue(self,
        value): if
        self.is_full():
            print("Queue is full.")
            return
        if self.front == -
        1: self.front =
            0
            self.rear = (self.rear + 1) % self.size
            self.arr[self.rear] = value print("Enqueued:", value)
    def
    dequeue(self):
```

```
if self.is_empty():
    print("Queue is empty.")
    return None
element = self.arr[self.front]
if self.front == self.rear:
    # Reset queue
    self.front = self.rear = -1
else:
    self.front = (self.front + 1) % self.size
print("Dequeued:", element)
return element

def display(self):
    if self.is_empty():
        print("Queue is empty.")
    return
    print("Elements in queue:", end=" ")
    i = self.front
    while i != self.rear:
        print(self.arr[i], end=" ")
        i = (i + 1) % self.size
    print()

def main():
    q = CircularQueue(5)
    q.enqueue(10)
    q.enqueue(20)
    q.enqueue(30)
    q.enqueue(40)
    q.display()
    q.dequeue()
    q.dequeue()
    q.display()
    q.enqueue(50)
    q.enqueue(60)
    q.display()
    if __name__ == "__main__":
        main()
```

EXPERIMENT – 8

Stack using Array (Push, Pop, Peek, Display)

AIM:

To implement stack operations push, pop, peek and display using array.

Python Code:

```
class ArrayStack: def
    __init__(self, capacity):
        self.capacity = capacity
        self.stack = [None] * capacity
        self.top = -1
    def
        is_empty(self):
            return self.top == -1
    def
        is_full(self):
            return self.top == self.capacity - 1
    def push(self, item):
        if self.is_full():
            print("Stack overflow.")
        return self.top += 1
        self.stack[self.top] = item
        print("Pushed", item)
    def pop(self):
        if
            self.is_empty():
                print("Stack underflow.")
        return None
        item =
        self.stack[self.top]
        self.top
        -= 1
        print("Popped", item)
        return item
    def peek(self):
        if
            self.is_empty():
```

```
print("Stack is empty.")
return None
print("Top element is", self.stack[self.top])
return self.stack[self.top]
def
display(self):
if self.is_empty():
print("Stack is empty.")
return
print("Stack elements:", end=" ")
for i in range(self.top + 1):
print(self.stack[i], end=" ") print()
def
main():
st = ArrayStack(10)
while
True:
print("\n1. Push 2. Pop 3. Peek 4. Display 5.
Exit")
ch = input("Enter your choice: ").strip() if ch ==
"1": int(input("Enter element: "))
st.push(x)
x =
elif ch ==
"2":
st.pop()
elif ch ==
"3":
st.peek()
st.display()
elif ch == "4":
elif ch ==
"5":
print("Exiting.")
break
else:
```

```
print("Invalid choice.")  
if __name__ ==  
"__main__":  
    main()
```

EXPERIMENT – 9

Evaluate Postfix Expression using Stack

AIM:

To evaluate a postfix arithmetic expression using stack.

Python Code:

```
def apply_operation(op1, op2, operator):  
    if operator == '+': return op1 +  
    op2 if operator == '-':  
        return op1 - op2 if operator == '*':  
            return op1 * op2 if operator == '/':  
                return int(op1 / op2)  
    raise ValueError("Invalid operator")  
def  
    evaluate_postfix(expression):  
        stack = []  
        tokens = expression.split()  
        for token in tokens:  
            if token.isdigit():  
                stack.append(int(token))  
            else:  
                op2 = stack.pop()  
                op1 = stack.pop()  
                stack.append(result)  
                result = apply_operation(op1, op2, token[0])  
        return stack.pop()  
def  
    main():  
        expr = input("Enter postfix expression (space  
        separated): ")  
        result = evaluate_postfix(expr) print("Result =",  
        result)
```

```
if __name__ ==  
"__main__":  
    main()
```

EXPERIMENT – 10

Browser Back Button Simulation using Stack

AIM:

To simulate a browser back button using stack.

Question:

Write a program that allows the user to visit pages, go back to the previous page and show history using stack operations.

Python Code:

```
def main():
    history = []
    while
        True:
            print("\n1. Visit Page")
            print("2. Back") print("3.
            Show History") print("4.
            Exit")
            choice = input("Enter choice: ").strip()
            if choice ==
                "1":
                    page = input("Enter page name: ").strip()
                    history.append(page) print("Visited:",
                    page)
                elif choice ==
                    "2": if not
                    history:
                        print("No pages in history.")
                    else:
                        last_page = history.pop()
                        print("Going back from:", last_page)
                        if not
```

```
history:  
print("No pages left in history.")  
else:  
print("Current page:", history[-1])  
elif choice ==  
"3": if not  
history:  
else:  
print("History is empty.")  
print("History:", history)  
elif choice ==  
"4":  
break  
print("Exiting browser simulation.")  
else:print("Invali  
d choice.")  
if __name__ ==  
"__main__":  
main()
```

EXPERIMENT – 11

Bubble Sort

AIM:

To implement bubble sort and analyse its time complexity.

Question:

Write a program to sort an array of integers using bubble sort technique and display the sorted array.

Python Code:

```
def bubble_sort(arr):  
    n = len(arr)  for i in  
    range(n - 1): swapped =  
        False  for j in range(n - 1  
        - i): if arr[j] > arr[j  
        + 1]:  
            # swap  
            swapped = True  arr[j], arr[j + 1] = arr[j + 1],  
            arr[j]  
        if not swapped:  
            break  
def main(): n = int(input("Enter number of  
elements: ")) arr = []  
print(f"Enter {n}  
elements:")  for _ in  
range(n):  
    arr.append(int(input()))  
bubble_sort(arr)  
print("Sorted  
array:")  for x in arr:  
    print(x, end=" ")  
print()  if __name__ ==  
    "__main__": main()
```

EXPERIMENT – 12

Binary Search

AIM:

To implement binary search and analyse its time complexity.

Question:

Write a program to perform binary search on a sorted array of integers and find the position of a given key element.

Python Code:

```
def binary_search(arr, key):  
    low = 0  
    high = len(arr) - 1  
    while low <= high:  
        mid = (low + high) // 2  
        if arr[mid] == key:  
            return mid  
        elif arr[mid] < key:  
            low = mid + 1  
        else:  
            high = mid - 1  
    return -1  
def main(): n = int(input("Enter number of  
elements: ")) arr = []  
print(f"Enter {n} sorted  
elements:") for _ in range(n):  
    arr.append(int(input()))  
key = int(input("Enter key to search:  
"))  
index = binary_search(arr, key)  
if index == -
```

```
1:  
print("Element not found.")  
else:  
print("Element found at index:", index)  
if __name__ ==  
"__main__": main()
```