# K.R. MANGALAM UNIVERSITY

DATA STRUCTURES LAB

Course Code: ENCS253

Course Type: Practical

Course Credit: 1.00

Faculty Name: Dr. Swati

Submitted By:

Hridyanshu Singh

Roll No: 2401010159

Section: B

B.Tech CSE

# INDEX

| | |
|---|---|
| | Pop, Peek, Display). |
| 9 | Evaluate Postfix Expression using Stack in Python. |
| 10 | Simulate Browser Back Button using Stack in Python. |
| 11 | Implement Bubble Sort in Python and analyse time complexity. |
| 12 | Implement Binary Search in Python and analyse time complexity. |

## Experiment 1: Inventory Management System in Python

**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 Python 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 ArrayList 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 an ArrayList<Product> 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 ArrayList.

• 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


class InventoryManager:
    inventory = []

    @staticmethod
    def insert_product():
        sku = input("Enter SKU: ")

        # Check if SKU already exists
        for p in InventoryManager.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

        # Validate quantity
        qty_input = input("Enter Quantity: ")
        try:
            qty = int(qty_input)
            if qty <= 0:
                print("Quantity must be positive.")
                return
        except ValueError:
            print("Invalid quantity.")
            return

        InventoryManager.inventory.append(Product(sku, name, qty))
        print("Product inserted successfully.")

    @staticmethod
    def display_inventory():
        if not InventoryManager.inventory:
            print("Inventory is empty.")
            return

        print("SKU\t\tName\t\tQuantity")
        print("-------------------------------------")
        for p in InventoryManager.inventory:
            print(f"{p.sku}\t\t{p.name}\t\t{p.quantity}")

    @staticmethod
    def main():
        while True:
            print("\nInventory Management System")
            print("1. Insert Product")
            print("2. Display Inventory")
            print("3. Exit")
            choice = input("Enter your choice: ")

            if choice == "1":
                InventoryManager.insert_product()
            elif choice == "2":
                InventoryManager.display_inventory()
            elif choice == "3":
                print("Exiting...")
                break
            else:
                print("Invalid choice.")


# Run the program
if __name__ == "__main__":
    InventoryManager.main()
```

## Experiment 2: Inventory Stock Manager – Process Sales & Zero Stock (Python)

**AIM:** To implement an inventory stock manager in Python that processes sales for a given SKU and identifies items with zero stock.

**Question:** Write a Python 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 ArrayList<Item>.

• For processSale(sku, qtySold): search the list for given sku.

• If sku not found: display message.

• If found and quantity >= 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


class InventorySalesManager:

    @staticmethod
    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
```

```python
        if not found:
            print(f"SKU {sku} not found in inventory.")

    @staticmethod
    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(f"Zero stock SKUs: {zero_list}")

        return zero_list


def main():
    inventory = [
        Item(101, 50),
        Item(102, 20),
        Item(103, 0)
    ]

    InventorySalesManager.process_sale(inventory, 101, 30)  # normal sale
    InventorySalesManager.process_sale(inventory, 102, 25)  # insufficient stock
    InventorySalesManager.process_sale(inventory, 104, 10)  # sku not found

    InventorySalesManager.identify_zero_stock(inventory)

    print("Updated Inventory:", end=" ")
    for item in inventory:
        print(f"({item.sku}, {item.quantity})", end=" ")
    print()


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

## Experiment 3: Linear Search in Python

**AIM:** To implement linear search in Python and analyse its best and worst case time complexity.

**Question:** Write a Python 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 in range(len(arr)):
        if arr[i] == 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()
```

**Time Complexity:** Best Case: O(1) when the key is at the first position. Worst Case: O(n) when the key is at the last position or not present in the array.

## Experiment 4: Insertion in Circular Linked List (Beginning & End) in Python

**AIM:** To implement insertion at beginning and at end in a circular linked list using Python.

**Question:** Write a Python 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(temp.data, "->", end=" ")
            temp = temp.next
            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) in Python

**AIM:** To delete a node from the beginning and from the end of a circular linked list using Python.

**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
            self.head.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(temp.data, "->", end=" ")
            temp = temp.next
            if temp == self.head:
                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) in Python

**AIM:** To implement deletion of nodes from the beginning and end of a singly linked list using Python.

**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(temp.data, "->", end=" ")
            temp = temp.next
        print("null")


def main():
    list_obj = SinglyLinkedListDelete()

    list_obj.insert(10)
    list_obj.insert(20)
    list_obj.insert(30)
    list_obj.insert(40)

    print("Original List:")
    list_obj.display()

    list_obj.delete_from_beginning()
    print("After deleting from beginning:")
    list_obj.display()

    list_obj.delete_from_end()
    print("After deleting from end:")
    list_obj.display()


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

## Experiment 7: Circular Queue using Array in Python

**AIM:** To implement a circular queue using array in Python with enqueue, dequeue and display operations.

**Python Code:**

```
class CircularQueue:
    def __init__(self, capacity):
        self.arr = [0] * 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(f"Enqueued: {value}")

    def dequeue(self):
        if self.is_empty():
            print("Queue is empty.")
            return -1

        element = self.arr[self.front]

        if self.front == self.rear:
            self.front = self.rear = -1
        else:
            self.front = (self.front + 1) % self.size

        print(f"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 True:
            print(self.arr[i], end=" ")
            if i == self.rear:
                break
            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 in Python (Push, Pop, Peek, Display)

**AIM:** To implement stack operations push, pop, peek and display using array in Python.

**Python Code:**

```
class ArrayStack:
    def __init__(self, capacity):
        self.capacity = capacity
        self.stack = [0] * 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(f"Pushed {item}")

    def pop(self):
        if self.is_empty():
            print("Stack underflow.")
            return -1
        item = self.stack[self.top]
        self.top -= 1
        print(f"Popped {item}")
        return item

    def peek(self):
        if self.is_empty():
            print("Stack is empty.")
            return -1
        print(f"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")
        choice = input("Enter your choice: ")

        if choice == "1":
            x = int(input("Enter element: "))
            st.push(x)
        elif choice == "2":
            st.pop()
        elif choice == "3":
            st.peek()
        elif choice == "4":
            st.display()
        elif choice == "5":
            print("Exiting.")
            break
        else:
            print("Invalid choice.")


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

# Experiment 9: Evaluate Postfix Expression using Stack in Python

**AIM:** To evaluate a postfix arithmetic expression using stack in Python.

**Python Code:**

```
def apply_operation(op1, op2, operator):
    if operator == '+':
        return op1 + op2
    elif operator == '-':
        return op1 - op2
    elif operator == '*':
        return op1 * op2
    elif operator == '/':
        return op1 // op2  # integer division like Java
    else:
        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()
            result = apply_operation(op1, op2, token)
            stack.append(result)

    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 in Python

**AIM:** To simulate a browser back button using stack in Python.

**Question:** Write a Python 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: ")

        if choice == "1":
            page = input("Enter page name: ")
            history.append(page)
            print("Visited:", page)
        elif choice == "2":
            if not history:
                print("No pages in history.")
            else:
                last = history.pop()
                print("Going back from:", last)
                if not history:
                    print("No pages left in history.")
                else:
                    print("Current page:", history[-1])
        elif choice == "3":
            if not history:
                print("History is empty.")
            else:
                print("History:", history)
        elif choice == "4":
            print("Exiting browser simulation.")
            break
        else:
            print("Invalid choice.")


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

# Experiment 11: Bubble Sort in Python

**AIM:** To implement bubble sort in Python and analyse its time complexity.

**Question:** Write a Python 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]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]  # swap
                swapped = True
        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()
```

**Time Complexity:** Best Case: O(n) when array is already sorted and inner loop breaks early. Worst and Average Case: O(n^2).

# Experiment 12: Binary Search in Python

**AIM:** To implement binary search in Python and analyse its time complexity.

**Question:** Write a Python 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(f"Element found at index: {index}")


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

**Time Complexity:** Best, average and worst case time complexity of binary search is O(log n).