

# AI Assisted Coding (III Year) Assignment

**Name:** B. Rithwik

**HT NO:** 2303A52330

**Batch:** 35

## Lab 11 – Data Structures with AI: Implementing Fundamental Structures

**Week 6 – Monday**

### Lab Objectives

- Use AI to assist in designing and implementing fundamental data structures in Python
- Learn prompt-based design and optimization
- Improve understanding of Lists, Stacks, Queues, Linked Lists, Trees, Graphs, and Hash Tables
- Enhance code quality with comments and documentation

### Task #1 – Stack Implementation

```
class Stack:
    """Implementation of Stack using Python list."""

    def __init__(self):
        self.items = []

    def push(self, value):
        """Add an element to the top of the stack."""
        self.items.append(value)

    def pop(self):
        """Remove and return the top element."""
        if self.is_empty():
            return "Stack is empty"
        return self.items.pop()

    def peek(self):
        """Return the top element without removing it."""
        if self.is_empty():
            return "Stack is empty"
        return self.items[-1]

    def is_empty(self):
        """Check whether stack is empty."""
        return len(self.items) == 0
```

## Task #2 – Queue Implementation

```
class Queue:
    """FIFO Queue Implementation."""

    def __init__(self):
        self.items = []

    def enqueue(self, value):
        """Insert element at rear."""
        self.items.append(value)

    def dequeue(self):
        """Remove element from front."""
        if not self.items:
            return "Queue is empty"
        return self.items.pop(0)

    def peek(self):
        """View front element."""
        if not self.items:
            return "Queue is empty"
        return self.items[0]

    def size(self):
        """Return size of queue."""
        return len(self.items)
```

## Task #3 – Singly Linked List

```
class Node:
    """Node of a singly linked list."""

    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    """Singly linked list Implementation."""

    def __init__(self):
        self.head = None

    def insert(self, value):
        """Insert at end."""
        new_node = Node(value)
        if not self.head:
            self.head = new_node
            return

        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node

    def display(self):
        """Display linked list."""
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
```

## Task #4 – Binary Search Tree

```
class BST:
    """Binary Search Tree Implementation."""

    def __init__(self, value=None):
        self.value = value
        self.left = None
        self.right = None

    def insert(self, value):
        """Recursive insert."""
        if self.value is None:
            self.value = value
        elif value < self.value:
            if self.left:
                self.left.insert(value)
            else:
                self.left = BST(value)
        else:
            if self.right:
                self.right.insert(value)
            else:
                self.right = BST(value)

    def inorder(self):
        """In-order traversal."""
        if self.left:
            self.left.inorder()
        print(self.value, end=" ")
        if self.right:
            self.right.inorder()
```

## Task #5 – Hash Table with Chaining

```
class HashTable:
    """Hash table using chaining."""

    def __init__(self, size=10):
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash(self, key):
        return hash(key) % self.size

    def insert(self, key, value):
        """Insert key-value."""
        index = self._hash(key)
        for pair in self.table[index]:
            if pair[0] == key:
                pair[1] = value
                return
        self.table[index].append([key, value])

    def search(self, key):
        """Search value by key."""
        index = self._hash(key)
        for pair in self.table[index]:
            if pair[0] == key:
                return pair[1]
        return None

    def delete(self, key):
        """Delete key."""
        index = self._hash(key)
        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:
                self.table[index].pop(i)
                return
```

## Task #6 – Graph (Adjacency List)

```
class Graph:
    """Graph using adjacency list."""

    def __init__(self):
        self.graph = {}

    def add_vertex(self, vertex):
        if vertex not in self.graph:
            self.graph[vertex] = []

    def add_edge(self, v1, v2):
        self.graph.setdefault(v1, []).append(v2)

    def display(self):
        for v in self.graph:
            print(v, '->', self.graph[v])
```

## Task #7 – Priority Queue

```
import heapq

class PriorityQueue:
    """Priority Queue using heap."""

    def __init__(self):
        self.heap = []

    def enqueue(self, value, priority):
        """Lower number = higher priority."""
        heapq.heappush(self.heap, (priority, value))

    def dequeue(self):
        if not self.heap:
            return "Empty"
        return heapq.heappop(self.heap)[1]

    def display(self):
        print(self.heap)
```

# Task #8 – Deque

```
from collections import deque

class DequeDS:
    """Double-ended queue."""
    def __init__(self):
        self.dq = deque()

    def insert_front(self, value):
        self.dq.appendleft(value)

    def insert_rear(self, value):
        self.dq.append(value)

    def remove_front(self):
        if not self.dq:
            return "Empty"
        return self.dq.popleft()

    def remove_rear(self):
        if not self.dq:
            return "Empty"
        return self.dq.pop()
```

# Task #9 – Campus Resource Management System

Feature Mapping Table

| Feature            | Data Structure Justification |                                                                                       |
|--------------------|------------------------------|---------------------------------------------------------------------------------------|
| Student Attendance | Deque                        | Students enter and exit from both ends, making deque efficient for real-time logging. |
| Event Registration | Hash Table                   | Fast search, insertion, and deletion based on student ID.                             |
| Library Borrowing  | Linked List                  | Dynamic insertion and removal of books without shifting elements.                     |
| Bus Scheduling     | Graph                        | Routes and stops form a network structure.                                            |
| Cafeteria Orders   | Queue                        | Orders processed in arrival order (FIFO).                                             |

# Implementation – Cafeteria Order Queue

```
class CafeteriaQueue:
    """Queue system for cafeteria orders."""
    def __init__(self):
        self.orders = []

    def place_order(self, student):
        """Add new order."""
        self.orders.append(student)

    def serve_order(self):
        """Serve next order."""
        if not self.orders:
            return "No orders"
        return self.orders.pop(0)

    def display(self):
        print("Pending:", self.orders)

# Example
cq = CafeteriaQueue()
cq.place_order("Akshay")
cq.place_order("Rithwik")
cq.display()
print("Served:", cq.serve_order())

''' Pending: ['Akshay', 'Rithwik']
    Served: Akshay
```

## Task #10 – Smart E-Commerce Platform

### Feature Mapping Table

| Feature              | Data Structure | Justification                                  |
|----------------------|----------------|------------------------------------------------|
| Shopping Cart        | Linked List    | Dynamic addition and removal without shifting. |
| Order Processing     | Queue          | Orders handled sequentially.                   |
| Top-Selling Products | Priority Queue | Highest sales given priority.                  |
| Product Search       | Hash Table     | Fast lookup by product ID.                     |
| Delivery Planning    | Graph          | Represents routes between locations.           |

### Implementation – Product Search using Hash Table

```
class ProductSearch:
    """Fast lookup of products."""
    def __init__(self):
        self.products = {}
    def add_product(self, product_id, name):
        """Add product"""
        self.products[product_id] = name
    def search_product(self, product_id):
        """Search by ID"""
        return self.products.get(product_id, "Not found")
    def remove_product(self, product_id):
        """Delete product"""
        if product_id in self.products:
            del self.products[product_id]

# Example
ps = ProductSearch()
ps.add_product(101, "Laptop")
ps.add_product(102, "Mobile")
print(ps.search_product(101))
```

## Conclusion

This lab demonstrated how AI can assist in:

- Designing and implementing core data structures
- Improving documentation and readability
- Selecting optimal structures for real-world applications
- Enhancing efficiency and scalability