SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE			DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**		**Academic Year:**2025-2026
Course Coordinator Name		Venkataramana Veeramsetty		
Instructor(s) Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding	
Year/Sem	II/I	**Regulation**	R24	
Date and Day of Assignment	Week6 - Monday	**Time(s)**		
Duration	2 Hours	**Applicable to Batches**		
AssignmentNumber:11.1(Present assignment number)/**24**(Total number of assignments)				

Q.No.	Question	*Expected Time to complete*
1	**Lab 11 – Data Structures with AI: Implementing Fundamental Structures** **Lab Objectives** • Use AI to assist in designing and implementing fundamental data structures in Python. • Learn how to prompt AI for structure creation, optimization, and documentation. • Improve understanding of Lists, Stacks, Queues, Linked Lists, Trees, Graphs, and Hash Tables.	Week6 - Monday

- Enhance code quality with AI-generated comments and performance suggestions.

Task Description #1 – Stack Implementation

Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.

Sample Input Code:

class Stack:

pass

Expected Output:

- A functional stack implementation with all required methods and docstrings.

ANSWER:

```python
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        """Checks if the stack is empty."""
        return not self.items

    def push(self, item):
        """Adds an item to the top of the stack."""
        self.items.append(item)

    def pop(self):
        """Removes and returns the item from the top of the stack.
        Returns None if the stack is empty.
        """
        if not self.is_empty():
            return self.items.pop()
        else:
            return None

    def peek(self):
        """Returns the item at the top of the stack without removing it.
        Returns None if the stack is empty.
        """
        if not self.is_empty():
            return self.items[-1]
        else:
            return None

# Example usage:
stack = Stack()
print(f"Is the stack empty? {stack.is_empty()}")
stack.push(10)
stack.push(20)
stack.push(30)
print(f"Is the stack empty? {stack.is_empty()}")
print(f"Top element (peek): {stack.peek()}")
print(f"Popped element: {stack.pop()}")
print(f"Top element after pop (peek): {stack.peek()}")
print(f"Popped element: {stack.pop()}")
print(f"Popped element: {stack.pop()}")
print(f"Popped element: {stack.pop()}") # Trying to pop from an empty stack
print(f"Is the stack empty? {stack.is_empty()}")
```

```
Is the stack empty? True
Is the stack empty? False
Top element (peek): 30
Popped element: 30
Top element after pop (peek): 20
Popped element: 20
Popped element: 10
Popped element: None
Is the stack empty? True
```

	Task Description #2 – Queue Implementation Task: Use AI to implement a Queue using Python lists. Sample Input Code: class Queue: pass Expected Output: • FIFO-based queue class with enqueue, dequeue, peek, and size methods. ANSWER:	

```python
class Queue:
    pass # Placeholder as requested

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

    def is_empty(self):
        """Checks if the queue is empty."""
        return not self.items

    def enqueue(self, item):
        """Adds an item to the rear of the queue."""
        self.items.append(item)

    def dequeue(self):
        """Removes and returns the item from the front of the queue.
        Returns None if the queue is empty.
        """
        if not self.is_empty():
            return self.items.pop(0)
        else:
            return None

    def peek(self):
        """Returns the item at the front of the queue without removing it.
        Returns None if the queue is empty.
        """
        if not self.is_empty():
            return self.items[0]
        else:
            return None

# Example usage:
queue = Queue()
print(f"Is the queue empty? {queue.is_empty()}")
queue.enqueue(10)
queue.enqueue(20)
queue.enqueue(30)
print(f"Is the queue empty? {queue.is_empty()}")
print(f"Front element (peek): {queue.peek()}")
print(f"Dequeued element: {queue.dequeue()}")
print(f"Front element after dequeue (peek): {queue.peek()}")
print(f"Dequeued element: {queue.dequeue()}")
print(f"Dequeued element: {queue.dequeue()}")
print(f"Dequeued element: {queue.dequeue()}") # Trying to dequeue from an empty queue
print(f"Is the queue empty? {queue.is_empty()}")
```

```
Is the queue empty? True
Is the queue empty? False
Front element (peek): 10
Dequeued element: 10
Front element after dequeue (peek): 20
Dequeued element: 20
Dequeued element: 30
Dequeued element: None
Is the queue empty? True
```

Task Description #3 – Linked List

Task: Use AI to generate a Singly Linked List with insert and display methods.

Sample Input Code:

```python
class Node:
    pass
```

class LinkedList:
    pass

Expected Output:

- A working linked list implementation with clear method documentation.

ANSWER:

```python
class Node:
    """Represents a node in a singly linked list."""
    def __init__(self, data=None):
        self.data = data
        self.next = None

class SinglyLinkedList:
    """Represents a singly linked list."""
    def __init__(self):
        self.head = None

    def insert(self, data):
        """Inserts a new node at the end of the linked list."""
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def display(self):
        """Displays the elements of the linked list."""
        elements = []
        current = self.head
        while current:
            elements.append(current.data)
            current = current.next
        print(" -> ".join(map(str, elements)))

# Example usage:
linked_list = SinglyLinkedList()
linked_list.insert(10)
linked_list.insert(20)
linked_list.insert(30)
print("Linked List:")
linked_list.display()
```

```
Linked List:
10 -> 20 -> 30
```

Task Description #4 – Binary Search Tree (BST)

Task: Use AI to create a BST with insert and in-order traversal methods.

Sample Input Code:

class BST:
    pass

Expected Output:

- BST implementation with recursive insert and traversal methods.

ANSWER:

```python
class TreeNode:
    """Represents a node in a Binary Search Tree."""
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

class BinarySearchTree:
    """Represents a Binary Search Tree."""
    def __init__(self):
        self.root = None

    def insert(self, key):
        """Inserts a new node with the given key into the BST."""
        self.root = self._insert_recursive(self.root, key)

    def _insert_recursive(self, root, key):
        """Helper function for recursive insertion."""
        if root is None:
            return TreeNode(key)
        if key < root.key:
            root.left = self._insert_recursive(root.left, key)
        elif key > root.key:
            root.right = self._insert_recursive(root.right, key)
        return root

    def in_order_traversal(self):
        """Performs an in-order traversal of the BST."""
        elements = []
        self._in_order_traversal_recursive(self.root, elements)
        print(" -> ".join(map(str, elements)))

    def _in_order_traversal_recursive(self, root, elements):
        """Helper function for recursive in-order traversal."""
        if root:
            self._in_order_traversal_recursive(root.left, elements)
            elements.append(root.key)
            self._in_order_traversal_recursive(root.right, elements)

# Example usage:
bst = BinarySearchTree()
bst.insert(50)
bst.insert(30)
bst.insert(20)
bst.insert(40)
bst.insert(70)
bst.insert(60)
bst.insert(80)

print("In-order traversal of the BST:")
bst.in_order_traversal()
```

```
In-order traversal of the BST:
20 -> 30 -> 40 -> 50 -> 60 -> 70 -> 80
```

Task Description #5 – Hash Table

Task: Use AI to implement a hash table with basic insert, search, and delete methods.

Sample Input Code:

class HashTable:

  pass

Expected Output:

- Collision handling using chaining, with well-commented methods.

ANSWER

```python
class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [[] for _ in range(self.size)]

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

    def insert(self, key, value):
        """Inserts a key-value pair into the hash table."""
        index = self._hash(key)
        for pair in self.table[index]:
            if pair[0] == key:
                pair[1] = value  # Update value if key exists
                return
        self.table[index].append([key, value])

    def search(self, key):
        """Searches for a key in the hash table and returns its value."""
        index = self._hash(key)
        for pair in self.table[index]:
            if pair[0] == key:
                return pair[1]
        return None  # Key not found

    def delete(self, key):
        """Deletes a key-value pair from the hash table."""
        index = self._hash(key)
        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:
                del self.table[index][i]
                return True  # Deletion successful
        return False  # Key not found

# Example usage:
ht = HashTable(10)
ht.insert("apple", 1)
ht.insert("banana", 2)
ht.insert("cherry", 3)

print(f"Search 'banana': {ht.search('banana')}")
print(f"Search 'grape': {ht.search('grape')}")

ht.delete("banana")
print(f"Search 'banana' after deletion: {ht.search('banana')}")
```

```
Search 'banana': 2
Search 'grape': None
Search 'banana' after deletion: None
```

Task Description #6 – Graph Representation

Task: Use AI to implement a graph using an adjacency list.

Sample Input Code:

```python
class Graph:
    pass
```

Expected Output:

- Graph with methods to add vertices, add edges, and display connections.

ANSWER:

```python
class Graph:
    def __init__(self):
        """Initializes an empty graph using an adjacency list."""
        self.adj_list = {}

    def add_vertex(self, vertex):
        """Adds a vertex to the graph if it doesn't already exist."""
        if vertex not in self.adj_list:
            self.adj_list[vertex] = []

    def add_edge(self, u, v):
        """Adds an edge between vertices u and v."""
        # Assuming an undirected graph, add edges in both directions
        if u in self.adj_list and v in self.adj_list:
            self.adj_list[u].append(v)
            self.adj_list[v].append(u)
        else:
            print(f"One or both vertices ({u}, {v}) not found in the graph.")


    def display(self):
        """Displays the graph's adjacency list."""
        for vertex in self.adj_list:
            print(f"{vertex}: {self.adj_list[vertex]}" )

# Example usage:
graph = Graph()

graph.add_vertex("A")
graph.add_vertex("B")
graph.add_vertex("C")
graph.add_vertex("D")

graph.add_edge("A", "B")
graph.add_edge("A", "C")
graph.add_edge("B", "D")
graph.add_edge("C", "D")

print("Graph (Adjacency List):")
graph.display()

# Example of adding an edge with a non-existent vertex
graph.add_edge("A", "E")
```

```
Graph (Adjacency List):
A: ['B', 'C']
B: ['A', 'D']
C: ['A', 'D']
D: ['B', 'C']
One or both vertices (A, E) not found in the graph.
```

Task Description #7 – Priority Queue

Task: Use AI to implement a priority queue using Python’s heapq module.

Sample Input Code:

class PriorityQueue:

pass

Expected Output:

- Implementation with enqueue (priority), dequeue (highest priority), and display methods.

ANSWER:

```python
import heapq

class PriorityQueue:
    """Implements a priority queue using the heapq module."""
    def __init__(self):
        self._queue = []
        self._index = 0

    def push(self, item, priority):
        """Add a new item to the priority queue with a given priority."""
        # Use a tuple (priority, index, item) to handle items with the same priority
        heapq.heappush(self._queue, (-priority, self._index, item))
        self._index += 1

    def pop(self):
        """Remove and return the item with the highest priority."""
        if not self._queue:
            return None  # Queue is empty
        return heapq.heappop(self._queue)[-1] # Return the item, ignoring priority and index

    def is_empty(self):
        """Check if the priority queue is empty."""
        return not self._queue

# Example usage:
pq = PriorityQueue()
pq.push('task1', 3)
pq.push('task2', 1)
pq.push('task3', 2)

print("Priority Queue (highest priority first):")
while not pq.is_empty():
    print(pq.pop())

pq.push('task4', 2)
pq.push('task5', 2)
pq.push('task6', 1)

print("\nPriority Queue after adding more items:")
while not pq.is_empty():
    print(pq.pop())
```

```
Priority Queue (highest priority first):
task1
task3
task2

Priority Queue after adding more items:
task4
task5
task6
```

Task Description #8 – Deque

Task: Use AI to implement a double-ended queue using collections.deque.

Sample Input Code:

class DequeDS:

    pass

Expected Output:

- Insert and remove from both ends with docstrings.

ANSWER:

```python
from collections import deque

# Create a deque
dq = deque(['a', 'b', 'c'])
print(f"Initial deque: {dq}")

# Add elements to the right end
dq.append('d')
dq.append('e')
print(f"Deque after appending elements to the right: {dq}")

# Add elements to the left end
dq.appendleft('z')
dq.appendleft('y')
print(f"Deque after appending elements to the left: {dq}")

# Remove elements from the right end
right_popped = dq.pop()
print(f"Popped from right: {right_popped}")
print(f"Deque after popping from right: {dq}")

# Remove elements from the left end
left_popped = dq.popleft()
print(f"Popped from left: {left_popped}")
print(f"Deque after popping from left: {dq}")

# Peek at elements (access without removing)
print(f"Element at the right end: {dq[-1]}")
print(f"Element at the left end: {dq[0]}")

# Check if deque is empty
print(f"Is deque empty? {not dq}")

# Clear the deque
dq.clear()
print(f"Deque after clearing: {dq}")
print(f"Is deque empty? {not dq}")
```

```
Initial deque: deque(['a', 'b', 'c'])
Deque after appending elements to the right: deque(['a', 'b', 'c', 'd', 'e'])
Deque after appending elements to the left: deque(['y', 'z', 'a', 'b', 'c', 'd', 'e'])
Popped from right: e
Deque after popping from right: deque(['y', 'z', 'a', 'b', 'c', 'd'])
Popped from left: y
Deque after popping from left: deque(['z', 'a', 'b', 'c', 'd'])
Element at the right end: d
Element at the left end: z
Is deque empty? False
Deque after clearing: deque([])
Is deque empty? True
```

Task Description #9 – AI-Generated Data Structure Comparisons

Task: Use AI to generate a comparison table of different data structures (stack, queue, linked list, etc.) including time complexities.

Sample Input Code:

No code, prompt AI for a data structure comparison table

Expected Output:

- A markdown table with structure names, operations, and complexities.

ANSWER:

Task Description #10 Real-Time Application Challenge – Choose the Right Data Structure

Scenario:

Your college wants to develop a Campus Resource Management System that handles:

1. Student Attendance Tracking – Daily log of students entering/exiting the campus.
2. Event Registration System – Manage participants in events with quick search and removal.
3. Library Book Borrowing – Keep track of available books and their due dates.
4. Bus Scheduling System – Maintain bus routes and stop connections.
5. Cafeteria Order Queue – Serve students in the order they arrive.

Student Task:

- For each feature, select the most appropriate data structure from the list below:
    - Stack
    - Queue
    - Priority Queue
    - Linked List
    - Binary Search Tree (BST)
    - Graph
    - Hash Table
    - Deque
- Justify your choice in 2–3 sentences per feature.
- Implement one selected feature as a working Python program with AI-assisted code generation.

Expected Output:

- A table mapping feature → chosen data structure → justification.
- A functional Python program implementing the chosen feature with comments and docstrings.

☑ Deliverables (For All Tasks)

1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.
4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanation.
6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.