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.	Week 6 - Monday

- Improve understanding of Lists, Stacks, Queues, Linked Lists, Trees, Graphs, and Hash Tables.
- 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.

PROMPT: Generate a Python Stack class starting from Implement the methods push(item), pop(), peek(), and is_empty(). Each method must include clear docstrings explaining its purpose.

CODE:

```
# Create a new stack
my_stack = Stack()

# Check if the stack is empty
print(f"Is the stack empty? {my_stack.is_empty()}")

# Push some items onto the stack
my_stack.push(10)
my_stack.push(20)
my_stack.push(30)

# Check if the stack is empty again
print(f"Is the stack empty after pushing items? {my_stack.is_empty()}")

# Peek at the top item
print(f"Top item on the stack: {my_stack.peek()}")

# Pop items from the stack
print(f"Popped item: {my_stack.pop()}")
print(f"Popped item: {my_stack.pop()}")

# Peek at the new top item
print(f"Top item after popping: {my_stack.peek()}")

# Pop the last item
print(f"Popped item: {my_stack.pop()}")

# Check if the stack is empty after popping all items
print(f"Is the stack empty after popping all items? {my_stack.is_empty()}")

# Attempt to pop from an empty stack (this will raise an error)
try:
    my_stack.pop()
except IndexError as e:
    print(f"Error when popping from empty stack: {e}")
```

```
Is the stack empty? True
Is the stack empty after pushing items? False
Top item on the stack: 30
Popped item: 30
Popped item: 20
Top item after popping: 10
Popped item: 10
Is the stack empty after popping all items? True
Error when popping from empty stack: pop from empty stack
```

OBSERVATION: Overall, the code successfully defines and demonstrates a basic Stack data structure in Python.

- The Stack class provides the essential methods (push, pop, peek, is_empty) required for stack operations.
- The implementation using a Python list is simple and effective for this basic example.
- The execution code clearly illustrates how to use the Stack class and confirms that the methods behave according to the LIFO principle.
- Basic error handling for popping from an empty stack is included and demonstrated.

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.

PROMPT: Generate a Python Queue class using lists with FIFO behavior, implementing enqueue, dequeue, peek, and size methods with docstrings and example usage.
CODE:

```python
# Create a new queue
my_queue = Queue()

# Check if the queue is empty
print(f"Is the queue empty? {my_queue.is_empty()}")

# Add some elements to the queue
my_queue.enqueue(10)
my_queue.enqueue(20)
my_queue.enqueue(30)

# Check the size of the queue
print(f"Size of the queue after enqueuing: {my_queue.size()}")

# Peek at the front element
print(f"Front item on the queue: {my_queue.peek()}")

# Remove an element from the queue
print(f"Dequeued item: {my_queue.dequeue()}")

# Peek at the new front element
print(f"New front item after dequeuing: {my_queue.peek()}")

# Remove another element
print(f"Dequeued item: {my_queue.dequeue()}")

# Check the size after removing elements
print(f"Size of the queue after dequeuing: {my_queue.size()}")

# Remove the remaining element
print(f"Dequeued item: {my_queue.dequeue()}")
```

```
# Remove the remaining element
print(f"Dequeued item: {my_queue.dequeue()}")

# Check if the queue is empty
print(f"Is the queue empty after dequeuing all items? {my_queue.is_empty()}")

# Attempt to dequeue from an empty queue
try:
    my_queue.dequeue()
except IndexError as e:
    print(f"Error when dequeuing from empty queue: {e}")
```

```
Is the queue empty? True
Size of the queue after enqueuing: 3
Front item on the queue: 10
Dequeued item: 10
New front item after dequeuing: 20
Dequeued item: 20
Size of the queue after dequeuing: 1
Dequeued item: 30
Is the queue empty after dequeuing all items? True
Error when dequeuing from empty queue: dequeue from empty queue
```

OBSERVATION: The code provides a correct and well-explained implementation of a basic Queue using a Python list, with the caveat regarding the potential performance issue of pop(0) for very large queues.

Task Description #3 – Linked List

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

class Node:

pass

class LinkedList:

pass

Expected Output:

- A working linked list implementation with clear method documentation.

PROMPT: Generate a code block to demonstrate the usage of the LinkedList class by creating an instance, inserting elements, and displaying the list.

CODE:

```
# Create a new linked list
my_list = LinkedList()

# Insert elements into the list
my_list.insert(30)
my_list.insert(20)
my_list.insert(10)

# Display the linked list
print("Linked list after insertions:")
my_list.display()
```

```
Linked list after insertions:
10 -> 20 -> 30 -> None
```

OBSERVATION: The provided code successfully implements a basic singly linked list.

- It clearly defines the Node and LinkedList structures.
- The insert method correctly adds elements to the beginning of the list, demonstrating the manipulation of the head pointer.

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.

PROMPT: Generate a Python BST class with recursive insert and inorder_traversal methods, starting from class BST: pass, including docstrings.

CODE:

```
# Create a new BST
my_bst = BST()

# Insert elements into the BST
print("Inserting elements: 50, 30, 70, 20, 40, 60, 80")
my_bst.insert(50)
my_bst.insert(30)
my_bst.insert(70)
my_bst.insert(20)
my_bst.insert(40)
my_bst.insert(60)
my_bst.insert(80)

# Perform and display inorder traversal
my_bst.inorder_traversal()
```

```
Inserting elements: 50, 30, 70, 20, 40, 60, 80
Inorder Traversal: 20 30 40 50 60 70 80
```

OBSERVATION: The BST implementation successfully demonstrates recursive insertion and that in order traversal yields a sorted sequence of elements.

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.

PROMPT: Generate a python code for hash table which includes insert, search, and delete methods from user input.

CODE:

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

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

    def insert(self, key, value):
        index = self._hash_function(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, value) # Update value if key exists
                return
        self.table[index].append((key, value)) # Add new key-value pair

    def search(self, key):
        index = self._hash_function(key)
        for k, v in self.table[index]:
            if k == key:
                return v
        return None # Key not found

    def delete(self, key):
        index = self._hash_function(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                del self.table[index][i]
                return
        print(f"Key '{key}' not found in the hash table.")
```

```python
# Example usage:
if __name__ == "__main__":
    # Create a hash table with size 10
    ht = HashTable(10)

    # Insert key-value pairs
    ht.insert("apple", 1)
    ht.insert("banana", 2)
    ht.insert("cherry", 3)
    ht.insert("apple", 10) # Update value for existing key

    # Search for keys
    print("Search results:")
    print(f"apple: {ht.search('apple')}")
    print(f"banana: {ht.search('banana')}")
    print(f"date: {ht.search('date')}") # Key not found

    # Delete a key
    print("\nDeleting 'banana':")
    ht.delete("banana")
    print(f"banana after deletion: {ht.search('banana')}")

    # Attempt to delete a non-existent key
    print("\nAttempting to delete 'grape':")
    ht.delete("grape")

    # Print the hash table (for demonstration purposes)
    print("\nHash Table:")
    for i, bucket in enumerate(ht.table):
        print(f"Bucket {i}: {bucket}")
```

```
Search results:
apple: 10
banana: 2
date: None

Deleting 'banana':
banana after deletion: None

Attempting to delete 'grape':
Key 'grape' not found in the hash table.

Hash Table:
Bucket 0: []
Bucket 1: []
Bucket 2: [('cherry', 3)]
Bucket 3: []
Bucket 4: []
Bucket 5: []
Bucket 6: []
Bucket 7: []
Bucket 8: [('apple', 10)]
Bucket 9: []
```

OBSERVATION: it's a good starting point for understanding hash tables, but it could be enhanced for real-world applications**.**

Task Description #6 – Graph Representation

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

Sample Input Code:

class Graph: pass

Expected Output:

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

PROMPT: Write a python code to implement a graph using a adjacency list and graph with methods to add vertices, edges, and dispay

CODE:

```
class Graph:
    def __init__(self):
        self.graph = {}

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

    def add_edge(self, vertex1, vertex2):
        if vertex1 in self.graph and vertex2 in self.graph:
            self.graph[vertex1].append(vertex2)
            # If it's an undirected graph, uncomment the next line:
            # self.graph[vertex2].append(vertex1)
        else:
            print(f"One or both vertices ({vertex1}, {vertex2}) not in graph.")

    def display(self):
        for vertex, edges in self.graph.items():
            print(f"{vertex}: {edges}")
```

OBSERVATION:

```python
# Create a graph instance
my_graph = Graph()

# Add vertices
my_graph.add_vertex("A")
my_graph.add_vertex("B")
my_graph.add_vertex("C")
my_graph.add_vertex("D")

# Add edges
my_graph.add_edge("A", "B")
my_graph.add_edge("A", "C")
my_graph.add_edge("B", "D")
my_graph.add_edge("C", "D")

# Display the graph
my_graph.display()
```

```
A: ['B', 'C']
B: ['D']
C: ['D']
D: []
```

OBSERVATION: This code provides a basic framework for creating and managing a directed graph using an adjacency list, allowing you to add vertices and edges and visualize the graph's connections.

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.

PROMPT: Generate a python code to implement a priority queue using pythons heapq module and methods like enqueue, dequeue, and display methods

CODE:

```python
import heapq

class PriorityQueue:
    def __init__(self):
        self._queue = []
        self._index = 0 # To ensure elements with same priority are dequeued in insertion order

    def enqueue(self, item, priority):
        # Items are tuples of (priority, index, item)
        heapq.heappush(self._queue, (-priority, self._index, item))
        self._index += 1

    def dequeue(self):
        if not self._queue:
            return None # Or raise an exception
        return heapq.heappop(self._queue)[-1] # Return the item (last element of the tuple)

    def display(self):
        # Displaying the raw heap might not be intuitive due to heap property
        # You might want to display sorted by priority (but this changes the heap)
        # A simple display of the current heap state:
        print("Priority Queue (internal heap representation):", self._queue)

    def peek(self):
        if not self._queue:
            return None
        return self._queue[0][-1] # Return the item of the highest priority element without removing

    def is_empty(self):
        return len(self._queue) == 0
```

```python
# Create a PriorityQueue instance
pq = PriorityQueue()

# Enqueue elements with priorities
pq.enqueue("Task A", 3)
pq.enqueue("Task B", 1)
pq.enqueue("Task C", 2)
pq.enqueue("Task D", 3) # Same priority as Task A

# Display the internal representation (optional, for understanding)
pq.display()

# Dequeue elements (should be in order of priority)
print("\nDequeuing elements:")
while not pq.is_empty():
    print(pq.dequeue())
```

```
Priority Queue (internal heap representation): [(-3, 0, 'Task A'), (-3, 3, 'Task D'), (-2, 2, 'Task C'), (-1, 1, 'Task B')]

Dequeuing elements:
Task A
Task D
Task C
Task B
```

OBSERVATION: This code leverages the heapq module to efficiently manage a collection of items where each item has a priority, allowing for quick retrieval of the item with the highest priority. The use of the index in the tuple ensures that items with the same priority are handled in a First-In, First-Out (FIFO) manner.

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. **PROMPT: Write a python code for to implement a double ended queue using collections dequeue with insert and remove from both ends with docstrings** **CODE:**	

```python
from collections import deque

class Deque:
    """
    A double-ended queue (deque) implementation using collections.deque.
    """
    def __init__(self):
        """
        Initializes an empty deque.
        """
        self._deque = deque()

    def add_front(self, item):
        """
        Adds an item to the front of the deque.

        Args:
            item: The item to add.
        """
        self._deque.appendleft(item)

    def add_rear(self, item):
        """
        Adds an item to the rear of the deque.

        Args:
            item: The item to add.
        """
        self._deque.append(item)

    def remove_front(self):
        """
        Removes and returns the item from the front of the deque.

        Returns:
            The item removed from the front, or None if the deque is empty.
        """
```

```
        if not self.is_empty():
            return self._deque[0]
        return None

    def peek_rear(self):
        """
        Returns the item at the rear of the deque without removing it.

        Returns:
            The item at the rear, or None if the deque is empty.
        """
        if not self.is_empty():
            return self._deque[-1]
        return None

    def is_empty(self):
        """
        Checks if the deque is empty.

        Returns:
            True if the deque is empty, False otherwise.
        """
        return len(self._deque) == 0

    def size(self):
        """
        Returns the number of items in the deque.

        Returns:
            The number of items in the deque.
        """
        return len(self._deque)

    def display(self):
        """
        Displays the items in the deque.
        """
        print("Deque:", list(self._deque))
```

```python
# Create a Deque instance
my_deque = Deque()

# Add elements to the rear
my_deque.add_rear(10)
my_deque.add_rear(20)
my_deque.display()

# Add elements to the front
my_deque.add_front(5)
my_deque.add_front(1)
my_deque.display()

# Remove from the front
removed_front = my_deque.remove_front()
print(f"\nRemoved from front: {removed_front}")
my_deque.display()

# Remove from the rear
removed_rear = my_deque.remove_rear()
print(f"Removed from rear: {removed_rear}")
my_deque.display()

# Peek at the front and rear
print(f"\nPeek at front: {my_deque.peek_front()}")
print(f"Peek at rear: {my_deque.peek_rear()}")

# Check if empty and size
print(f"\nIs deque empty? {my_deque.is_empty()}")
print(f"Deque size: {my_deque.size()}")
```

```
Deque: [10, 20]
Deque: [1, 5, 10, 20]

Removed from front: 1
Deque: [5, 10, 20]
Removed from rear: 20
Deque: [5, 10]

Peek at front: 5
Peek at rear: 10

Is deque empty? False
Deque size: 2
```

OBSERVATION: This code provides a user-friendly class interface for a deque, leveraging the optimized implementation provided by collections.deque for efficient operations at both ends. The docstrings clearly explain the purpose and usage of each method.

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.

PROMPT: Generate a python code for example taking one, compare with different data structures like stack, queue,linked list etc

CODE:

[2] ✓ 0s

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

    def enqueue(self, item):
        self.items.append(item)
        print(f"Enqueued: {item}") # Added print for clarity

    def dequeue(self):
        if not self.is_empty():
            item = self.items.pop(0)
            print(f"Dequeued: {item}") # Added print for clarity
            return item
        else:
            print("Queue is empty") # Added print for clarity
            return "Queue is empty"

    def peek(self):
        if not self.is_empty():
            item = self.items[0]
            print(f"Peek: {item}") # Added print for clarity
            return item
        else:
            print("Queue is empty") # Added print for clarity
            return "Queue is empty"
```

[2] ✓ 0s

```
        def is_empty(self):
            return len(self.items) == 0

        def size(self):
            return len(self.items)

# Example Usage:
queue = Queue()
queue.enqueue(10)
queue.enqueue(20)
queue.peek()
queue.dequeue()
queue.is_empty()
queue.size()
queue.dequeue()
queue.is_empty()
queue.size()
```

```
Enqueued: 10
Enqueued: 20
Peek: 10
Dequeued: 10
Dequeued: 20
0
```

OBSERVATION: This code provides basic implementations for Stack, Queue, and Linked List data structures in Python, each with methods for common operations like adding, removing, and checking emptiness, demonstrating their different underlying principles and operational characteristics.

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. **PROMPT:** My college wants to develop a campus resource management system that handles the student attendence tracking, event registration system, library book borrowing, bus scheduling system, cafeteria order queue, here for each feature select the data struture like stack, queue, priority queue, linkd list, binary search tree, graph, hashtable, deque now generate a python code which includes all this from user input. **CODE:**	

```
from collections import deque
import heapq

class StudentAttendance:
    def __init__(self):
        self.attendance_records = {}  # Using a dictionary (Hash Table)

    def record_attendance(self, student_id, date, status):
        pass # Placeholder

    def get_attendance(self, student_id):
        pass # Placeholder

class EventRegistration:
    def __init__(self):
        self.events = {}  # Using a dictionary (Hash Table)

    def add_event(self, event_id, event_details):
        pass # Placeholder

    def register_student(self, event_id, student_id, registration_details
        pass # Placeholder

    def get_registered_students(self, event_id):
        pass # Placeholder
```

```
    def get_neighbors(self, stop_name):
        pass # Placeholder

class CafeteriaOrderQueue:
    def __init__(self):
        self.order_queue = deque()  # Using a deque

    def enqueue_order(self, order_details):
        pass # Placeholder

    def dequeue_order(self):
        pass # Placeholder

    def peek_order(self):
        pass # Placeholder

    def is_empty(self):
        pass # Placeholder

class CampusSystem:
    def __init__(self):
        self.student_registry = {}  # Placeholder for Central Student Reg
        self.attendance_system = StudentAttendance()
        self.event_system = EventRegistration()
        self.library_system = LibraryBorrowing()
        self.bus_system = BusScheduling()
```

```
    def add_student_to_registry(self, student_id, student_details):
        pass # Placeholder

    def get_student_from_registry(self, student_id):
        pass # Placeholder

    # Placeholder methods to interact with the feature systems
    def record_student_attendance(self, student_id, date, status):
        self.attendance_system.record_attendance(student_id, date, status

    def register_student_for_event(self, event_id, student_id, registrati
        self.event_system.register_student(event_id, student_id, registra

    def student_borrows_book(self, isbn, student_id):
        self.library_system.borrow_book(isbn, student_id)

    def student_returns_book(self, isbn, student_id):
        self.library_system.return_book(isbn, student_id)

    def add_bus_stop(self, stop_name):
        self.bus_system.add_stop(stop_name)

    def add_bus_route(self, stop1, stop2):
        self.bus_system.add_route(stop1, stop2)
```

```
from collections import deque

class StudentAttendance:
    def __init__(self):
        self.attendance_records = {}  # Using a dictionary (Hash Table)

    def record_attendance(self, student_id, date, status):
        if student_id not in self.attendance_records:
            self.attendance_records[student_id] = {}
        self.attendance_records[student_id][date] = status
        print(f"Attendance recorded for student {student_id} on {date}: {

    def get_attendance(self, student_id):
        if student_id in self.attendance_records:
            print(f"Attendance records for student {student_id}:")
            for date, status in self.attendance_records[student_id].items
                print(f"  {date}: {status}")
            return self.attendance_records[student_id]
        else:
            print(f"No attendance records found for student {student_id}.
            return None

class EventRegistration:
    def __init__(self):
        self.events = {}  # Using a dictionary (Hash Table)
```

```
    def add_event(self, event_id, ever
        if event_id not in self.events:
            self.events[event_id] = {"details": event_details, "registere
            print(f"Event '{event_id}' added.")
        else:
            print(f"Event '{event_id}' already exists.")

    def register_student(self, event_id, student_id, registration_details
        if event_id in self.events:
            if student_id not in self.events[event_id]["registered_studen
                self.events[event_id]["registered_students"][student_id]
                print(f"Student {student_id} registered for event '{event
            else:
                print(f"Student {student_id} already registered for event
        else:
            print(f"Event '{event_id}' not found.")

    def get_registered_students(self, event_id):
        if event_id in self.events:
            print(f"Registered students for event '{event_id}':")
            if self.events[event_id]["registered_students"]:
                for student_id, details in self.events[event_id]["registe
                    print(f"  Student ID: {student_id}, Details: {details
                return self.events[event_id]["registered_students"]
            else:
                print("  No students registered yet.")
```

```
    def add_event(self, event_id, ever
        if event_id not in self.events:
            self.events[event_id] = {"details": event_details, "registere
            print(f"Event '{event_id}' added.")
        else:
            print(f"Event '{event_id}' already exists.")

    def register_student(self, event_id, student_id, registration_details
        if event_id in self.events:
            if student_id not in self.events[event_id]["registered_studen
                self.events[event_id]["registered_students"][student_id]
                print(f"Student {student_id} registered for event '{event
            else:
                print(f"Student {student_id} already registered for event
        else:
            print(f"Event '{event_id}' not found.")

    def get_registered_students(self, event_id):
        if event_id in self.events:
            print(f"Registered students for event '{event_id}':")
            if self.events[event_id]["registered_students"]:
                for student_id, details in self.events[event_id]["registe
                    print(f"  Student ID: {student_id}, Details: {details
                return self.events[event_id]["registered_students"]
            else:
                print("  No students registered yet.")
```

```
                print("  No students r
                return {}
        else:
            print(f"Event '{event_id}' not found.")
            return None

class LibraryBorrowing:
    def __init__(self):
        self.books = {}  # Using a dictionary (Hash Table)

    def add_book(self, isbn, book_details):
        if isbn not in self.books:
            self.books[isbn] = {"details": book_details, "available": Tru
            print(f"Book '{book_details.get('title', isbn)}' (ISBN: {isbn
        else:
            print(f"Book with ISBN '{isbn}' already exists.")

    def borrow_book(self, isbn, student_id):
        if isbn in self.books:
            if self.books[isbn]["available"]:
                self.books[isbn]["available"] = False
                self.books[isbn]["borrowed_by"] = student_id
                print(f"Book '{self.books[isbn]['details'].get('title', i
            else:
                print(f"Book '{self.books[isbn]['details'].get('title', i
        else:
```

```
        else:
            print(f"Book with ISBN '{isbn
                                    not found.")

    def return_book(self, isbn, student_id):
        if isbn in self.books:
            if not self.books[isbn]["available"] and self.books[isbn]["borro
                self.books[isbn]["available"] = True
                self.books[isbn]["borrowed_by"] = None
                print(f"Book '{self.books[isbn]['details'].get('title', isbn
            elif self.books[isbn]["available"]:
                print(f"Book '{self.books[isbn]['details'].get('title', isbn
            else:
                print(f"Book '{self.books[isbn]['details'].get('title', isbn
        else:
            print(f"Book with ISBN '{isbn}' not found.")


    def is_book_available(self, isbn):
        if isbn in self.books:
            return self.books[isbn]["available"]
        return False

ss BusScheduling:
    def __init__(self):
        self.routes = {}  # Using a dictionary to represent a Graph (Adjacer
```

```
            print(f"Bus stop '{stop_name}
        else:
            print(f"Bus stop '{stop_name}' already exists.")

    def add_route(self, stop1, stop2):
        if stop1 in self.routes and stop2 in self.routes:
            if stop2 not in self.routes[stop1]:
                self.routes[stop1].append(stop2)
                self.routes[stop2].append(stop1) # For an undirected route
                print(f"Route added between '{stop1}' and '{stop2}'.")
            else:
                print(f"Route already exists between '{stop1}' and '{stop2}'
        else:
            print("One or both bus stops not found.")

    def get_neighbors(self, stop_name):
        if stop_name in self.routes:
            print(f"Stops reachable from '{stop_name}': {self.routes[stop_na
            return self.routes[stop_name]
        else:
            print(f"Bus stop '{stop_name}' not found.")
            return None

ss CafeteriaOrderQueue:
    def __init__(self):
        self.order_queue = deque()  # Using a deque
```

```
    def get_student_attendance(self, stu
        if student_id in self.student_re
            self.attendance_system.get_attendance(student_id)
        else:
            print(f"Student {student_id} not found in registry. Cannot get


    def add_event(self, event_id, event_details):
        self.event_system.add_event(event_id, event_details)

    def register_student_for_event(self, event_id, student_id, registration_
        if student_id in self.student_registry:
            self.event_system.register_student(event_id, student_id, registr
            if event_id in self.event_system.events and student_id in self.e
                self.student_registry[student_id]["registered_events"].appe
        else:
            print(f"Student {student_id} not found in registry. Cannot regis

    def get_event_registered_students(self, event_id):
        self.event_system.get_registered_students(event_id)


    def add_library_book(self, isbn, book_details):
        self.library_system.add_book(isbn, book_details)

    def student_borrows_book(self, isbn, student_id):
        if student_id in self.student_registry:
```

```
        if student_id in self.student_reg
            if self.library_system.is_boo
                self.library_system.borrow_book(isbn, student_id)
                self.student_registry[student_id]["borrowed_books"].append(i
            else:
                print(f"Book with ISBN {isbn} is not available.")
        else:
            print(f"Student {student_id} not found in registry. Cannot borro

    def student_returns_book(self, isbn, student_id):
        if student_id in self.student_registry:
            if isbn in self.student_registry[student_id]["borrowed_books"]:
                self.library_system.return_book(isbn, student_id)
                self.student_registry[student_id]["borrowed_books"].remove(
            else:
                print(f"Student {student_id} did not borrow book with ISBN
        else:
            print(f"Student {student_id} not found in registry. Cannot retu

    def check_book_availability(self, isbn):
        if self.library_system.is_book_available(isbn):
            print(f"Book with ISBN {isbn} is available.")
        else:
            print(f"Book with ISBN {isbn} is not available.")
```

```
    def add_bus_stop(self, stop_name):
        self.bus_system.add_stop(stop_name)

    def add_bus_route(self, stop1, stop2):
        self.bus_system.add_route(stop1, stop2)

    def get_bus_neighbors(self, stop_name):
        self.bus_system.get_neighbors(stop_name)


    def place_cafeteria_order(self, order_details):
        self.cafeteria_queue.enqueue_order(order_details)

    def process_next_cafeteria_order(self):
        self.cafeteria_queue.dequeue_order()

    def peek_cafeteria_order(self):
        print(f"Next order in queue: {self.cafeteria_queue.peek_order()}")

    def is_cafeteria_queue_empty(self):
        if self.cafeteria_queue.is_empty():
            print("Cafeteria order queue is empty.")
        else:
            print("Cafeteria order queue is not empty.")
```

```
ain interactive part
pus_system = CampusSystem()

le True:
 print("\nCampus Resource Management System:")
 print("1. Student Registry")
 print("2. Student Attendance")
 print("3. Event Registration")
 print("4. Library Borrowing")
 print("5. Bus Scheduling")
 print("6. Cafeteria Order Queue")
 print("7. Exit")

 main_choice = input("Enter your choice: ")

 if main_choice == '1':
     while True:
         print("\nStudent Registry Operations:")
         print("1. Add Student")
         print("2. Get Student Info")
         print("3. Back to Main Menu")
         reg_choice = input("Enter your choice: ")
         if reg_choice == '1':
             student_id = input("Enter student ID: ")
             name = input("Enter student name: ")
             contact = input("Enter student contact: ")
```

```
Campus Resource Management System:
1. Student Registry
2. Student Attendance
3. Event Registration
4. Library Borrowing
5. Bus Scheduling
6. Cafeteria Order Queue
7. Exit
Enter your choice: 1

Student Registry Operations:
1. Add Student
2. Get Student Info
3. Back to Main Menu
Enter your choice: 1
Enter student ID: 2
Enter student name: priyanka
Enter student contact: 465768
Student 2 added to registry.

Student Registry Operations:
1. Add Student
2. Get Student Info
3. Back to Main Menu
Enter your choice: 2
Enter student ID: 2
Student Info: {'name': 'priyanka', 'contact': '465768', 'borrowed_books':
```

```
class LibraryBorrowing:
    def __init__(self):
        self.books = {}  # Using a dictionary (Hash Table)

    def add_book(self, isbn, book_details):
        pass # Placeholder

    def borrow_book(self, isbn, student_id):
        pass # Placeholder

    def return_book(self, isbn, student_id):
        pass # Placeholder

    def is_book_available(self, isbn):
        pass # Placeholder

class BusScheduling:
    def __init__(self):
        self.routes = {}  # Using a dictionary to represent a Graph (Adj
    def add_stop(self, stop_name):
        pass # Placeholder

    def add_route(self, stop1, stop2):
        pass # Placeholder
```

OBSERVATION:

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.