

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		
1	Lab 11 – Data Structures with AI: Implementing Fundamental Structures Lab Objectives <ul style="list-style-type: none"> • 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. 	Expected Time to complete Week6 - Monday	

	<ul style="list-style-type: none"> Enhance code quality with AI-generated comments and performance suggestions. 	
	<p>Task Description #1 – Stack Implementation</p> <p>Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.</p> <p>Sample Input Code:</p> <pre>class Stack: pass</pre> <p>Expected Output:</p> <ul style="list-style-type: none"> A functional stack implementation with all required methods and docstrings. <p>PROMPT:</p> <p>"Generate a Stack class in Python with the following methods: push, pop, peek, and is_empty. Include comments explaining each method."</p> <p>SCREENSHOT:</p> <p>The screenshot shows a code editor with two tabs: 'ass11.1.py' and 'ass11.1-2.py'. The code in 'ass11.1-2.py' is as follows:</p> <pre>62 63 >push(10) 64 >push(20) 65 >push(30) 66 67 'Top of stack:', stack.peek() # 30 68 'Is stack empty?', stack.is_empty() # False 69 70 'Popped item:', stack.pop() # 30 71 'Top of stack after pop:', stack.peek() # 20 72 73 >pop() 74 >pop() 75 76 'Is stack empty after popping all items?', stack.is_empty() # True</pre> <p>Below the code editor is a terminal window showing the execution of the code:</p> <pre>PS C:\Users\VYSHNAVI\OneDrive\Desktop\k-midlink> & C:/Python313/python.exe "c:/Users/VYSHNAVI/OneDrive/AI ASSISTED CODING/ass11.1-2.py" Queue size now: 2 ● PS C:\Users\VYSHNAVI\OneDrive\Desktop\k-midlink> & C:/Python313/python.exe "c:/Users/VYSHNAVI/OneDrive/AI ASSISTED CODING/ass11.1.py" Is stack empty? True Top of stack: 30 Is stack empty? False Popped item: 30 Top of stack after pop: 20 Is stack empty after popping all items? True ○ PS C:\Users\VYSHNAVI\OneDrive\Desktop\k-midlink></pre>	
	<p>Task Description #2 – Queue Implementation</p> <p>Task: Use AI to implement a Queue using Python lists.</p> <p>Sample Input Code:</p> <pre>class Queue: pass</pre> <p>Expected Output:</p>	

- FIFO-based queue class with enqueue, dequeue, peek, and size methods.

PROMPT:

"Write a Python class called Queue that uses a list to implement."

SCREENSHOT:

ass11.1.py ass11.1-2.py

C: > Users > VYSHNAVI > OneDrive > AI ASSISSTED CODING > ass11.1-2.py > [e] queue

```
1 class Queue:
2     """
3         A simple FIFO (First-In-First-Out) queue implementation using a Python list.
4         Supports enqueue, dequeue, peek, and size operations.
5     """
6
7     def __init__(self):
8         """Initialize an empty queue."""
9         self._items = []
10
11    def enqueue(self, item):
12        """
13            Add an item to the end of the queue.
14
15            Parameters:
16            item (any): The item to be added to the queue.
17        """
18        self._items.append(item)
19
20    def dequeue(self):
21
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⌂ ⌂ ... ⌂ ⌂ ⌂
```

PS C:\Users\VYSHNAVI\OneDrive\Desktop\k-midlink> & C:/Python313/python.exe "c:/Users/VYSHNAVI/OneDrive/AI ASSISSTED CODING/ass11.1.py"

Is stack empty after popping all items? True

PS C:\Users\VYSHNAVI\OneDrive\Desktop\k-midlink> & C:/Python313/python.exe "c:/Users/VYSHNAVI/OneDrive/AI ASSISSTED CODING/ass11.1-2.py"

Queue size: 0

Queue size after enqueue: 3

Front item: apple

Dequeued: apple

Front item after dequeue: banana

Queue size now: 2

PS C:\Users\VYSHNAVI\OneDrive\Desktop\k-midlink>

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:

"Write a Python class for a Singly Linked List with two methods: insert to add a node at the end, and display to print all node values. Include comments explaining each part of the code."

SCREENSHOT:

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

PROMPT:
"Write a Python class for a Binary Search Tree with two methods: insert to add nodes, and `in_order_traversal` to print the tree in sorted order. Include comments explaining each method."

Comments explain SCREENSHOT:

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:

"Write a Python class for a hash table using lists. Include methods for insert, search, and delete. Add comments explaining each method."

SCREENSHOT:

The screenshot shows a code editor interface with a terminal window below it. The code editor displays a Python script named `ass11.1-5.py`. The script defines a `HashTable` class with methods for insertion, search, and deletion. The terminal window shows the command `C:/Python313/python.exe "c:/Users/VYSHNAVI/OneDrive/AI ASSISTED CODING/ass11.1-5.py"` being run, followed by the output of the program's execution.

```
1  class HashTable:
2      def __init__(self):
3          self.bucket = [None] * 10
4
5      def insert(self, key, value):
6          index = hash(key) % len(self.bucket)
7          if self.bucket[index] is None:
8              self.bucket[index] = [[key, value]]
9          else:
10             for i in range(len(self.bucket[index])):
11                 if self.bucket[index][i][0] == key:
12                     self.bucket[index][i][1] = value
13                     return
14             self.bucket[index].append([key, value])
15
16      def search(self, key):
17          index = hash(key) % len(self.bucket)
18          if self.bucket[index] is None:
19              return None
20          for i in range(len(self.bucket[index])):
21              if self.bucket[index][i][0] == key:
22                  return self.bucket[index][i][1]
23          return None
24
25      def delete(self, key):
26          index = hash(key) % len(self.bucket)
27          if self.bucket[index] is None:
28              return False
29          for i in range(len(self.bucket[index])):
30              if self.bucket[index][i][0] == key:
31                  del self.bucket[index][i]
32                  return True
33          return False
34
35 ht = HashTable()
36 ht.insert("apple", 100)
37 ht.insert("banana", 200)
38 ht.insert("orange", 300)
39
40 print(ht.search("banana")) # 200
41 print(ht.search("grape")) # None
42
43 ht.delete("banana")
44 print(ht.search("banana")) # None
```

```
PS C:\Users\VYSHNAVI\OneDrive\Desktop\k-midlink> & C:/Python313/python.exe "c:/Users/VYSHNAVI/OneDrive/AI ASSISTED CODING/ass11.1-5.py"
Search 'banana': 200
Search 'grape': None
Search 'banana' after deletion: None
PS C:\Users\VYSHNAVI\OneDrive\Desktop\k-midlink>
```

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 class to represent a graph using an adjacency list. Include methods to add vertices, add edges, and display the graph. Add comments explaining each method."

SCREENSHOT:

The screenshot shows a VS Code interface with the following details:

- Code Editor:** The file `ass11.1-6.py` is open, displaying Python code for a graph class. The code includes methods for initializing the graph, adding vertices, and adding edges.
- Terminal:** The terminal shows the command to run the script: `python ass11.1-6.py`. The output indicates the graph connections: A → B, C → A, and C → B.
- Status Bar:** Shows the current line (Ln 1, Col 1), spaces used (Spaces: 4), encoding (UTF-8), and other status information like Python version (3.13.7) and Go Live.

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:

"Write a Python class for a priority queue using the heapq module.

Include methods to insert items with priority, remove the highest priority item, and peek at the highest priority item. Add comments explaining each method."

SCREENSHOT:

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 class that wraps collections.deque to implement a double-ended queue. Include methods to add and remove elements from both ends, and display the current state. Add comments explaining each method."

SCREENSHOT:

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.

Structure	Access	Search	Insert	Delete	Notes
Array	$O(1)$	$O(n)$	$O(n)$	$O(n)$	Fast access, fixed size
Dyn. Array	$O(1)*$	$O(n)$	$O(1)*$	$O(n)$	Resizable, amortized ops
Singly List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	Insert/delete at head
Doubly List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	Bidirectional links
Stack (Arr)	$O(n)$	$O(n)$	$O(1)$	$O(1)$	LIFO, fixed size
Stack (List)	$O(n)$	$O(n)$	$O(1)$	$O(1)$	LIFO, dynamic
Queue (Arr)	$O(n)$	$O(n)$	$O(1)$	$O(1)$	FIFO, may need shifting
Queue (List)	$O(n)$	$O(n)$	$O(1)$	$O(1)$	FIFO, dynamic
Circ. Queue	$O(n)$	$O(n)$	$O(1)$	$O(1)$	Wrap-around buffer
Deque	$O(n)$	$O(n)$	$O(1)$	$O(1)$	Insert/delete both

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

- | | | |
|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| | <ul style="list-style-type: none"> • Justify your choice in 2–3 sentences per feature. • Implement one selected feature as a working Python program with AI-assisted code generation. | |
|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
- 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:

Choose the best data structure for each campus system feature (attendance, events, library, buses, cafeteria) from a given list. Justify each choice in 2–3 sentences. Then, implement one feature in Python with clear comments and docstrings.

SCREENSHOT:

Feature	Chosen Data Structure	Justification
1. Student Attendance Tracking	Stack	A stack is ideal for tracking entry/exit logs in reverse chronological order. It allows quick access to the most recent activity, which is useful for audits or tracing last movements.
2. Event Registration System	Hash Table	A hash table enables constant-time search, insertion, and deletion of participants using unique IDs. This ensures fast registration lookup and efficient removal.
3. Library Book Borrowing	Priority Queue	A priority queue can manage books based on due dates, allowing the system to prioritize overdue returns or upcoming deadlines.
4. Bus Scheduling System	Graph	A graph is perfect for modeling bus routes and stop connections, where nodes represent stops and edges represent routes. It supports pathfinding and route optimization.
5. Cafeteria Order Queue	Queue	A queue ensures students are served in the order they arrive (FIFO), maintaining fairness and simplicity in processing orders.

CODE:

C:\Users\YVSHNAVI> 11.1.10.py

```
1 from collections import deque
2
3 class CafeteriaQueue:
4     """
5         A class to manage cafeteria orders using a FIFO queue.
6     """
7
8     def __init__(self):
9         """Initialize an empty queue for orders."""
10        self.order_queue = deque()
11
12    def place_order(self, student_name):
13        """
14            Add a student's order to the queue.
15
16            Args:
17                student_name (str): Name of the student placing the order.
18        """
19        self.order_queue.append(student_name)
20        print(f"Order placed by {student_name}.")
21
22    def serve_order(self):
23        """
24            Serve the next student in the queue.
25        """
26
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

; & 'c:\Python313\python.exe' "c:\Users\YVSHNAVI\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher" '52353' '--' 'C:\Users\YVSHNAVI\OneDrive\AI ASSISTED CODING\11.1.10.py'

Order placed by Aarav.
Order placed by Sneha.
Order placed by Ravi.
Current order queue: ['Aarav', 'Sneha', 'Ravi']
Order served for Aarav.
Current order queue: ['Sneha', 'Ravi']

PS C:\Users\YVSHNAVI\OneDrive\AI ASSISTED CODING>

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.