

CH .VASU
2303a51618
Batch 25

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF C	
Program Name: B. Tech		Assignment Type: Lab	
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		<div style="background-color: #f2f2f2; padding: 5px;"> Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju </div>	
Course Code	23CS002PC304	Course Title	AI Assisted
Year/Sem	III/II	Regulation	R23
Date and Day of Assignment	Week6 – Wednesday	Time(s)	23CSBTB0
Duration	2 Hours	Applicable to Batches	All batches
Assignment Number: 11.3(Present assignment number)/ 24 (Total number of assignments)			

	Question
	<p>Lab 11: Data Structures with AI Implementing Fundamental Data Structures using AI Assistance</p> <p>Lab Objectives: By the end of this lab, students will be able to:</p> <ul style="list-style-type: none"> • Design and implement fundamental data structures in Python using AI assistance. • Effectively prompt AI tools (e.g., GitHub Copilot) for code generation, optimization, and documentation. • Understand and compare core data structures: Arrays, Linked Lists, Stacks, Queues, Priority Queues, Tree • Improve code readability, efficiency, and maintainability using AI-generated suggestions. <p>Learning Outcomes After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> • Apply appropriate data structures to solve real-world problems. • Analyze time and space complexity of different data structure operations. • Use AI tools responsibly to assist (not replace) logical thinking and problem-solving. • Validate, test, and refine AI-generated code. <hr/> <p>Task 1: Smart Contact Manager (Arrays & Linked Lists)</p> <p>Scenario SR University's student club requires a simple Contact Manager Application to store members' names and photos. The application must support efficient addition, searching, and deletion of contacts.</p> <p>Tasks</p> <ol style="list-style-type: none"> 1. Implement the contact manager using arrays (lists). 2. Implement the same functionality using a linked list for dynamic memory allocation. 3. Implement the following operations in both approaches: <ul style="list-style-type: none"> ○ Add a contact ○ Search for a contact ○ Delete a contact 4. Use GitHub Copilot to assist in generating search and delete methods. 5. Compare array vs. linked list approaches with respect to: <ul style="list-style-type: none"> ○ Insertion efficiency ○ Deletion efficiency <p>Expected Outcome</p> <ul style="list-style-type: none"> • Two working implementations (array-based and linked-list-based). • A brief comparison explaining performance differences.

Prompt

Generate Python code for a contact manager using a list with add, search, and delete operations.

The screenshot shows a Jupyter Notebook interface with a dark theme. A code cell titled "Contact Manager" contains the following Python code:

```
contacts = []

def add_contact(name, phone, email):
    """
    Adds a new contact to the contacts list.
    """
    contact = {
        'name': name,
        'phone': phone,
        'email': email
    }
    contacts.append(contact)
    print(f"Contact '{name}' added successfully.")

def search_contact(name):
    """
    Searches for contacts by name and returns a list of matching contacts.
    """
    found_contacts = [c for c in contacts if name.lower() in c['name']]
    if found_contacts:
        print(f"Found {len(found_contacts)} contact(s) matching '{name}'")
        for contact in found_contacts:
            print(f"Name: {contact['name']}, Phone: {contact['phone']}, Email: {contact['email']}")
    else:
        print("No contacts found matching '{name}'")
        return found_contacts
```

A Gemini 2.5 Flash sidebar is open, showing a progress bar and the message "Working...". Below it, a message box asks "What can I help you build?" and displays the generated code. The sidebar also includes a "Gemini 2.5 Flash" dropdown and a "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)" link.

Generate a linked list implementation in Python with add, search, and delete contact methods.

The screenshot shows a Jupyter Notebook interface with a dark theme. In the code cell, the following Python code is run:

```

delete_contact("Non Existnt Contact")
list_all_contacts()

# Search again after deletion
search_contact("Bob")

... Contact 'Alice Smith' added successfully.
Contact 'Bob Johnson' added successfully.
Contact 'Charlie Brown' added successfully.
Contact 'Alice Wonderland' added successfully.
--- All Contacts ---
Name: Alice Smith, Phone: 111-222-3333, Email: alice@example.com
Name: Bob Johnson, Phone: 444-555-6666, Email: bob@example.com
Name: Charlie Brown, Phone: 777-888-9999, Email: charlie@example.com
Name: Alice Wonderland, Phone: 000-111-2222, Email: alice.w@example.com
...
Found 2 contact(s) matching 'Alice':
  Name: Alice Smith, Phone: 111-222-3333, Email: alice@example.com
  Name: Alice Wonderland, Phone: 000-111-2222, Email: alice.w@example.com
No contacts found matching 'David'.
Contact 'Bob Johnson' deleted successfully.
No contact found with the name 'Non Existnt Contact'.
--- All Contacts ---
Name: Alice Smith, Phone: 111-222-3333, Email: alice@example.com
Name: Charlie Brown, Phone: 777-888-9999, Email: charlie@example.com
Name: Alice Wonderland, Phone: 000-111-2222, Email: alice.w@example.com
...
No contacts found matching 'Bob'.
[]

```

A Gemini AI-generated error message is displayed in a tooltip:

Additionally, `global.contacts` is unnecessary when reassigned the global contacts list. I have already provided a fix for this by removing that line. I will re-apply this correction to cell `4ed7bd7e` and ensure cell `34cb050b` is also executed so that the contact manager functions correctly.

It looks like we're still running into an issue with the `delete_contact` function, specifically an `UnboundLocalError`. This occurs because when Generate a linked list implementation in Python with add, search, and delete contact methods.

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Conclusion:

Linked lists are more efficient for frequent insertions and deletions, while arrays are simpler and faster for index

Task 2: Library Book Search System (Queues & Priority Queues)

Scenario

The SRU Library manages book borrow requests. Students and faculty submit requests, but **faculty requests must** be prioritized.

Tasks

1. Implement a **Queue (FIFO)** to manage book requests.
2. Extend the system to a **Priority Queue**, prioritizing faculty requests.
3. Use **GitHub Copilot** to assist in generating:
 - o enqueue() method
 - o dequeue() method
4. Test the system with a mix of student and faculty requests.

Expected Outcome

- Working queue and priority queue implementations.
- Correct prioritization of faculty requests.

Prompt

Generate Python queue implementation with enqueue and dequeue methods.

The screenshot shows two AI Assistant windows side-by-side, both titled "AI ASST 11.3_V.ipynb".

Top Window: The title bar says "Queue Implementation". It contains a list of methods and their descriptions:

- enqueue(item) : Adds an item to the rear of the queue.
- dequeue() : Removes and returns the item from the front of the queue.
- is_empty() : Checks if the queue is empty.
- size() : Returns the number of items in the queue.
- peek() : Returns the item at the front of the queue without removing it.

The code editor shows the following Python class definition:

```
class Queue:  
    def __init__(self):  
        self.items = []  
  
    def enqueue(self, item):  
        """Adds an item to the rear of the queue."""  
        self.items.append(item)  
        print(f"Enqueued: {item}. Queue: {self.items}")  
  
    def dequeue(self):  
        """Removes and returns the item from the front of  
        if not self.is_empty():  
            dequeued_item = self.items.pop(0) # pop(0) re  
            print(f"Dequeued: {dequeued_item}. Queue: {se  
        return dequeued_item  
    else:  
        print("Queue is empty. Cannot dequeue.")  
        return None  
  
    def is_empty(self):
```

A modal dialog box is open, showing a message about the contact manager's functionality and a "Gemini 2.5 Flash" button.

Bottom Window: The title bar says "Gemini". It contains a generated Python code snippet for a queue:

```
# Create a new queue  
my_queue = Queue()  
print("Initial queue: {my_queue}")  
  
# Enqueue some items  
my_queue.enqueue(10)  
my_queue.enqueue(20)  
my_queue.enqueue(30)  
  
# Check the size and peek at the front item  
print(f"Queue size: {my_queue.size()}")  
print(f"Front item (peek): {my_queue.peek()}")  
  
# Dequeue items  
my_queue.dequeue()  
my_queue.dequeue()  
  
# Check if the queue is empty and try to dequeue again  
print(f"Is queue empty? {my_queue.is_empty()}")  
my_queue.dequeue()  
  
# Try to dequeue from an empty queue  
my_queue.dequeue()  
print(f"Final queue: {my_queue}")  
print(f"Is queue empty? {my_queue.is_empty()}")
```

The code editor shows the output of the generated code:

```
Initial queue: []  
Enqueued: 10. Queue: [10]  
Enqueued: 20. Queue: [10, 20]  
Enqueued: 30. Queue: [10, 20, 30]  
Queue size: 3  
Front item (peek): 10  
Dequeued: 10. Queue: [20, 30]
```

A modal dialog box is open, showing a message about the generated code and a "Gemini 2.5 Flash" button.

The screenshot shows a Jupyter Notebook cell with the following Python code:

```
my_queue.dequeue()
my_queue.dequeue()

# Check if the queue is empty and try to dequeue again
print("Is queue empty? " + str(my_queue.is_empty()))
my_queue.dequeue()

# Try to dequeue from an empty queue
my_queue.dequeue()
print("Final queue: " + str(my_queue))
print("Is queue empty? " + str(my_queue.is_empty()))

Initial queue: []
Enqueued: 10, Queue: [10]
Enqueued: 20, Queue: [10, 20]
Enqueued: 30, Queue: [10, 20, 30]
Queue size: 3
Front item (peak): 10
Dequeued: 10, Queue: [20, 30]
Dequeued: 20, Queue: [30]
Is queue empty? False
Dequeued: 30, Queue: []
Queue is empty. Cannot dequeue.
Final queue: []
Is queue empty? True
```

A floating window from Gemini 2.5 Flash is visible, showing the prompt "Generate a priority queue in Python where faculty requests have higher priority." and the status "Working...".

Faculty requests are processed first due to higher priority (1).
Priority Queue improves fairness and urgency handling.

Task 3: Emergency Help Desk (Stack Implementation)

Scenario

SR University's IT Help Desk receives technical support tickets from students and staff. While tickets are received, it follows a **Last-In, First-Out (LIFO)** approach.

Tasks

1. Implement a **Stack** to manage support tickets.
2. Provide the following operations:
 - o push(ticket)
 - o pop()
 - o peek()
3. Simulate at least **five tickets** being raised and resolved.
4. Use **GitHub Copilot** to suggest additional stack operations such as:
 - o Checking whether the stack is empty
 - o Checking whether the stack is full (if applicable)

Expected Outcome

- Functional stack-based ticket management system.
- Clear demonstration of LIFO behavior.

Prompt

Generate a stack class in Python with push, pop, peek, isEmpty methods.

The screenshot shows a Jupyter Notebook interface titled "AI ASST 11.3_V.ipynb". The notebook has a dark theme. On the left, there's a sidebar with icons for file operations. The main area has a code cell containing the following Python code:

```
class Stack:  
    def __init__(self):  
        self.items = []  
  
    def push(self, item):  
        """Adds an item to the top of the stack."""  
        self.items.append(item)  
        print(f"Pushed: {item}. Stack: {self.items}")  
  
    def pop(self):  
        """Removes and returns the item from the top of the stack.  
        If not self.is_empty():  
            popped_item = self.items.pop() # pop() remove  
            print(f"Popped: {popped_item}. Stack: {self.items}")  
            return popped_item  
        else:  
            print("Stack is empty. Cannot pop.")  
            return None
```

To the right of the code cell, a GitHub Copilot AI interface is open. It shows a prompt: "Generate a stack class in Python with push, pop, peek, isEmpty methods." Below the prompt, a message says: "Certainly! Here's a Python implementation of a Stack class. It uses a Python list internally and includes push to add elements, pop to remove the top element, peek to view the top element without removing it, and is_empty to check if the stack is empty. I'll also add a size method and a peek method." At the bottom of the AI interface, there are buttons for "Accept & Run", "Accept", and "Cancel".

The screenshot shows a Jupyter Notebook interface titled "AI ASST 11.3_V.ipynb". The code cell contains the following Python code:

```

my_stack.push('C')
# Check the size and peek at the top item
print("Stack size: " + str(my_stack.size()))
print("Top item (peek): " + my_stack.peek())

# Pop items
my_stack.pop()
my_stack.pop()

# Check if the stack is empty and try to pop again
print("Is stack empty? " + str(my_stack.is_empty()))
my_stack.pop()

# Try to pop from an empty stack
my_stack.pop()
print("Final stack: " + str(my_stack))
print("Is stack empty? " + str(my_stack.is_empty()))

```

The output cell shows the execution results:

```

Initial stack: []
Pushed: A. Stack: ['A']
Pushed: B. Stack: ['A', 'B']
Pushed: C. Stack: ['A', 'B', 'C']
Stack size: 3
Top item (peek): C
Popped: C. Stack: ['A', 'B']
Popped: B. Stack: ['A']
Is stack empty? False
Popped: A. Stack: []
Stack is empty. Cannot pop.
Final stack: []
Is stack empty? True

```

A sidebar on the right contains a message from Gemini 2.5 Flash:

Generate a stack class in Python with push, pop, peek, isEmpty methods.

Certainly! Here's a Python implementation of a Stack class. It uses a Python list internally and includes `push` to add elements, `pop` to remove the top element, `peek` to view the top element without removing it, and `is_empty` to check if the stack is empty. I'll also add a `size` method and example usage.

What can I help you build?

Stack follows **LIFO**. Latest ticket is resolved first.

Task 4: Hash Table

Objective

To implement a **Hash Table** and understand collision handling.

Task Description

Use AI to generate a hash table with:

- Insert
- Search
- Delete

Starter Code

class HashTable:

```
pass
```

Expected Outcome

- Collision handling using chaining
 - Well-commented methods
-

Prompt

Generate a hash table in Python using chaining with insert, search, and delete methods.

AI ASST 11.3_V.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text | Run all

```
Pushed: A. Stack: [A]
Pushed: B. Stack: ['A', 'B']
Pushed: C. Stack: ['A', 'B', 'C']
Stack size: 3
Top item (peek): C
Popped: C. Stack: ['A', 'B']
Popped: B. Stack: ['A']
Is stack empty? False
Popped: A. Stack: []
Stack is empty. Cannot pop.
Final stack: []
Is stack empty? True
```

[] Start coding or generate with AI.

Generate a hash table in Python using chaining with insert, search, and delete methods.

Working...

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Gemini 2.5 Flash

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** AI ASST 11.3_V.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Code Cell:** Displays the following Python code for a stack implementation:

```
initial_stack: []
Pushed: A. Stack: ['A']
Pushed: B. Stack: ['A', 'B']
Pushed: C. Stack: ['A', 'B', 'C']
Stack size: 3
Top item (peek): C
Popped: C. Stack: ['A', 'B']
Popped: B. Stack: ['A']
Is stack empty? False
Popped: A. Stack: []
Stack is empty. Cannot pop.
Final stack: []
Is stack empty? True
```
- Text Cell:** Contains the heading "Hash Table Implementation (with Chaining)".

This section implements a basic Hash Table data structure using Python's built-in dictionary. A hash table is a list that can hold multiple key-value pairs.

It provides the following methods:

 - `__init__(capacity)`: Initializes the hash table with a specified capacity.
 - `_hash_function(key)`: A private method to compute the index for a given key.
 - `insert(key, value)`: Adds a key-value pair to the hash table.
 - `search(key)`: Searches for a key and returns its associated value.
 - `delete(key)`: Removes a key-value pair from the hash table.
- AI Assistant Panel:** A floating panel on the right side with the following content:
 - A profile icon and the text: "Generate a hash table in Python using chaining with insert, delete methods."
 - A blue diamond icon and the text: "Certainly! Here's a Python implementation of a hash table using chaining. It includes methods for inserting key-value pairs, searching by key, and deleting key-value pairs, handling collisions via chaining."

Buttons at the bottom of the panel: Accept & Run, Accept, Cancel.
- Bottom Status Bar:** Gemini 2.0.0
- Bottom Footer:** Gemini can make mistakes so double-check it and use code with caution. Learn more.

```

ATLASST 11.3_v.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▶ Run all ▾

[15]
✓ Os
print("\n--- Final Hash Table State ---")
print(my_hash_table)

print("\n--- Searching after deletion ---")
print(f"Value for 'cherry': {my_hash_table.search('cherry')}")

```

Initialized Hash Table with capacity: 5
Hash Table is empty.

--- Inserting items ---
Inserted ('apple', '10') at index 4.
Inserted ('banana', '20') at index 1.
Inserted ('cherry', '30') at index 0.
Inserted ('date', '40') at index 0.
Inserted ('elderberry', '50') at index 2.
Inserted ('grape', '60') at index 2.
Updated key 'apple' at index 4 with new value '100'.

--- Current Hash Table State ---
Hash Table
[0]: [('cherry', 30), ('date', 40)]
[1]: [('banana', 20)]
[2]: [('elderberry', 50), ('grape', 60)]
[4]: [('apple', 100)]

--- Searching for items ---
Found key 'banana' at index 1 with value '20'.
Value for 'banana': 20
Found key 'apple' at index 4 with value '100'.
Value for 'apple': 100
Found key 'grape' at index 2 with value '60'.
Value for 'grape': 60
Key 'fig' not found.
Value for 'fig': None

--- Deleting items ---

Variables Terminal

Generate a hash table in Python using chaining with insert, search, delete methods.

Certainly! Here's a Python implementation of a hash table using chaining. It includes methods for inserting key-value pairs, searching for values, and deleting items.

[Implement a linked list data structure](#) [Explain the Hash Table chaining implementation](#)

What can I help you build?

Gemini 2.5 Flan

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Task 5: Real-Time Application Challenge

Scenario

Design a **Campus Resource Management System** with the following features:

- Student Attendance Tracking
- Event Registration System
- Library Book Borrowing
- Bus Scheduling System
- Cafeteria Order Queue

Student Tasks

1. Choose the most appropriate data structure for each feature.
2. Justify your choice in **2–3 sentences**.
3. Implement **one selected feature** using AI-assisted code generation.

Expected Outcome

- Mapping table: Feature → Data Structure → Justification

- One fully working Python implementation

Note: Report

should be submitted as a word document for all tasks in a single document with prompts, comments & code explanations & screenshots.

Task 5: Real-Time Application Challenge

◊ Mapping Table

Feature	Data Structure	Justification
Attendance	Hash Table	Fast lookup
Event Registration	Queue	FIFO
Library Borrowing	Priority Queue	Faculty priority
Bus Scheduling	Queue	Sequential order
Cafeteria Orders	Queue	Order of arrival

Conclusion

AI tools like GitHub Copilot helped generate boilerplate code faster, but logical understanding and testing were essential to verify correctness and efficiency.