Assignment-11.3

2303A51304

Batch-05

Task 1: Smart Contact Manager (Arrays & Linked Lists)

Prompt:

Create a Contact Manager in Python with TWO implementations:

Array (list) based

Singly linked list based

Each contact has a name and phone number.

For BOTH implementations, implement:

add_contact(name, phone)

search_contact(name)

delete_contact(name)

use choice to select which implementation to use.

Code:

```
class Contact:
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone
class ArrayContactManager:
    def __init__(self):
        self.contacts = []
    def add_contact(self, name, phone):
        contact = Contact(name, phone)
        self.contacts.append(contact)
    def search_contact(self, name):
        for contact in self.contacts:
            if contact.name == name:
                return contact.phone
        return None
    def delete_contact(self, name):
        for i, contact in enumerate(self.contacts):
            if contact.name == name:
                del self.contacts[i]
                return True
        return False
class Node:
    def __init__(self, contact):
        self.contact = contact
        self.next = None
class LinkedListContactManager:
    def __init__(self):
        self.head = None
    def add_contact(self, name, phone):
        contact = Contact(name, phone)
        new_node = Node(contact)
        new_node.next = self.head
        self.head = new_node
    def search_contact(self, name):
        current = self.head
        while current:
            if current.contact.name == name:
```

assignment-11.3.py > ...

```
35  class LinkedListContactManager:
38      def add_contact(self, name, phone):
42          self.head = new_node
43      def search_contact(self, name):
44          current = self.head
45          while current:
46              if current.contact.name == name:
47                  return current.contact.phone
48              current = current.next
49          return None
50      def delete_contact(self, name):
51          current = self.head
52          prev = None
53          while current:
54              if current.contact.name == name:
55                  if prev:
56                      prev.next = current.next
57                  else:
58                      self.head = current.next
59                  return True
60              prev = current
61              current = current.next
62          return False
63  def main():
64      print("Choose implementation:")
65      print("1. Array (list) based")
66      print("2. Singly linked list based")
67      choice = int(input("Enter choice (1 or 2): "))
68      if choice == 1:
69          manager = ArrayContactManager()
70      elif choice == 2:
71          manager = LinkedListContactManager()
72      else:
73          print("Invalid choice!")
74          return
75      while True:
76          print("\nMenu:")
```

```
def main():
        manager = LinkedListContactManager()
    else:
        print("Invalid choice!")
        return
    while True:
        print("\nMenu:")
        print("1. Add Contact")
        print("2. Search Contact")
        print("3. Delete Contact")
        print("4. Exit")
        option = int(input("Enter option: "))
        if option == 1:
            name = input("Enter name: ")
            phone = input("Enter phone number: ")
            manager.add_contact(name, phone)
            print("Contact added.")
        elif option == 2:
            name = input("Enter name to search: ")
            phone = manager.search_contact(name)
            if phone:
                print(f"Phone number for {name}: {phone}")
            else:
                print(f"{name} not found.")
        elif option == 3:
            name = input("Enter name to delete: ")
            if manager.delete_contact(name):
                print(f"{name} deleted.")
            else:
                print(f"{name} not found.")
        elif option == 4:
            break
        else:
            print("Invalid option!")
if __name__ == "__main__":
    main()
```

Output:

```
signment-11.3.py
Choose implementation:
1. Array (list) based
2. Singly linked list based
Enter choice (1 or 2): 1

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 1
Enter name: abc
Enter phone number: 123456
Contact added.

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 1
Enter name: bcd
Enter phone number: 890754
Contact added.

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 1
Enter name: ghtf
Enter phone number: 87645
Contact added.

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 2
Enter name to search: abc
Phone number for abc: 123456
```

```
Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 3
Enter name to delete: ghtf
ghtf deleted.

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 4
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3
signment-11.3.py
Choose implementation:
1. Array (list) based
2. Singly linked list based
Enter choice (1 or 2): 2

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 1
Enter name: lmn
Enter phone number: 789
Contact added.

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 1
Enter name: opi
Enter phone number: 4560
Contact added.

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 1
```

```
3. Delete Contact
4. Exit
Enter option: 1
Enter name: opi
Enter phone number: 4560
Contact added.

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 1
Enter name: rty
Enter phone number: 1230
Contact added.

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 2
Enter name to search: lmn
Phone number for lmn: 789

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 3
Enter name to delete: opi
opi deleted.

Menu:
1. Add Contact
2. Search Contact
3. Delete Contact
4. Exit
Enter option: 4
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

The array-based contact manager stores contacts in a list, allowing simple insertion but requiring element shifting during deletion. Searching in the array implementation takes linear time as each contact is checked sequentially.
The linked list implementation stores contacts using dynamically allocated nodes, allowing efficient insertion at the beginning. Deletion in a linked list is more efficient than arrays since no shifting of elements is required.
Both implementations have linear time complexity for searching, but linked lists handle frequent updates better.

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

Prompt:

Create a Library Book Search System in Python.

Implement:

A normal Queue (FIFO) to manage book borrow requests.

A Priority Queue where faculty requests are served before student requests.

Each request should store:

requester_name, role (faculty or student), book_name

Implement and clearly define:

enqueue(request),dequeue(),FIFO behavior in the normal queue.Faculty requests have higher priority in the priority queue.,

Dequeue removes and returns the correct request.

Code is clean, correct, and well-commented for a Data Structures lab.

Finally, test the system with a mix of student and faculty requests and display the order of processing.

use choice for entering the names and also for selecting the implementation.

Code:

```
class Request:
    def __init__(self, requester_name, role, book_name):
        self.requester_name = requester_name
        self.role = role
        self.book_name = book_name
class Queue:
    def __init__(self):
        self.queue = []
    def enqueue(self, request):
        self.queue.append(request)
    def dequeue(self):
        if not self.is_empty():
            return self.queue.pop(0)
        return None
    def is_empty(self):
        return len(self.queue) == 0
class PriorityQueue:
    def __init__(self):
        self.faculty_queue = []
        self.student_queue = []
    def enqueue(self, request):
        if request.role == 'faculty':
            self.faculty_queue.append(request)
        else:
            self.student_queue.append(request)
    def dequeue(self):
        if not self.is_empty():
            if self.faculty_queue:
                return self.faculty_queue.pop(0)
            else:
                return self.student_queue.pop(0)
        return None
    def is_empty(self):
        return len(self.faculty_queue) == 0 and len(self.student_queue) == 0
# Example usage
def main():
    print("Select implementation: 1 for Normal Queue, 2 for Priority Queue")
```

```
        return None
    def is_empty(self):
        return len(self.faculty_queue) == 0 and len(self.student_queue) == 0
# Example usage
def main():
    print("Select implementation: 1 for Normal Queue, 2 for Priority Queue")
    choice = int(input())
    if choice == 1:
        queue = Queue()
    else:
        queue = PriorityQueue()
    while True:
        print("Enter requester name (or 'exit' to stop):")
        name = input()
        if name.lower() == 'exit':
            break
        print("Enter role (faculty/student):")
        role = input()
        print("Enter book name:")
        book_name = input()
        request = Request(name, role, book_name)
        queue.enqueue(request)
    print("\nProcessing requests:")
    while not queue.is_empty():
        request = queue.dequeue()
        print(f"Processing {request.role} request from {request.requester_name} for {request.book_name}")
if __name__ == "__main__":
    main()
```

Output:

```
1/OneDrive/Desktop/AI_Assistance_coding/11.3-task-02.py
Select implementation: 1 for Normal Queue, 2 for Priority Queue
2
Enter requester name (or 'exit' to stop):
abc
Enter role (faculty/student):
student
Enter book name:
hjk
Enter requester name (or 'exit' to stop):
def
Enter role (faculty/student):
faculty
Enter book name:
uij
Enter requester name (or 'exit' to stop):
ghi
Enter role (faculty/student):
hjkl
Enter book name:
jik
Enter requester name (or 'exit' to stop):
tyik
Enter role (faculty/student):
faculty
Enter book name:
hujk
Enter requester name (or 'exit' to stop):
exit

Processing requests:
Processing faculty request from def for uij
Processing faculty request from tyik for hujk
Processing student request from abc for hjk
Processing hjkl request from ghi for jik
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

This program implements a library book request system using a normal queue and a priority queue.
The normal queue follows FIFO order and processes requests in the order they arrive.
The priority queue separates faculty and student requests and always serves faculty first.
The enqueue method adds requests, while the dequeue method removes the correct request based on priority.
This approach demonstrates how priority queues handle real-world situations where some requests are more important.

Task 3: Emergency Help Desk (Stack Implementation)

Prompt:

Create an Emergency IT Help Desk system in Python using a Stack (LIFO).

Each support ticket should store:

ticket_id, requester_name, issue_description

Implement the following stack operations:

push(ticket)

pop()

peek()

Also include: is_empty(),is_full() (if a fixed stack size is used)

Simulate at least five tickets being raised and resolved to clearly demonstrate LIFO behavior.

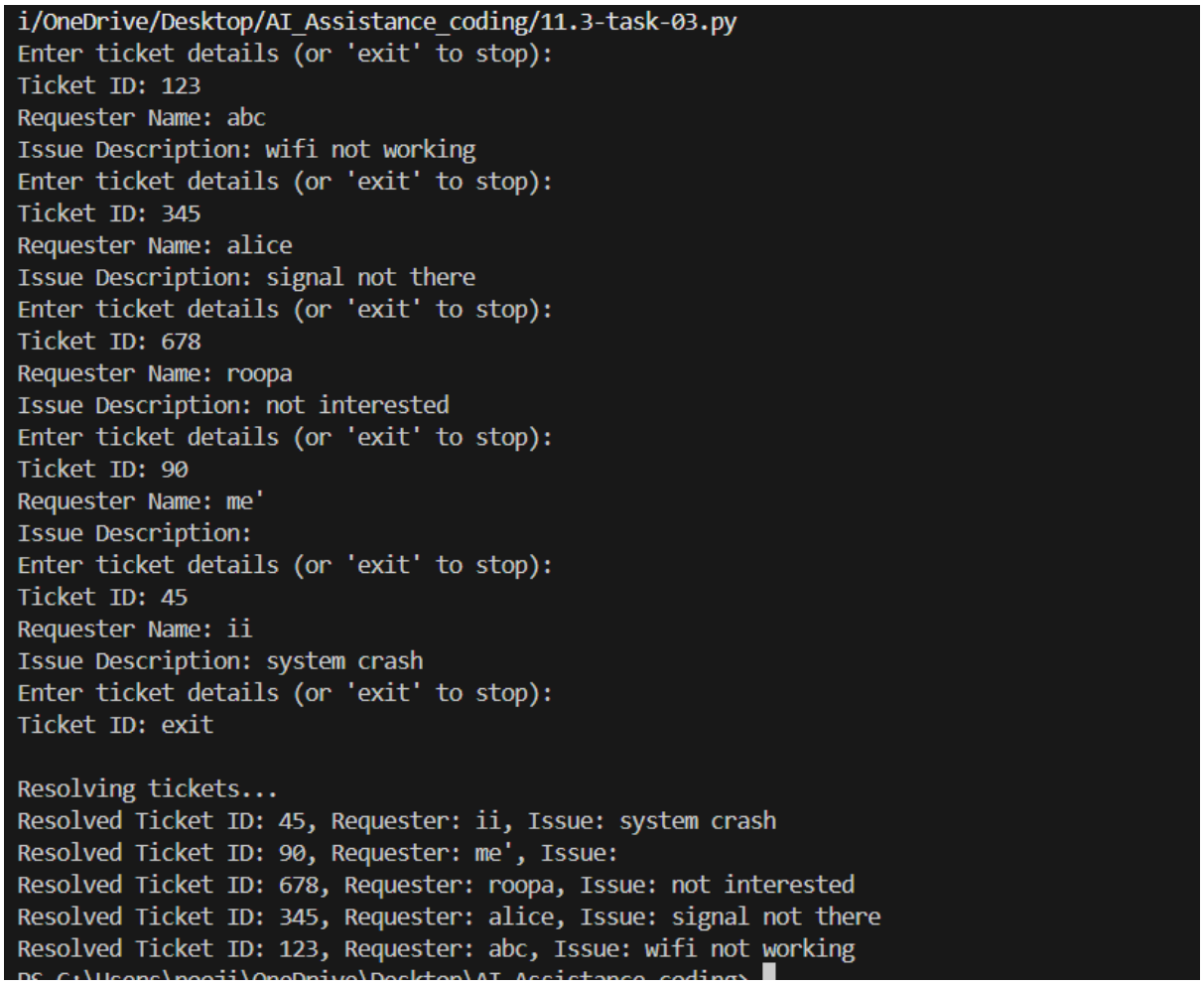
Ensure the code is clean, correct, and well-commented for a Data Structures lab.

use choice for entering the ticket details and for selecting the implementation.

Code:

```
use choice for entering the ticket details and for selecting the implementation.'''
class Ticket:
    def __init__(self, ticket_id, requester_name, issue_description):
        self.ticket_id = ticket_id
        self.requester_name = requester_name
        self.issue_description = issue_description
class Stack:
    def __init__(self, max_size=10):
        self.stack = []
        self.max_size = max_size
    def push(self, ticket):
        if not self.is_full():
            self.stack.append(ticket)
        else:
            print("Stack is full. Cannot add more tickets.")
    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        else:
            print("Stack is empty. No tickets to resolve.")
            return None
    def peek(self):
        if not self.is_empty():
            return self.stack[-1]
        else:
            print("Stack is empty. No tickets to peek.")
            return None
    def is_empty(self):
        return len(self.stack) == 0
    def is_full(self):
        return len(self.stack) >= self.max_size
# Example usage
def main():
    stack = Stack()
    while True:
        print("Enter ticket details (or 'exit' to stop):")
        ticket_id = input("Ticket ID: ")
        if ticket_id.lower() == 'exit':
            break
        requester_name = input("Requester Name: ")
        issue_description = input("Issue Description: ")
        ticket = Ticket(ticket_id, requester_name, issue_description)
        stack.push(ticket)
    print("\nResolving tickets...")
    while not stack.is_empty():
        resolved_ticket = stack.pop()
        print(f"Resolved Ticket ID: {resolved_ticket.ticket_id}, Requester: {resolved_ticket.requester_name}, Issue: {resolved_ticket.issue_description}")
if __name__ == "__main__":
    main()
```

Output:



Explanation:

This program models an IT help desk using a stack, where support tickets are handled in Last-In, First-Out order.
Each ticket stores an ID, requester name, and issue description, and is added to the stack using the push operation.
The pop operation resolves the most recently added ticket first, clearly showing LIFO behavior.
Additional methods like peek, is_empty, and is_full help manage and check the stack safely.

Task 4: Hash Table

Prompt:

Complete the following starter code by implementing a Hash Table in Python using chaining for collision handling.

```
class HashTable:
    pass
```

Requirements:

- Use a list of buckets (linked lists or Python lists) for chaining.
- Implement the following methods with clear comments:

  insert(key, value),search(key) ,delete(key)

- Handle collisions correctly using chaining.
- Ensure search returns the value if the key exists, otherwise "Key not found".
- Ensure delete safely removes the key-value pair without breaking the table.
- Keep the implementation clean, correct, and suitable for a Data Structures lab.

Use choice for entering key-value pairs and for selecting the implementation.

```
'''class HashTable:
    pass'''
```

Code:

```python
class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [None] * self.size
    def hash_function(self, key):
        return hash(key) % self.size
    def insert(self, key, value):
        index = self.hash_function(key)
        new_node = Node(key, value)
        if self.table[index] is None:
            self.table[index] = new_node
        else:
            current = self.table[index]
            while current:
                if current.key == key:
                    current.value = value  # Update existing key
                    return
                if current.next is None:
                    break
                current = current.next
            current.next = new_node
    def search(self, key):
        index = self.hash_function(key)
        current = self.table[index]
        while current:
            if current.key == key:
                return current.value
            current = current.next
        return "Key not found"
    def delete(self, key):
        index = self.hash_function(key)
        current = self.table[index]
```

1.3-task-04.py > ...

```python
class HashTable:
    def delete(self, key):
        index = self.hash_function(key)
        current = self.table[index]
        prev = None
        while current:
            if current.key == key:
                if prev:
                    prev.next = current.next
                else:
                    self.table[index] = current.next  # Deleting head of the chain
                return True
            prev = current
            current = current.next
        return False  # Key not found
# Example usage
def main():
    hash_table = HashTable()
    while True:
        print("Enter key-value pair (or 'exit' to stop):")
        key = input("Key: ")
        if key.lower() == 'exit':
            break
        value = input("Value: ")
        hash_table.insert(key, value)
    print("\nSearch for keys:")
    while True:
        search_key = input("Enter key to search (or 'exit' to stop): ")
        if search_key.lower() == 'exit':
            break
        result = hash_table.search(search_key)
        print(f"Search result: {result}")
    print("\nDelete keys:")
    while True:
        delete_key = input("Enter key to delete (or 'exit' to stop): ")
        if delete_key.lower() == 'exit':
            break
        success = hash_table.delete(delete_key)
        if success:
            print(f"Key '{delete_key}' deleted successfully.")
        else:
            print(f"Key '{delete_key}' not found.")
if __name__ == "__main__":
    main()
```

Output:

```
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pyt
i/OneDrive/Desktop/AI_Assistance_coding/11.3-task-04.py
Enter key-value pair (or 'exit' to stop):
Key: name
Value: alice
Enter key-value pair (or 'exit' to stop):
Key: age
Value: 123
Enter key-value pair (or 'exit' to stop):
Key: city
Value: hyd
Enter key-value pair (or 'exit' to stop):
Key: exit

Search for keys:
Enter key to search (or 'exit' to stop): name
Search result: alice
Enter key to search (or 'exit' to stop): city
Search result: hyd
Enter key to search (or 'exit' to stop): age
Search result: 123
Enter key to search (or 'exit' to stop): exit

Delete keys:
Enter key to delete (or 'exit' to stop): name
Key 'name' deleted successfully.
Enter key to delete (or 'exit' to stop): phone
Key 'phone' not found.
Enter key to delete (or 'exit' to stop): exit
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

This program implements a hash table using the chaining method to handle collisions.
Each table index stores a linked list of nodes, allowing multiple keys to share the same index safely.
The insert() method places a key–value pair into the appropriate bucket or updates it if the key already exists.
The search() method traverses the linked list at the computed index to find the required key.
The delete() method removes the node without breaking the chain, ensuring correct hash table behavior.

Task-05: Real-Time Application Challenge

Prompt:

'''Design a Campus Resource Management System in Python.

Create a mapping table with:

Feature → Chosen Data Structure → Justification (2-3 sentences each)

Features:

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

Choose the most appropriate data structure for each feature

(e.g., list, queue, stack, hash table, priority queue).

Select ONE feature from the above list and implement it fully in Python.

The implementation must:

- Use the chosen data structure correctly
- Include add, remove/update, and display operations
- Clearly demonstrate why the selected data structure is suitable
- Be clean, correct, and well-commented for a Data Structures lab

use choice for entering key-value pairs and for selecting the implementation.'''

Code:

```python
# ------------------ Mapping Table ------------------
def show_mapping_table():
    print("\nFeature → Data Structure → Justification\n")

    print("Student Attendance Tracking → Hash Table")
    print("Stores student IDs with attendance status for fast lookup and updates.\n")

    print("Event Registration System → List")
    print("Simple sequential storage of registered participants.\n")

    print("Library Book Borrowing → Hash Table")
    print("Allows fast search, insert, and delete of book records using book ID.\n")

    print("Bus Scheduling System → Priority Queue")
    print("Ensures buses are scheduled based on priority such as time or route.\n")

    print("Cafeteria Order Queue → Queue")
    print("Processes food orders in First-In-First-Out order.\n")


# ------------------ Selected Feature Implementation ------------------
class LibrarySystem:
    def __init__(self):
        self.books = {}  # Hash Table using dictionary

    def add_book(self, book_id, book_name):
        self.books[book_id] = book_name
        print("Book added successfully.")

    def remove_book(self, book_id):
        if book_id in self.books:
            del self.books[book_id]
            print("Book removed successfully.")
        else:
            print("Book not found.")

    def display_books(self):
```

```
    def display_books(self):
        if not self.books:
            print("No books in library.")
        else:
            print("\nLibrary Books:")
            for book_id, name in self.books.items():
                print(f"ID: {book_id}, Name: {name}")


# ------------------ Main Program ------------------
def main():
    show_mapping_table()

    print("Select feature to implement:")
    print("1. Library Book Borrowing System")
    choice = int(input("Enter choice: "))

    if choice == 1:
        library = LibrarySystem()

        while True:
            print("\n1. Add Book")
            print("2. Remove Book")
            print("3. Display Books")
            print("4. Exit")
            option = int(input("Enter option: "))

            if option == 1:
                book_id = input("Enter Book ID: ")
                book_name = input("Enter Book Name: ")
                library.add_book(book_id, book_name)

            elif option == 2:
                book_id = input("Enter Book ID to remove: ")
                library.remove_book(book_id)
```

```
            print("3. Display Books")
            print("4. Exit")
            option = int(input("Enter option: "))

            if option == 1:
                book_id = input("Enter Book ID: ")
                book_name = input("Enter Book Name: ")
                library.add_book(book_id, book_name)

            elif option == 2:
                book_id = input("Enter Book ID to remove: ")
                library.remove_book(book_id)

            elif option == 3:
                library.display_books()

            elif option == 4:
                break

            else:
                print("Invalid option.")

    else:
        print("Invalid feature choice.")


if __name__ == "__main__":
    main()
```

Output:

```
Enter key to delete (or 'exit' to stop): exit
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Loca
i/OneDrive/Desktop/AI_Assistance_coding/11.3-task-05.py

Feature → Data Structure → Justification

Student Attendance Tracking → Hash Table
Stores student IDs with attendance status for fast lookup and updates.

Event Registration System → List
Simple sequential storage of registered participants.

Library Book Borrowing → Hash Table
Allows fast search, insert, and delete of book records using book ID.

Bus Scheduling System → Priority Queue
Ensures buses are scheduled based on priority such as time or route.

Cafeteria Order Queue → Queue
Processes food orders in First-In-First-Out order.

Select feature to implement:
1. Library Book Borrowing System
Enter choice: 1

1. Add Book
2. Remove Book
3. Display Books
4. Exit
Enter option: 1
Enter Book ID: 101
Enter Book Name: python
Book added successfully.

1. Add Book
2. Remove Book
3. Display Books
4. Exit
Enter option: 1
Enter Book ID: 102
Enter Book Name: java
```

```
4. Exit
Enter option: 1
Enter Book ID: 102
Enter Book Name: java
Book added successfully.

1. Add Book
2. Remove Book
3. Display Books
4. Exit
Enter option: 1
Enter Book ID: 103
Enter Book Name: c
Book added successfully.

1. Add Book
2. Remove Book
3. Display Books
4. Exit
Enter option: 3

Library Books:
ID: 101, Name: python
ID: 102, Name: java
ID: 103, Name: c

1. Add Book
2. Remove Book
3. Display Books
4. Exit
Enter option: 2
Enter Book ID to remove: 103
Book removed successfully.

1. Add Book
2. Remove Book
3. Display Books
4. Exit
Enter option: 4
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

I selected the Library Book Borrowing System because it requires fast access, insertion, and deletion of book records. A hash table is used where the key is book_id and the value is book_name. Adding a book stores the key–value pair directly, giving O(1) average time complexity. Removing a book is efficient because the book ID allows direct lookup without traversal.