Submitted To:

Engineer Sheharyar Khan

Name: Iqra Fatima

Reg. Number: 23-CP-62

Semester: 4th

Department: CPED

Data Structures and Algorithms

(DSA)  
Lab Report

# Lab Report

# Examples

Task 1-5:

Code:

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class SinglyLinkedList:

def \_\_init\_\_(self):

self.head = None

def append(self, data):

node = Node(data)

if not self.head:

self.head = node

else:

current = self.head

while current.next:

current = current.next

current.next = node

def insert\_at\_beginning(self, data):

node = Node(data)

node.next = self.head

self.head = node

def insert\_at\_position(self, data, pos):

if pos == 0:

self.insert\_at\_beginning(data)

return

node = Node(data)

current = self.head

for \_ in range(pos - 1):

if not current.next:

break

current = current.next

node.next = current.next

current.next = node

def delete(self, data):

current = self.head

prev = None

while current:

if current.data == data:

if prev:

prev.next = current.next

else:

self.head = current.next

return

prev = current

current = current.next

def search(self, data):

current = self.head

while current:

if current.data == data:

return True

current = current.next

return False

def reverse(self):

prev = None

current = self.head

while current:

next\_node = current.next

current.next = prev

prev = current

current = next\_node

self.head = prev

def display(self):

current = self.head

while current:

print(current.data, end=" -> ")

current = current.next

print("None")

# Example Usage

sll = SinglyLinkedList()

sll.append(10)

sll.append(20)

sll.append(30)

sll.display()

sll.insert\_at\_beginning(5)

sll.insert\_at\_position(15, 2)

sll.display()

sll.delete(20)

sll.display()

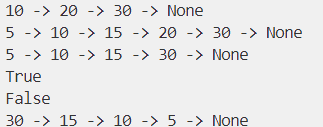
print(sll.search(15))

print(sll.search(100))

sll.reverse()

sll.display()

**Output:**

****

# Problems

## Easy Problems

1. Student Name List: Store and manage a list of student names in a linked list.

**Code:**

class Node:

    def \_\_init\_\_(self, name):

        self.name = name

        self.next = None

class StudentList:

    def \_\_init\_\_(self):

        self.head = None

    def add\_student(self, name):

        new\_node = Node(name)

        if not self.head:

            self.head = new\_node

        else:

            temp = self.head

            while temp.next:

                temp = temp.next

            temp.next = new\_node

    def remove\_student(self, name):

        if not self.head:

            print("List is empty.")

            return

        if self.head.name == name:

            self.head = self.head.next

            return

        temp = self.head

        while temp.next and temp.next.name != name:

            temp = temp.next

        if temp.next:

            temp.next = temp.next.next

        else:

            print("Student not found.")

    def display\_students(self):

        temp = self.head

        while temp:

            print(temp.name, end=" -> ")

            temp = temp.next

        print("None")

# Example Usage

students = StudentList()

students.add\_student("Ali")

students.add\_student("Ayesha")

students.add\_student("Ahmed")

students.display\_students()

students.remove\_student("Ayesha")

students.display\_students()

**Output:**

****

2. Task Scheduler: Implement a simple task manager where users can add/remove

tasks.

**Code:**

class TaskNode:

    def \_\_init\_\_(self, task):

        self.task = task

        self.next = None

class TaskScheduler:

    def \_\_init\_\_(self):

        self.head = None

    def add\_task(self, task):

        new\_node = TaskNode(task)

        if not self.head:

            self.head = new\_node

        else:

            temp = self.head

            while temp.next:

                temp = temp.next

            temp.next = new\_node

        print(f'Task "{task}" added.')

    def remove\_task(self, task):

        if not self.head:

            print("Task list is empty.")

            return

        if self.head.task == task:

            self.head = self.head.next

            print(f'Task "{task}" removed.')

            return

        temp = self.head

        while temp.next and temp.next.task != task:

            temp = temp.next

        if temp.next:

            temp.next = temp.next.next

            print(f'Task "{task}" removed.')

        else:

            print(f'Task "{task}" not found.')

    def display\_tasks(self):

        if not self.head:

            print("No tasks scheduled.")

            return

        print("Scheduled Tasks:")

        temp = self.head

        while temp:

            print(f"- {temp.task}")

            temp = temp.next

# Example Usage

tasks = TaskScheduler()

tasks.add\_task("Complete Python assignment")

tasks.add\_task("Prepare for OOP quiz")

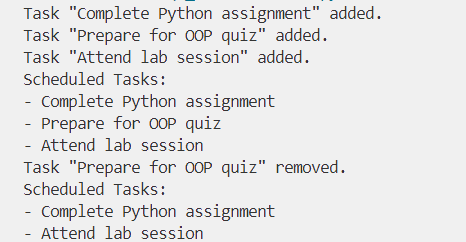
tasks.add\_task("Attend lab session")

tasks.display\_tasks()

tasks.remove\_task("Prepare for OOP quiz")

tasks.display\_tasks()

**Output:**



3. Contact List: Create a contact list using linked lists where users can search by

name.

**Code:**

class ContactNode:

    def \_\_init\_\_(self, name, phone):

        self.name = name

        self.phone = phone

        self.next = None

class ContactList:

    def \_\_init\_\_(self):

        self.head = None

    def add\_contact(self, name, phone):

        new\_node = ContactNode(name, phone)

        if not self.head:

            self.head = new\_node

        else:

            temp = self.head

            while temp.next:

                temp = temp.next

            temp.next = new\_node

        print(f'Contact "{name}" added.')

    def search\_contact(self, name):

        temp = self.head

        while temp:

            if temp.name.lower() == name.lower():

                print(f'Found: {temp.name} - {temp.phone}')

                return

            temp = temp.next

        print(f'Contact "{name}" not found.')

    def display\_contacts(self):

        if not self.head:

            print("No contacts available.")

            return

        print("Contact List:")

        temp = self.head

        while temp:

            print(f"- {temp.name}: {temp.phone}")

            temp = temp.next

# Example Usage

contacts = ContactList()

contacts.add\_contact("Ali", "03001234567")

contacts.add\_contact("Ayesha", "03219876543")

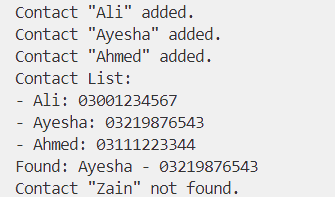
contacts.add\_contact("Ahmed", "03111223344")

contacts.display\_contacts()

contacts.search\_contact("Ayesha")

contacts.search\_contact("Zain")

**Output:**



4. Undo Feature in Editor: Implement a basic undo feature where previous actions

are stored.

**Code:**

class ActionNode:

    def \_\_init\_\_(self, action):

        self.action = action

        self.next = None

class UndoFeature:

    def \_\_init\_\_(self):

        self.top = None  # Stack کے لیے ٹاپ پوائنٹر

    def perform\_action(self, action):

        new\_node = ActionNode(action)

        new\_node.next = self.top

        self.top = new\_node

        print(f'Action performed: {action}')

    def undo(self):

        if not self.top:

            print("No actions to undo.")

            return

        print(f'Undoing: {self.top.action}')

        self.top = self.top.next  # پچھلے ایکشن پر واپس جانا

    def display\_actions(self):

        if not self.top:

            print("No actions recorded.")

            return

        print("Action History:")

        temp = self.top

        while temp:

            print(f"- {temp.action}")

            temp = temp.next

# Example Usage

editor = UndoFeature()

editor.perform\_action("Typed 'Hello'")

editor.perform\_action("Bolded text")

editor.perform\_action("Deleted a word")

editor.display\_actions()

editor.undo()

editor.display\_actions()

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

5. Simple Playlist Manager: Store a list of songs and provide a method to display them.

**Code:**

class SongNode:

    def \_\_init\_\_(self, title):

        self.title = title

        self.next = None

class Playlist:

    def \_\_init\_\_(self):

        self.head = None

    def add\_song(self, title):

        new\_node = SongNode(title)

        if not self.head:

            self.head = new\_node

        else:

            temp = self.head

            while temp.next:

                temp = temp.next

            temp.next = new\_node

        print(f'Song "{title}" added to the playlist.')

    def remove\_song(self, title):

        if not self.head:

            print("Playlist is empty.")

            return

        if self.head.title == title:

            self.head = self.head.next

            print(f'Song "{title}" removed from the playlist.')

            return

        temp = self.head

        while temp.next and temp.next.title != title:

            temp = temp.next

        if temp.next:

            temp.next = temp.next.next

            print(f'Song "{title}" removed from the playlist.')

        else:

            print(f'Song "{title}" not found in the playlist.')

    def display\_playlist(self):

        if not self.head:

            print("No songs in the playlist.")

            return

        print("Playlist:")

        temp = self.head

        while temp:

            print(f"- {temp.title}")

            temp = temp.next

# Example Usage

playlist = Playlist()

playlist.add\_song("Shape of You")

playlist.add\_song("Believer")

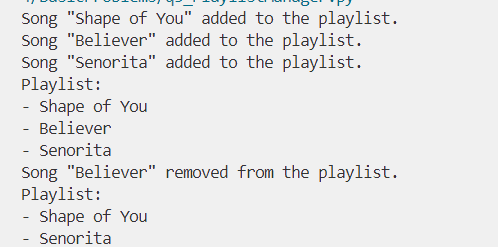
playlist.add\_song("Senorita")

playlist.display\_playlist()

playlist.remove\_song("Believer")

playlist.display\_playlist()

**Output:**

****

## Intermediate Problems

1. Version Control System: Simulate a Git commit history where commits are stored

in a linked list.

**Code:**

class CommitNode:

    def \_\_init\_\_(self, commit\_id, message):

        self.commit\_id = commit\_id

        self.message = message

        self.next = None

class VersionControlSystem:

    def \_\_init\_\_(self):

        self.head = None

        self.commit\_count = 0

    def add\_commit(self, message):

        self.commit\_count += 1

        new\_node = CommitNode(self.commit\_count, message)

        if not self.head:

            self.head = new\_node

        else:

            temp = self.head

            while temp.next:

                temp = temp.next

            temp.next = new\_node

        print(f'Commit {self.commit\_count}: "{message}" added.')

    def display\_commits(self):

        if not self.head:

            print("No commits found.")

            return

        print("Commit History:")

        temp = self.head

        while temp:

            print(f'Commit {temp.commit\_id}: {temp.message}')

            temp = temp.next

# Example Usage

vcs = VersionControlSystem()

vcs.add\_commit("Initial commit")

vcs.add\_commit("Added login feature")

vcs.add\_commit("Fixed logout bug")

vcs.display\_commits()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

2. Hospital Patient Queue: Implement a queue system where patients are treated in

order.

**Code:**

class PatientNode:

    def \_\_init\_\_(self, name, age, condition):

        self.name = name

        self.age = age

        self.condition = condition

        self.next = None

class HospitalQueue:

    def \_\_init\_\_(self):

        self.front = None  # Queue کا پہلا مریض

        self.rear = None   # Queue کا آخری مریض

    def add\_patient(self, name, age, condition):

        new\_node = PatientNode(name, age, condition)

        if not self.rear:  # اگر queue خالی ہے

            self.front = self.rear = new\_node

        else:

            self.rear.next = new\_node

            self.rear = new\_node

        print(f'Patient "{name}" added to the queue.')

    def treat\_patient(self):

        if not self.front:

            print("No patients in the queue.")

            return

        print(f'Treating patient: {self.front.name}, Condition: {self.front.condition}')

        self.front = self.front.next  # اگلا مریض queue میں آئے گا

        if not self.front:  # اگر queue خالی ہو جائے

            self.rear = None

    def display\_patients(self):

        if not self.front:

            print("No patients in the queue.")

            return

        print("Patients in Queue:")

        temp = self.front

        while temp:

            print(f'- {temp.name}, Age: {temp.age}, Condition: {temp.condition}')

            temp = temp.next

# Example Usage

hospital = HospitalQueue()

hospital.add\_patient("Ali", 30, "Fever")

hospital.add\_patient("Ayesha", 25, "Flu")

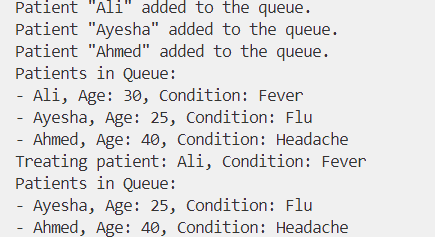
hospital.add\_patient("Ahmed", 40, "Headache")

hospital.display\_patients()

hospital.treat\_patient()

hospital.display\_patients()

**Output:**



3. Web Browser Navigation: Implement a forward/backward navigation in a web

browser.

**Code:**

class PageNode:

    def \_\_init\_\_(self, url):

        self.url = url

        self.next = None

        self.prev = None

class BrowserNavigation:

    def \_\_init\_\_(self):

        self.current\_page = None

        self.history = None

    def visit\_page(self, url):

        new\_page = PageNode(url)

        if not self.current\_page:

            self.history = self.current\_page = new\_page

        else:

            self.current\_page.next = new\_page

            new\_page.prev = self.current\_page

            self.current\_page = new\_page

        print(f'Visited: {url}')

    def go\_back(self):

        if not self.current\_page or not self.current\_page.prev:

            print("No previous page.")

            return

        self.current\_page = self.current\_page.prev

        print(f'Back to: {self.current\_page.url}')

    def go\_forward(self):

        if not self.current\_page or not self.current\_page.next:

            print("No forward page.")

            return

        self.current\_page = self.current\_page.next

        print(f'Forward to: {self.current\_page.url}')

    def display\_history(self):

        if not self.history:

            print("No browsing history.")

            return

        print("Browsing History:")

        temp = self.history

        while temp:

            print(f'- {temp.url}')

            temp = temp.next

# Example Usage

browser = BrowserNavigation()

browser.visit\_page("www.google.com")

browser.visit\_page("www.facebook.com")

browser.visit\_page("www.github.com")

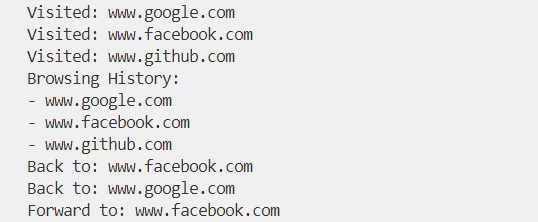
browser.display\_history()

browser.go\_back()

browser.go\_back()

browser.go\_forward()

**Output:**



4. File Management System: Simulate a hierarchical file system using linked lists.

**Code:**

class FileNode:

    def \_\_init\_\_(self, file\_name):

        self.file\_name = file\_name

        self.next = None

class Directory:

    def \_\_init\_\_(self, directory\_name):

        self.directory\_name = directory\_name

        self.files = None

    def create\_file(self, file\_name):

        new\_file = FileNode(file\_name)

        if not self.files:

            self.files = new\_file

        else:

            temp = self.files

            while temp.next:

                temp = temp.next

            temp.next = new\_file

        print(f'File "{file\_name}" created in {self.directory\_name}.')

    def delete\_file(self, file\_name):

        if not self.files:

            print("No files in this directory.")

            return

        if self.files.file\_name == file\_name:

            self.files = self.files.next

            print(f'File "{file\_name}" deleted from {self.directory\_name}.')

            return

        temp = self.files

        while temp.next and temp.next.file\_name != file\_name:

            temp = temp.next

        if temp.next:

            temp.next = temp.next.next

            print(f'File "{file\_name}" deleted from {self.directory\_name}.')

        else:

            print(f'File "{file\_name}" not found in {self.directory\_name}.')

    def display\_files(self):

        if not self.files:

            print(f"No files in {self.directory\_name}.")

            return

        print(f"Files in {self.directory\_name}:")

        temp = self.files

        while temp:

            print(f'- {temp.file\_name}')

            temp = temp.next

# Example Usage

directory = Directory("Documents")

directory.create\_file("file1.txt")

directory.create\_file("file2.txt")

directory.create\_file("file3.txt")

directory.display\_files()

directory.delete\_file("file2.txt")

directory.display\_files()

**Output:**

5. Movie Recommendation System: Store user ratings and suggest similar movies.

**Code:**

class MovieNode:

    def \_\_init\_\_(self, movie\_name, rating):

        self.movie\_name = movie\_name

        self.rating = rating

        self.next = None

class MovieRecommendationSystem:

    def \_\_init\_\_(self):

        self.head = None

    def add\_movie(self, movie\_name, rating):

        new\_movie = MovieNode(movie\_name, rating)

        if not self.head:

            self.head = new\_movie

        else:

            temp = self.head

            while temp.next:

                temp = temp.next

            temp.next = new\_movie

        print(f'Movie "{movie\_name}" with rating {rating} added.')

    def recommend\_movies(self, min\_rating):

        temp = self.head

        found = False

        print(f"Movies with rating greater than or equal to {min\_rating}:")

        while temp:

            if temp.rating >= min\_rating:

                print(f'- {temp.movie\_name} - Rating: {temp.rating}')

                found = True

            temp = temp.next

        if not found:

            print("No movies found with the specified rating.")

# Example Usage

movie\_system = MovieRecommendationSystem()

movie\_system.add\_movie("Inception", 4.8)

movie\_system.add\_movie("Titanic", 4.5)

movie\_system.add\_movie("Avatar", 4.7)

movie\_system.recommend\_movies(4.7)

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

## Advanced Problems

1. Facebook Messenger Chat History

o Implement a chat system where messages are stored in a linked list and

retrieved in order.

o Hint: Store messages as nodes with timestamps.

**Code:**

from datetime import datetime

class MessageNode:

    def \_\_init\_\_(self, message, timestamp):

        self.message = message

        self.timestamp = timestamp

        self.next = None

class ChatHistory:

    def \_\_init\_\_(self):

        self.head = None

    def send\_message(self, message):

        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        new\_message = MessageNode(message, timestamp)

        if not self.head:

            self.head = new\_message

        else:

            temp = self.head

            while temp.next:

                temp = temp.next

            temp.next = new\_message

        print(f'Message sent at {timestamp}: "{message}"')

    def display\_chat(self):

        if not self.head:

            print("No chat history.")

            return

        print("Chat History:")

        temp = self.head

        while temp:

            print(f'{temp.timestamp} - {temp.message}')

            temp = temp.next

# Example Usage

chat = ChatHistory()

chat.send\_message("Hello, how are you?")

chat.send\_message("I am doing well, thank you!")

chat.send\_message("What's up?")

chat.display\_chat()

**Output:**

A screenshot of a computer screen

AI-generated content may be incorrect.

**2. LinkedIn Profile Connections**

o Implement a user profile system where each user is a node connected to other

users.

o Hint: Each node contains a list of connections.

**Code:**

class ProfileNode:

    def \_\_init\_\_(self, username):

        self.username = username

        self.connections = []  # لیسٹ میں کنکشنز رکھے جائیں گے

        self.next = None

class LinkedInNetwork:

    def \_\_init\_\_(self):

        self.head = None

    def add\_profile(self, username):

        new\_profile = ProfileNode(username)

        if not self.head:

            self.head = new\_profile

        else:

            temp = self.head

            while temp.next:

                temp = temp.next

            temp.next = new\_profile

        print(f'Profile "{username}" added to the network.')

    def add\_connection(self, username, connection\_username):

        temp = self.head

        while temp:

            if temp.username == username:

                temp.connections.append(connection\_username)

                print(f'Connection between {username} and {connection\_username} added.')

                return

            temp = temp.next

        print(f'Profile {username} not found.')

    def display\_network(self):

        if not self.head:

            print("No profiles in the network.")

            return

        temp = self.head

        while temp:

            print(f'Profile: {temp.username} | Connections: {", ".join(temp.connections)}')

            temp = temp.next

# Example Usage

network = LinkedInNetwork()

network.add\_profile("john\_doe")

network.add\_profile("alice\_smith")

network.add\_profile("bob\_jones")

network.add\_connection("john\_doe", "alice\_smith")

network.add\_connection("alice\_smith", "bob\_jones")

network.display\_network()

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**3. Google Docs Edit History**

o Simulate edit history tracking in Google Docs.

o Hint: Each node stores a version of the document.

**Code:**

class DocumentNode:

    def \_\_init\_\_(self, version, content):

        self.version = version

        self.content = content

        self.next = None

class GoogleDocsHistory:

    def \_\_init\_\_(self):

        self.head = None

    def add\_version(self, content):

        version = 1

        if self.head:

            temp = self.head

            while temp.next:

                temp = temp.next

            version = temp.version + 1

        new\_version = DocumentNode(version, content)

        if not self.head:

            self.head = new\_version

        else:

            temp.next = new\_version

        print(f'Version {version} added.')

    def display\_history(self):

        if not self.head:

            print("No version history available.")

            return

        temp = self.head

        while temp:

            print(f'Version {temp.version}: {temp.content}')

            temp = temp.next

# Example Usage

docs = GoogleDocsHistory()

docs.add\_version("First version of the document.")

docs.add\_version("Added introduction section.")

docs.add\_version("Corrected some grammatical errors.")

docs.display\_history()

**Output:**

A screenshot of a computer screen

AI-generated content may be incorrect.

**4. Pathfinding Algorithm in Maps**

o Store a series of locations in a linked list and allow traversal.

o Hint: Each node represents a location.

**Code:**

class LocationNode:

def \_\_init\_\_(self, location\_name):

self.location\_name = location\_name

self.next = None

class Pathfinding:

def \_\_init\_\_(self):

self.head = None

def add\_location(self, location\_name):

new\_location = LocationNode(location\_name)

if not self.head:

self.head = new\_location

else:

temp = self.head

while temp.next:

temp = temp.next

temp.next = new\_location

print(f'Location "{location\_name}" added.')

def find\_path(self, start\_location, end\_location):

temp = self.head

path = []

while temp:

path.append(temp.location\_name)

if temp.location\_name == end\_location:

break

temp = temp.next

if end\_location not in path:

print("Path not found.")

return

print("Path found:")

for loc in path:

print(f'- {loc}')

# Example Usage

path = Pathfinding()

path.add\_location("City Center")

path.add\_location("Park")

path.add\_location("Library")

path.add\_location("Museum")

path.find\_path("Park", "Museum")

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**5. Blockchain Implementation**

o Simulate a simple blockchain where each block stores transactions.

o Hint: Use linked list nodes to represent blocks.

**Code:**

class BlockNode:

def \_\_init\_\_(self, block\_number, transactions):

self.block\_number = block\_number

self.transactions = transactions

self.next = None

class Blockchain:

def \_\_init\_\_(self):

self.head = None

def add\_block(self, transactions):

block\_number = 1

if self.head:

temp = self.head

while temp.next:

temp = temp.next

block\_number = temp.block\_number + 1

new\_block = BlockNode(block\_number, transactions)

if not self.head:

self.head = new\_block

else:

temp.next = new\_block

print(f'Block {block\_number} added with transactions: {transactions}')

def display\_chain(self):

if not self.head:

print("No blocks in the chain.")

return

temp = self.head

while temp:

print(f'Block {temp.block\_number}: {temp.transactions}')

temp = temp.next

# Example Usage

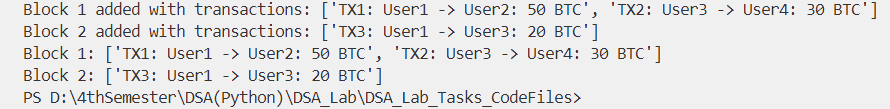
blockchain = Blockchain()

blockchain.add\_block(["TX1: User1 -> User2: 50 BTC", "TX2: User3 -> User4: 30 BTC"])

blockchain.add\_block(["TX3: User1 -> User3: 20 BTC"])

blockchain.display\_chain()

**Output:**

****