Assignment 1
Bridge course: Data Structure and Practices
Indian Institute of Technology, Jodhpur

Topic- Linked List

Q1. Write a program to insert a node at:
a. End of singly linked list
b. Start of singly linked list
c. Middle of a singly linked list.

Code:-

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def display(self):
        current = self.head
        while current:
            print(current.data, end=' -> ')
            current = current.next
        print("None")

    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
```

```
    def insert_at_start(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def insert_at_middle(self, data, position):
        if not self.head or position == 0:
            self.insert_at_start(data)
            return

        new_node = Node(data)
        current = self.head
        count = 1
        while count < position and current.next:
            current = current.next
            count += 1

        new_node.next = current.next
        current.next = new_node


if __name__ == "__main__":
    linked_list = SinglyLinkedList()
    linked_list.insert_at_end(1)
    linked_list.insert_at_end(3)
    linked_list.insert_at_end(5)
    linked_list.insert_at_end(7)

    print("Singly Linked List:")
    linked_list.display()

    linked_list.insert_at_start(0)
    print("Linked List after inserting at start:")
    linked_list.display()

    linked_list.insert_at_middle(2, 2)
    print("Linked List after inserting at middle:")
    linked_list.display()
```

Output:-

```
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Asus/Desktop/IITJ Assignment/Bridge course Assignment/Assignment 1/1_insert_a_node.py"
Singly Linked List:
1 -> 3 -> 5 -> 7 -> None
Linked List after inserting at start:
0 -> 1 -> 3 -> 5 -> 7 -> None
Linked List after inserting at middle:
0 -> 1 -> 2 -> 3 -> 5 -> 7 -> None
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1>
```

Q2. Write a program to remove the nth node from a Linked List.

Code:-

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class LinkedList:
    def __init__(self):
        self.head = None

    def insert_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def remove_nth_node(self, n):
        if n <= 0:
            print("Invalid value of n.")
            return

        if not self.head:
            print("List is empty.")
            return

        if n == 1:
            self.head = self.head.next
            return

        count = 1
        current = self.head
        prev = None
```

```
        while current:
            if count == n:
                prev.next = current.next
                return
            prev = current
            current = current.next
            count += 1

        print("Position out of range.")

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")


# Test the implementation
if __name__ == "__main__":
    linked_list = LinkedList()
    linked_list.insert_end(10)
    linked_list.insert_end(20)
    linked_list.insert_end(30)
    linked_list.insert_end(40)
    linked_list.insert_end(50)

    n = 3  # Remove the 3rd node (30 in this case)
    linked_list.remove_nth_node(n)

    linked_list.display()
```

Output:-


PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Asus/Desktop/IITJ Assignment/Bridge course Assignment/Assignment 1/2_remove_nth_node.py"
10 -> 20 -> 40 -> 50 -> None
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1>

Q3. Write a program to remove the nth node from a Linked List.

Code:-

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def display(self):
        current = self.head
        while current:
            print(current.data, end=' -> ')
            current = current.next
        print("None")

    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def reverse_from_x_to_y(self, x, y):
        if not self.head:
            return

        # Find the node at position (x-1)
        prev_x = None
        current = self.head
        count = 1
```

```
        while current and count < x:
            prev_x = current
            current = current.next
            count += 1

        if not current:
            print(
                "Invalid values of x and y. The linked list does not have
position X or Y.")
            return

        # Reverse the linked list from position X to position Y
        prev = None
        end = current
        for _ in range(y - x + 1):
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node

        if prev_x:
            prev_x.next = prev
        else:
            self.head = prev

        end.next = current


if __name__ == "__main__":
    linked_list = SinglyLinkedList()
    linked_list.insert_at_end(1)
    linked_list.insert_at_end(2)
    linked_list.insert_at_end(3)
    linked_list.insert_at_end(4)
    linked_list.insert_at_end(5)

    print("Original Singly Linked List:")
    linked_list.display()

    x = 2
```

```
    y = 4
    linked_list.reverse_from_x_to_y(x, y)
    print("\nLinked List after reversing from position {} to position
{}:".format(x, y))
    linked_list.display()
```

Output:-

```
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Asus/Des
ktop/IITJ Assignment/Bridge course Assignment/Assignment 1/3_reverse_Linked_List.py"
Original Singly Linked List:
1 -> 2 -> 3 -> 4 -> 5 -> None
Linked List after reversing from position 2 to position 4:
1 -> 4 -> 3 -> 2 -> 5 -> None
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1>
```

Q4. Write a program to merge two sorted Linked lists, L and K.

Code:-

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def display(self):
        current = self.head
        while current:
            print(current.data, end=' -> ')
            current = current.next
        print("None")

    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node


def merge_sorted_lists(l_list, k_list):
    merged_list = SinglyLinkedList()

    l_current = l_list.head
    k_current = k_list.head

    while l_current and k_current:
        if l_current.data < k_current.data:
```

```
            merged_list.insert_at_end(l_current.data)
            l_current = l_current.next
        else:
            merged_list.insert_at_end(k_current.data)
            k_current = k_current.next

    while l_current:
        merged_list.insert_at_end(l_current.data)
        l_current = l_current.next

    while k_current:
        merged_list.insert_at_end(k_current.data)
        k_current = k_current.next

    return merged_list


if __name__ == "__main__":
    l_list = SinglyLinkedList()
    l_list.insert_at_end(1)
    l_list.insert_at_end(3)
    l_list.insert_at_end(5)

    k_list = SinglyLinkedList()
    k_list.insert_at_end(2)
    k_list.insert_at_end(4)
    k_list.insert_at_end(6)

    print("Sorted Linked List L:")
    l_list.display()

    print("\nSorted Linked List K:")
    k_list.display()

    merged_list = merge_sorted_lists(l_list, k_list)
    print("\nMerged Sorted Linked List:")
    merged_list.display()
```

Output:-

```
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Asus/Desktop/IITJ Assignment/Bridge course Assignment/Assignment 1/4_merge_two_sorted Linked_lists.py"
Sorted Linked List L:
1 -> 3 -> 5 -> None
Sorted Linked List K:
2 -> 4 -> 6 -> None
Merged Sorted Linked List:
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> None
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1>
```

Q5. Write a program to remove duplicate elements from an unsorted Linked List.

Code:-

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def display(self):
        current = self.head
        while current:
            print(current.data, end=' -> ')
            current = current.next
        print("None")

    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def remove_duplicates(self):
        if not self.head:
            return

        seen = set()
        current = self.head
        seen.add(current.data)

        while current.next:
```

```
            if current.next.data in seen:
                current.next = current.next.next
            else:
                seen.add(current.next.data)
                current = current.next


if __name__ == "__main__":
    linked_list = SinglyLinkedList()
    linked_list.insert_at_end(3)
    linked_list.insert_at_end(1)
    linked_list.insert_at_end(2)
    linked_list.insert_at_end(2)
    linked_list.insert_at_end(4)
    linked_list.insert_at_end(3)
    linked_list.insert_at_end(5)

    print("Original Linked List:")
    linked_list.display()

    linked_list.remove_duplicates()

    print("\nLinked List after removing duplicates:")
    linked_list.display()
```

Output:-

```
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Asus/Desktop/IITJ Assignment/Bridge course Assignment/Assignment 1/5_remove_duplicate_unsorted_Linked_List.py"
Original Linked List:
3 -> 1 -> 2 -> 2 -> 4 -> 3 -> 5 -> None

Linked List after removing duplicates:
3 -> 1 -> 2 -> 4 -> 5 -> None
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1>
```

Q6. Write a program to create a doubly linked list with functions (inset at beginning, insert at end, delete from beginning, and delete from end) and display it in reverse order.

Code:-

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None


class DoublyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def insert_at_beginning(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = self.tail = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node

    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = self.tail = new_node
        else:
            new_node.prev = self.tail
            self.tail.next = new_node
            self.tail = new_node

    def delete_from_beginning(self):
        if not self.head:
            return
        if self.head == self.tail:
```

```
            self.head = self.tail = None
        else:
            self.head = self.head.next
            self.head.prev = None

    def delete_from_end(self):
        if not self.tail:
            return
        if self.head == self.tail:
            self.head = self.tail = None
        else:
            self.tail = self.tail.prev
            self.tail.next = None

    def display_reverse(self):
        current = self.tail
        while current:
            print(current.data, end=' <-> ')
            current = current.prev
        print("None")


if __name__ == "__main__":
    linked_list = DoublyLinkedList()

    linked_list.insert_at_end(1)
    linked_list.insert_at_end(2)
    linked_list.insert_at_end(3)
    linked_list.insert_at_beginning(0)

    print("Doubly Linked List:")
    linked_list.display_reverse()

    linked_list.delete_from_beginning()
    linked_list.delete_from_end()

    print("\nDoubly Linked List after deleting from beginning and end:")
    linked_list.display_reverse()
```

Output:-

```
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Asus/Desktop/IITJ Assignment/Bridge course Assignment/Assignment 1/6_create_doubly_linked_list_with_functions.py"
Doubly Linked List:
3 <-> 2 <-> 1 <-> 0 <-> None

Doubly Linked List after deleting from beginning and end:
2 <-> 1 <-> None
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1>
```

Q7. Write a program to find the maximum value in a doubly linked list.

Code:-

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None


class DoublyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = self.tail = new_node
        else:
            new_node.prev = self.tail
            self.tail.next = new_node
            self.tail = new_node

    def find_max_value(self):
        if not self.head:
            return None

        max_value = self.head.data
        current = self.head.next
        while current:
            if current.data > max_value:
                max_value = current.data
            current = current.next

        return max_value


if __name__ == "__main__":
```

```
linked_list = DoublyLinkedList()

linked_list.insert_at_end(10)
linked_list.insert_at_end(5)
linked_list.insert_at_end(20)
linked_list.insert_at_end(15)

max_value = linked_list.find_max_value()
if max_value is not None:
    print("Maximum value in the doubly linked list:", max_value)
else:
    print("The doubly linked list is empty.")
```

Output:-

```
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Asus/Desktop/IITJ Assignment/Bridge course Assignment/Assignment 1/7_maximum_value_in_doubly_linked_list.py"
Maximum value in the doubly linked list: 20
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1>
```

Q8. Write a program to create a circular linked list and insert a node at any position in the list.

Code:-

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class CircularLinkedList:
    def __init__(self):
        self.head = None

    def display(self):
        if not self.head:
            print("Circular Linked List is empty.")
            return

        current = self.head
        while True:
            print(current.data, end=" -> ")
            current = current.next
            if current == self.head:
                break
        print("Head")

    def insert_at_position(self, data, position):
        new_node = Node(data)

        if not self.head:  # If the list is empty, make new_node as head
and circular.
            new_node.next = new_node
            self.head = new_node
        else:
            current = self.head
            for _ in range(position - 1):
                current = current.next
```

```
            new_node.next = current.next
            current.next = new_node


if __name__ == "__main__":
    circular_linked_list = CircularLinkedList()

    circular_linked_list.insert_at_position(10, 0)
    circular_linked_list.insert_at_position(20, 1)
    circular_linked_list.insert_at_position(30, 2)

    circular_linked_list.display()
```

Output:-

```
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Asus/Desktop/IITJ Assignment/Bridge course Assignment/Assignment 1/8_create_circular_linked_list_&_insert_node.py"
10 -> 20 -> 30 -> Head
PS C:\Users\Asus\Desktop\IITJ Assignment\Bridge course Assignment\Assignment 1>
```

Q9. Write a program in C to delete a node from the middle of a circular linked list.

Code:-

```
# include <stdio.h>
# include <stdlib.h>

struct Node {
    int data
    struct Node * next
}

struct Node * createNode(int data) {
    struct Node * newNode = (struct Node*)malloc(sizeof(struct Node))
    if (newNode == NULL) {
        printf("Memory allocation failed.")
        exit(1)
    }
    newNode -> data = data
    newNode -> next = NULL
    return newNode
}

void insertAtEnd(struct Node ** head, int data) {
    struct Node * newNode = createNode(data)
    if (*head == NULL) {
        *head = newNode
        newNode -> next = *head
    } else {
        struct Node * temp = *head
        while (temp -> next != *head) {
            temp = temp -> next
        }
        temp -> next = newNode
        newNode -> next = *head
    }
}

void deleteFromMiddle(struct Node ** head) {
    if (*head == NULL | | (*head) -> next == *head) {
```

```
        printf("Cannot delete from an empty list or a list with only one
node.")
        return
    }

    struct Node * slow = *head
    struct Node * fast = *head
    struct Node * prev = NULL

    while (fast != *head & & fast -> next != *head) {
        fast = fast -> next -> next
        prev = slow
        slow = slow -> next
    }

    if (prev != NULL) {
        prev -> next = slow -> next
        free(slow)
    }
}

void display(struct Node * head) {
    if (head == NULL) {
        printf("Circular Linked List is empty.")
        return
    }

    struct Node * current = head
    do {
        printf("%d -> ", current -> data)
        current = current -> next
    } while (current != head)
    printf("Head\n")
}

int main() {
    struct Node * head = NULL

    insertAtEnd(& head, 10)
    insertAtEnd(& head, 20)
```

```
    insertAtEnd(& head, 30)
    insertAtEnd(& head, 40)
    insertAtEnd(& head, 50)

    printf("Circular Linked List before deletion:\n")
    display(head)

    deleteFromMiddle(& head)

    printf("\nCircular Linked List after deletion:\n")
    display(head)

    return 0
}
```

Output:-

```
input
Original Circular Linked List:
10 -> 20 -> 30 -> 40 -> 50 -> Head


...Program finished with exit code 0
Press ENTER to exit console.
```

Q10. Write a program in C to delete a node from any position in a doubly linked list.

Code:-

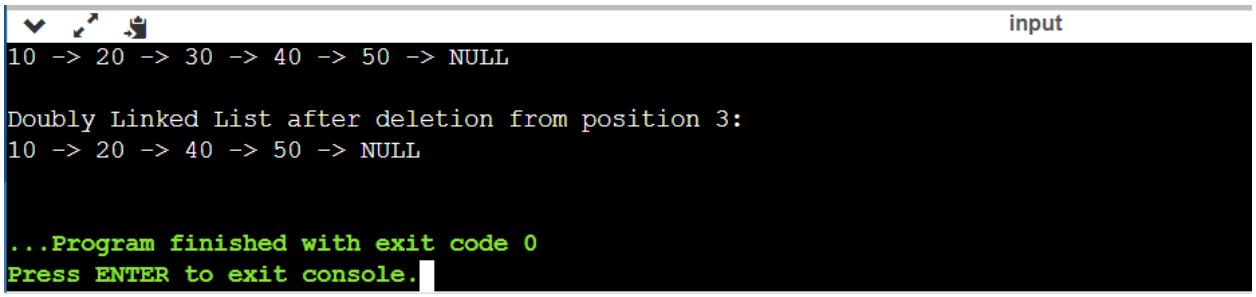
```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
```

```
            current = current->next;
        }
        current->next = newNode;
        newNode->prev = current;
    }
}

void deleteFromPosition(struct Node** head, int position) {
    if (*head == NULL) {
        printf("Cannot delete from an empty list.");
        return;
    }

    if (position == 1) {
        struct Node* temp = *head;
        *head = (*head)->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
        free(temp);
        return;
    }

    int count = 1;
    struct Node* current = *head;
    while (current != NULL && count < position) {
        current = current->next;
        count++;
    }

    if (current == NULL) {
        printf("Invalid position to delete.");
        return;
    }

    if (current->next != NULL) {
        current->next->prev = current->prev;
    }

    if (current->prev != NULL) {
```

```
        current->prev->next = current->next;
    }

    free(current);
}

void display(struct Node* head) {
    if (head == NULL) {
        printf("Doubly Linked List is empty.");
        return;
    }

    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;

    insertAtEnd(&head, 10);
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 30);
    insertAtEnd(&head, 40);
    insertAtEnd(&head, 50);

    printf("Doubly Linked List before deletion:\n");
    display(head);

    int positionToDelete = 3;
    deleteFromPosition(&head, positionToDelete);

    printf("\nDoubly Linked List after deletion from position %d:\n",
positionToDelete);
    display(head);

    return 0;
```

```
}
```

Output:-


input
10 -> 20 -> 30 -> 40 -> 50 -> NULL
Doubly Linked List after deletion from position 3:
10 -> 20 -> 40 -> 50 -> NULL
...Program finished with exit code 0
Press ENTER to exit console.