**NAME** :- Manish Shashikant Jadhav

**UID** :- 2023301005.

**BRANCH** :- Comps -B. **BRANCH:** B.

**EXPERIMENT 4: Implement of given problem statement using Doubly Linked List.**

**SUBJECT** :- DS (DATA STRUCTURES).

**CODE** :-

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

typedef struct Node {

    int val;

    struct Node\* prev;

    struct Node\* next;

} Node;

// Create a new node and return a pointer to it

Node\* create\_node(int val) {

    Node\* new\_node = (Node\*)malloc(sizeof(Node));

    if (new\_node == NULL) {

        fprintf(stderr, "Memory allocation failed\n");

        exit(EXIT\_FAILURE);

    }

    new\_node->val = val;

    new\_node->prev = NULL;

    new\_node->next = NULL;

    return new\_node;

}

// Insert a new node with 'val' at the specified position

void insert\_at\_pos(Node\*\* header, Node\*\* trailer, int val, int pos, bool is\_after) {

    Node\* new\_node = create\_node(val);

    if (pos == 0 && is\_after) {

        new\_node->next = (\*header)->next;

        new\_node->prev = \*header;

        if ((\*header)->next != NULL) {

            (\*header)->next->prev = new\_node;

        }

        (\*header)->next = new\_node;

    } else if (pos == -1 && !is\_after) {

        new\_node->prev = (\*trailer)->prev;

        new\_node->next = \*trailer;

        if ((\*trailer)->prev != NULL) {

            (\*trailer)->prev->next = new\_node;

        }

        (\*trailer)->prev = new\_node;

    } else {

        Node\* current = (\*header)->next;

        int current\_pos = 0;

        while (current != \*trailer && current\_pos != pos) {

            current = current->next;

            current\_pos++;

        }

        if (current == \*trailer && is\_after) {

            fprintf(stderr, "Position %d is out of bounds for insertion\n", pos);

            free(new\_node);

            return;

        }

        if (is\_after) {

            new\_node->prev = current;

            new\_node->next = current->next;

            current->next = new\_node;

            new\_node->next->prev = new\_node;

        } else {

            new\_node->prev = current->prev;

            new\_node->next = current;

            current->prev = new\_node;

            new\_node->prev->next = new\_node;

        }

    }

}

// Delete a node at the specified position

void delete\_at\_pos(Node\*\* header, Node\*\* trailer, int pos) {

    if (pos == 0) {

        fprintf(stderr, "Cannot delete the header node\n");

        return;

    }

    Node\* current = (\*header)->next;

    int current\_pos = 0;

    while (current != \*trailer && current\_pos != pos) {

        current = current->next;

        current\_pos++;

    }

    if (current == \*trailer) {

        fprintf(stderr, "Position %d is out of bounds for deletion\n", pos);

        return;

    }

    current->prev->next = current->next;

    current->next->prev = current->prev;

    free(current);

}

// Reverse the doubly linked list

void reverse(Node \*\*header) {

    Node \*temp = NULL;

    Node \*current = \*header;

    while (current != NULL) {

        temp = current->prev;

        current->prev = current->next;

        current->next = temp;

        current = current->prev;

    }

    if (temp != NULL) {

        \*header = temp->prev;

    } else {

        \*header = current;

    }

}

// Swap nodes at two specified positions

void swap(Node\*\* header, Node\*\* trailer, int pos\_1, int pos\_2) {

    if (pos\_1 == pos\_2) {

        return; // No need to swap if the positions are the same.

    }

    Node\* node\_1 = (\*header)->next;

    int current\_pos = 0;

    while (node\_1 != \*trailer && current\_pos != pos\_1) {

        node\_1 = node\_1->next;

        current\_pos++;

    }

    Node\* node\_2 = (\*header)->next;

    current\_pos = 0;

    while (node\_2 != \*trailer && current\_pos != pos\_2) {

        node\_2 = node\_2->next;

        current\_pos++;

    }

    if (node\_1 == \*trailer || node\_2 == \*trailer) {

        fprintf(stderr, "One or both positions are out of bounds for swapping\n");

        return;

    }

    // Swap the nodes

    if (node\_1->prev != NULL) {

        node\_1->prev->next = node\_2;

    } else {

        (\*header)->next = node\_2;

    }

    if (node\_2->prev != NULL) {

        node\_2->prev->next = node\_1;

    } else {

        (\*header)->next = node\_1;

    }

    Node\* temp\_prev = node\_1->prev;

    node\_1->prev = node\_2->prev;

    node\_2->prev = temp\_prev;

    Node\* temp\_next = node\_1->next;

    node\_1->next = node\_2->next;

    node\_2->next = temp\_next;

}

// Display the elements of the doubly linked list

void display(Node\* header, Node\* trailer) {

    Node\* current = header->next;

    printf("Doubly Linked List: ");

    while (current != trailer) {

        printf("%d ", current->val);

        current = current->next;

    }

    printf("\n");

}

int main() {

    Node\* header = create\_node(-1); // Dummy header node

    Node\* trailer = create\_node(-1); // Dummy trailer node

    header->next = trailer;

    trailer->prev = header;

    // Insert elements

    insert\_at\_pos(&header, &trailer, 28, 0, true);

    insert\_at\_pos(&header, &trailer, 9, 0, true);

    insert\_at\_pos(&header, &trailer, 13, 0, true);

    insert\_at\_pos(&header, &trailer, 2, 0, true);

    insert\_at\_pos(&header, &trailer, 45, 0, true);

    display(header, trailer);

    //reverse(&header);

    swap(&header, &trailer, 0, 2);

    printf("Doubly linked list after swapping");

    printf("\n");

    display(header, trailer);

    delete\_at\_pos(&header,&trailer,3);

    //printf("Doubly linked list after deletion of node");

    ///printf("\n");

    ///display(header, trailer);

    // Free memory and return

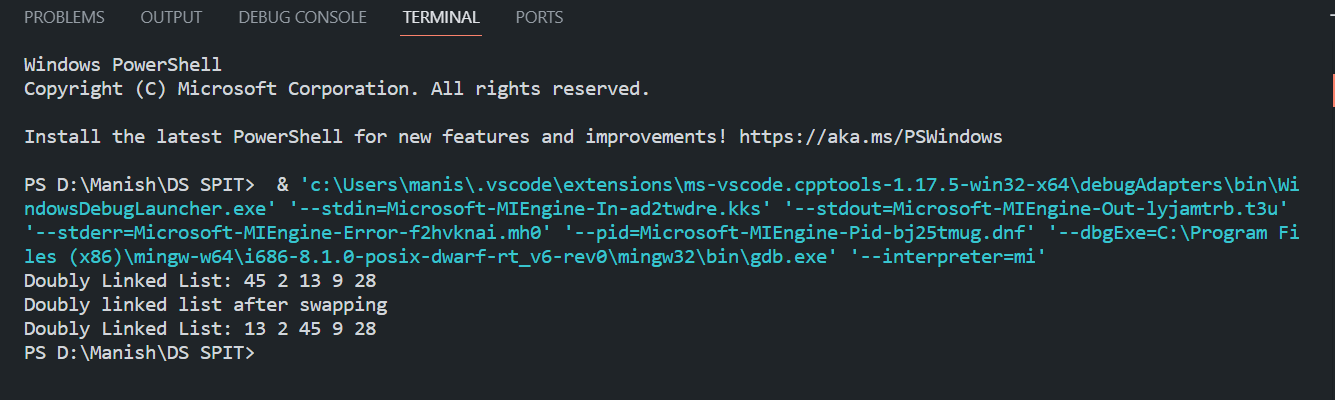
    free(header);

    free(trailer);

    return 0;

}

**Output:**

****

**Algorithm:**

1. **Node Structure:-**

Define a structure for a doubly linked list node with integer values and pointers to the previous and next nodes.

1. **Create a Node:-**

Implement a create\_node function that allocates memory for a new node, initializes it with the given value, and returns a pointer to the new node.

1. **Insertion at a Position:-**

Implement the insert\_at\_pos function that inserts a new node with a specified value at a given position in the doubly linked list.

If pos is 0 and is\_after is true, insert at the beginning.

If pos is -1 and is\_after is false, insert at the end.

Otherwise, insert at the specified position.

1. **Deletion at a Position:-**

Implement the delete\_at\_pos function that deletes a node at the specified position in the doubly linked list.

Ensure that you do not delete the header node.

1. **Reversing the List:-**

Implement the reverse function that reverses the doubly linked list in place.

1. **Swapping Nodes:-**

Implement the swap function that swaps nodes at two specified positions in the doubly linked list.

1. **Displaying the List:-**

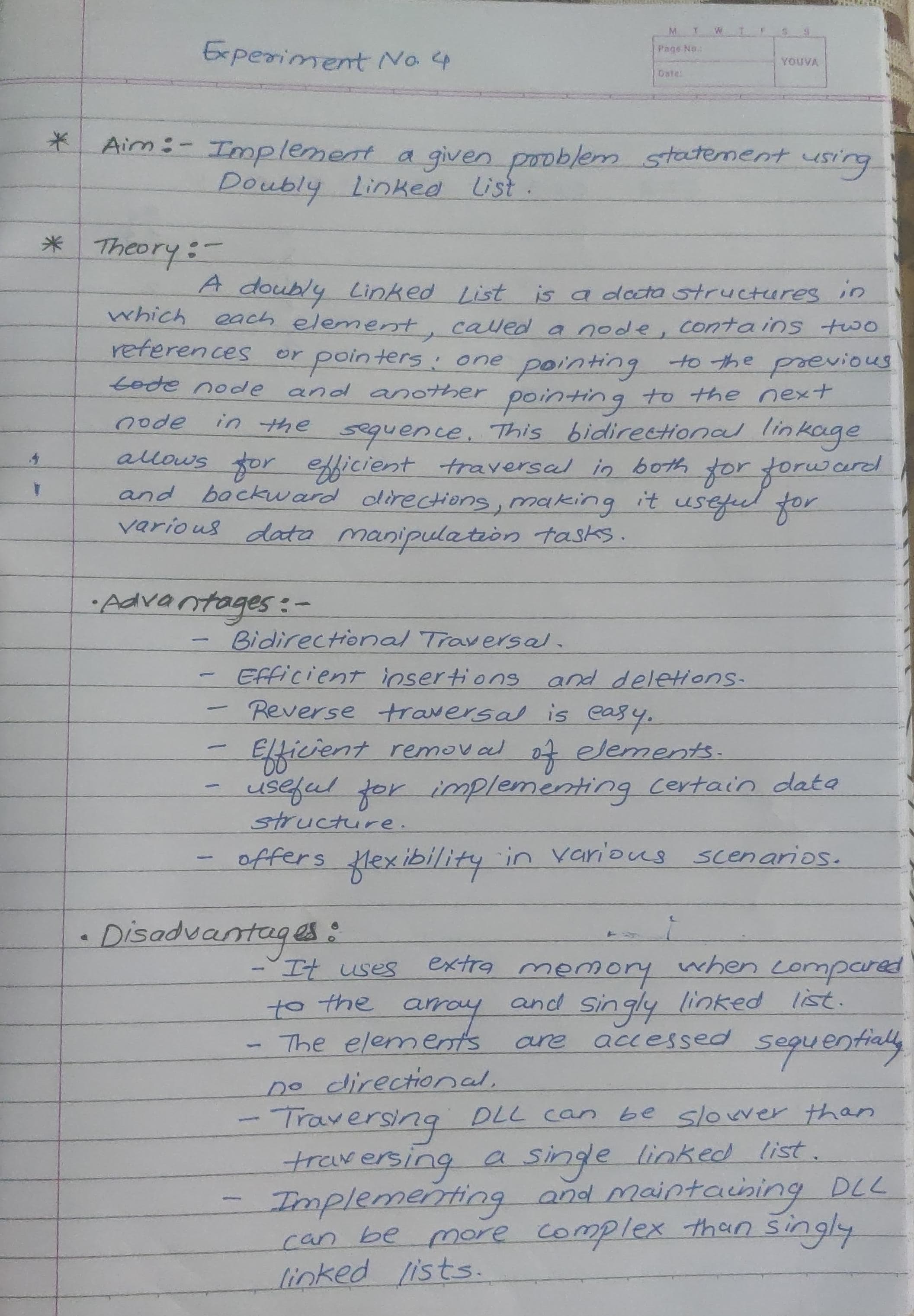
Implement the display function to print the elements of the doubly linked list.

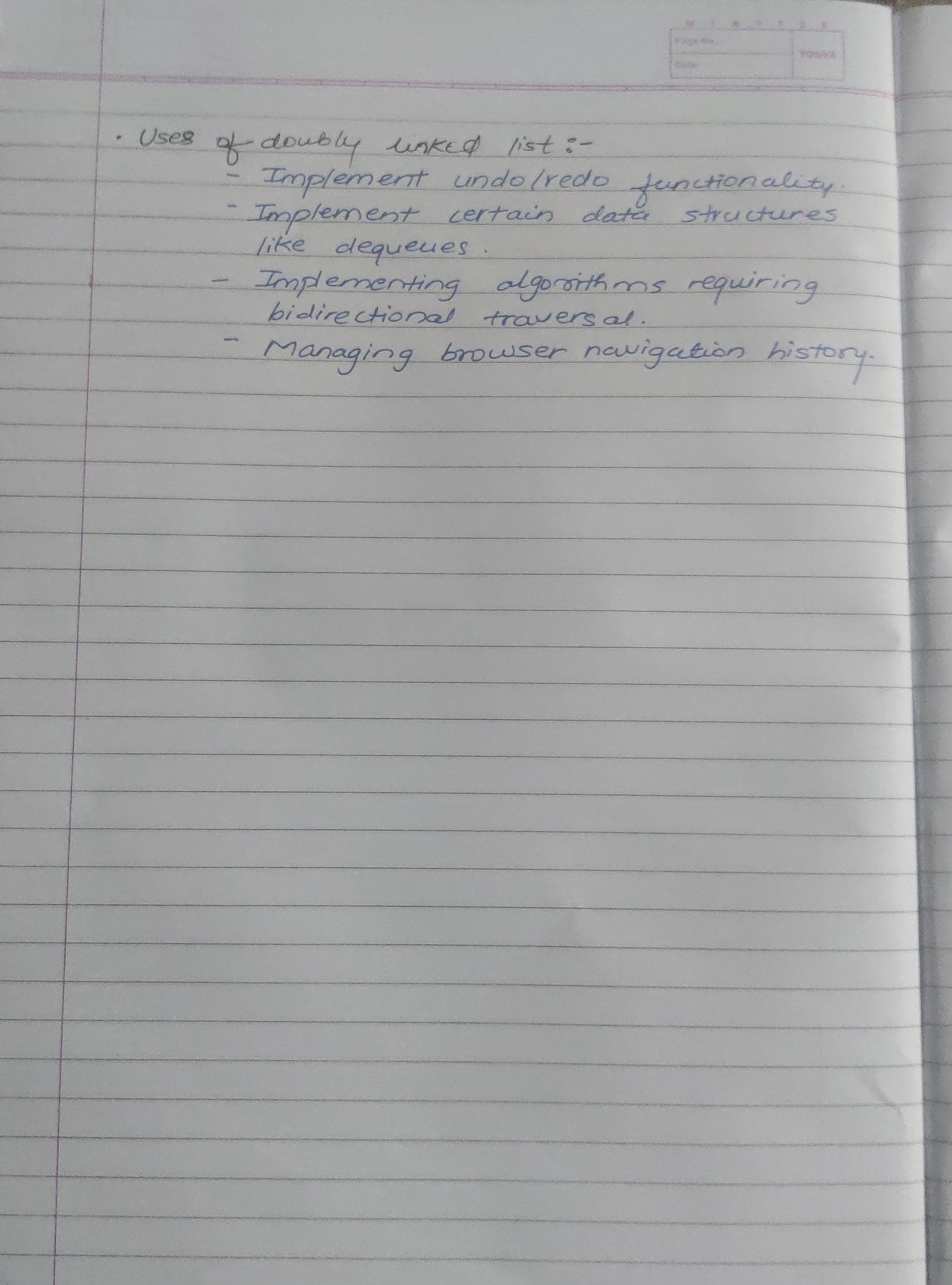
1. **Main Function:-**

In the main function,

Create a header and trailer node to represent the dummy nodes at the beginning and end of the list.

Perform various operations on the doubly linked list, such as inserting, swapping, deleting, reversing, and displaying.





**Conclusion:**

Hence, by completing this experiment I came to know about implementation of given problem statement for Doubly Linked List.