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 :-

```c
#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;
    }
}
```

```c
// 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:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Manish\DS SPIT>  & 'c:\Users\manis\.vscode\extensions\ms-vscode.cpptools-1.17.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-ad2twdre.kks' '--stdout=Microsoft-MIEngine-Out-lyjamtrb.t3u' '--stderr=Microsoft-MIEngine-Error-f2hvknai.mh0' '--pid=Microsoft-MIEngine-Pid-bj25tmug.dnf' '--dbgExe=C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin\gdb.exe' '--interpreter=mi'
Doubly Linked List: 45 2 13 9 28
Doubly linked list after swapping
Doubly Linked List: 13 2 45 9 28
PS D:\Manish\DS SPIT>
```

Algorithm:

1. **Node Structure:-**

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

2. **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.

3. **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.

4. **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.

5. **Reversing the List:-**

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

6. **Swapping Nodes:-**

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

7. **Displaying the List:-**

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

8. **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.

Experiment No. 4

* Aim :- Implement a given problem statement using Doubly Linked List.

* Theory :-

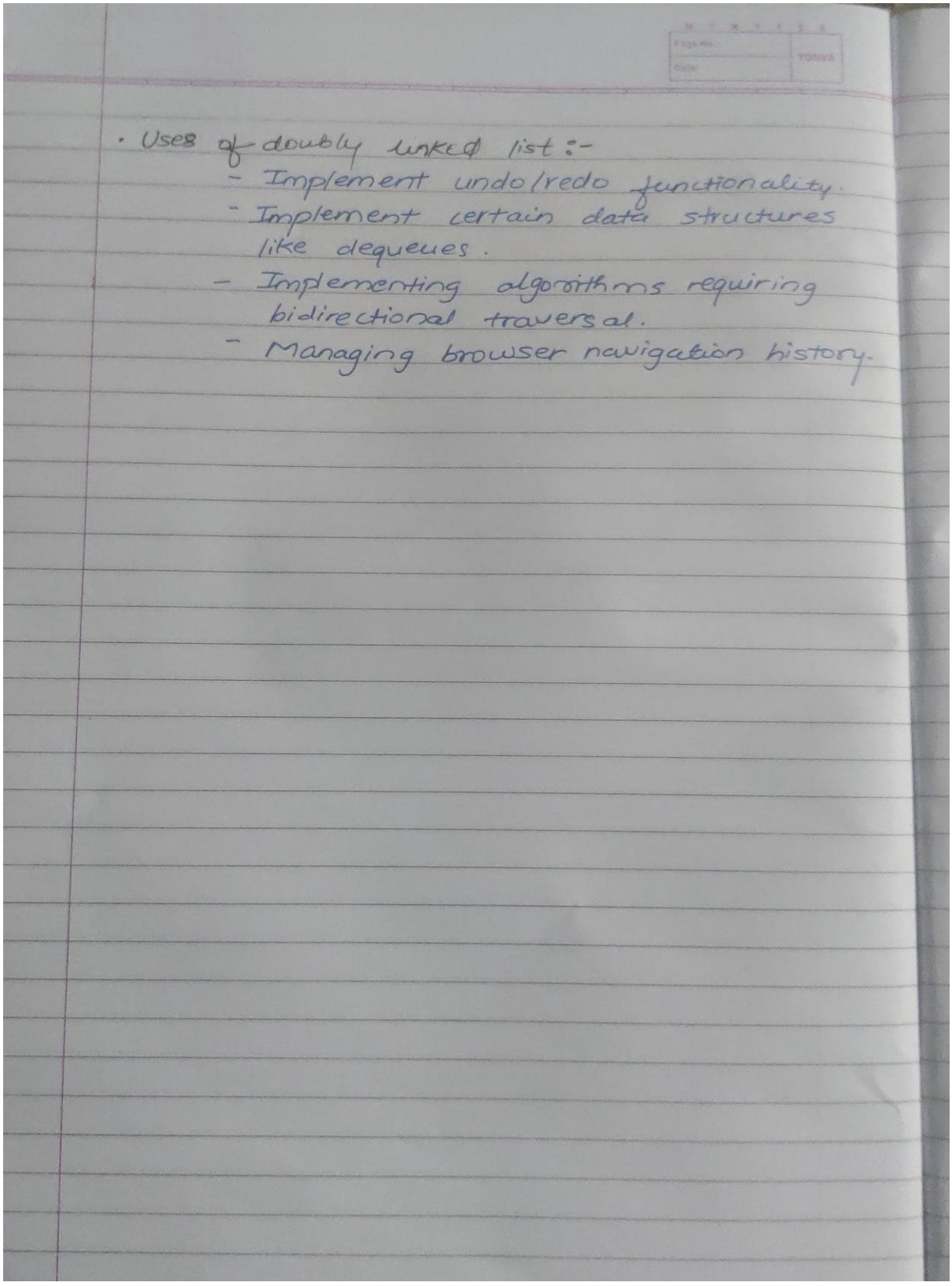
A doubly linked list is a data structures in which each element, called a node, contains two references or pointers : one pointing to the previous ~~code~~ node and another pointing to the next node in the sequence. This bidirectional linkage allows for efficient traversal in both for forward and backward directions, making it useful for various data manipulation tasks.

• Advantages :-

- Bidirectional Traversal.
- Efficient insertions and deletions.
- Reverse traversal is easy.
- Efficient removal of elements.
- useful for implementing certain data structure.
- offers flexibility in various scenarios.

• Disadvantages :

- It uses extra memory when compared to the array and singly linked list.
- The elements are accessed sequentially no directional.
- Traversing DLL can be slower than traversing a single linked list.
- Implementing and maintaining DLL can be more complex than singly linked lists.



- Uses of doubly linked list :-
  - Implement undo/redo functionality.
  - Implement certain data structures like dequeues.
  - Implementing algorithms requiring bidirectional traversal.
  - Managing browser navigation history.

Conclusion:

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