Aim ↬ To perform basic operations on Doubly Linked List.

Objectives ↬

i. Write a program to create a doubly linked list of integers entered by user and traverse the list:
   a. From first to last.
   b. From last to first.

ii. Write a program to create a doubly linked list of integers entered by user and insert a new node:
   a. After a node having a key value.
   b. Before a node having a key value.

iii. Write a program to delete a node having given value from a doubly linked list.

Software Required ↬ Visual Studio Code

Code 1 ↬

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

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

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

struct Node* addNode(struct Node* head, int value) {
    struct Node* newNode = createNode(value);
    if (head == NULL) {
        return newNode;
```

```
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
        return head;
    }
}

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

void traverseBackward(struct Node* head) {
    struct Node* temp = head;
    if (temp == NULL) {
        return;
    }
    while (temp->next != NULL) {
        temp = temp->next;
    }
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->prev;
    }
    printf("\n");
}

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

```
    int n, value, i;

    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("Enter value for node %d: ", i + 1);
        scanf("%d", &value);
        head = addNode(head, value);
    }

    printf("List from first to last: ");
    traverseForward(head);

    printf("List from last to first: ");
    traverseBackward(head);

    return 0;
}
```

Output ↬

```
Enter the number of nodes: 5
Enter value for node 1: 2
Enter value for node 2: -3
Enter value for node 3: 6
Enter value for node 4: -1
Enter value for node 5: 8
List from first to last: 2 -3 6 -1 8
List from last to first: 8 -1 6 -3 2
```

Code 2 ↬

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

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

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

struct Node* addNode(struct Node* head, int value) {
    struct Node* newNode = createNode(value);
    if (head == NULL) {
        return newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
        return head;
    }
}

struct Node* insertAfter(struct Node* head, int key, int value) {
    struct Node* temp = head;
    while (temp != NULL && temp->data != key) {
        temp = temp->next;
```

```
    }
    if (temp != NULL) {
        struct Node* newNode = createNode(value);
        newNode->next = temp->next;
        newNode->prev = temp;
        if (temp->next != NULL) {
            temp->next->prev = newNode;
        }
        temp->next = newNode;
    }
    return head;
}

struct Node* insertBefore(struct Node* head, int key, int value) {
    struct Node* temp = head;
    while (temp != NULL && temp->data != key) {
        temp = temp->next;
    }
    if (temp != NULL) {
        struct Node* newNode = createNode(value);
        newNode->next = temp;
        newNode->prev = temp->prev;
        if (temp->prev != NULL) {
            temp->prev->next = newNode;
        } else {
            head = newNode;
        }
        temp->prev = newNode;
    }
    return head;
}

void traverseForward(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
```

```
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    int n, value, key, i, choice;

    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("Enter value for node %d: ", i + 1);
        scanf("%d", &value);
        head = addNode(head, value);
    }

    printf("Enter 1 to insert after a key, 2 to insert before a key: ");
    scanf("%d", &choice);

    printf("Enter the key value: ");
    scanf("%d", &key);

    printf("Enter the new value to insert: ");
    scanf("%d", &value);

    if (choice == 1) {
        head = insertAfter(head, key, value);
    } else if (choice == 2) {
        head = insertBefore(head, key, value);
    }

    printf("List after insertion: ");
    traverseForward(head);

    return 0;
}
```

Output ↬

```
Enter the number of nodes: 5
Enter value for node 1: 11
Enter value for node 2: 22
Enter value for node 3: 33
Enter value for node 4: 44
Enter value for node 5: 55
Enter 1 to insert after a key, 2 to insert before a key: 1
Enter the key value: 33
Enter the new value to insert: 23
List after insertion: 11 22 33 23 44 55
```

```
Enter the number of nodes: 6
Enter value for node 1: 11
Enter value for node 2: 44
Enter value for node 3: 66
Enter value for node 4: 88
Enter value for node 5: 99
Enter value for node 6: 7
Enter 1 to insert after a key, 2 to insert before a key: 2
Enter the key value: 88
Enter the new value to insert: 34
List after insertion: 11 44 66 34 88 99 7
```

Code 3 ↬

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

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

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->prev = NULL;
```

```
    newNode->next = NULL;
    return newNode;
}

struct Node* addNode(struct Node* head, int value) {
    struct Node* newNode = createNode(value);
    if (head == NULL) {
        return newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
        return head;
    }
}

struct Node* deleteNode(struct Node* head, int value) {
    struct Node* temp = head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }
    if (temp == NULL) {
        return head;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next;
    }
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    free(temp);
    return head;
```

```
}

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

int main() {
    struct Node* head = NULL;
    int n, value, i, key;

    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("Enter value for node %d: ", i + 1);
        scanf("%d", &value);
        head = addNode(head, value);
    }

    printf("Enter the value of the node to delete: ");
    scanf("%d", &key);

    head = deleteNode(head, key);

    printf("List after deletion: ");
    traverseForward(head);

    return 0;
}
```

Output ↬

```
Enter the number of nodes: 5
Enter value for node 1: 11
Enter value for node 2: 44
Enter value for node 3: 34
Enter value for node 4: 54
Enter value for node 5: 67
Enter the value of the node to delete: 34
List after deletion: 11 44 54 67
```

Result ↬

The programs successfully performed various doubly linked list operations, including:

- **Traversal**: Displayed the linked list elements from the first to the last and vice versa.
- **Insertion**: Inserted a new node either after or before a node with a given key value.
- **Deletion**: Removed a node from the list based on a given value. These operations demonstrated efficient dynamic memory management and proper manipulation of pointers in a doubly linked list.

Conclusion ↬

The experiment effectively showcased fundamental doubly linked list operations in C, including traversing, inserting, and deleting nodes. The programs illustrated the correct techniques for creating and managing a doubly linked list, enhancing the understanding of dynamic memory allocation and manipulation in linked data structures.

Precautions ↬

- Ensure valid inputs for all operations.
- Properly manage memory allocation and deallocation.
- Handle empty list cases for insertion and deletion.
- Implement error handling for missing nodes or keys.