1. WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

Code:

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

struct Node {
    int val;
    struct Node* next;
};

void sortList(struct Node** node);
void create(struct Node** node);
void display(struct Node** node);
void insert(struct Node** node, int value);
void reverse(struct Node** node);
void concat(struct Node** node1, struct Node** node2);

int main() {
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    printf("Create LL 1 : \n");
    create(&head1);
    printf("Create LL 2 : \n");
    create(&head2);

    printf("Concatination of two lists is : \n");
    concat(&head1, &head2);
    display(&head1);

    printf("Sorting of this list : \n");
    sortList(&head1);
    display(&head1);

    printf("Reversing of this list : \n");
    reverse(&head1);
}

void create(struct Node** node) {
    int ch, val;
    while (1) {
        printf("1. Insert\n2. Exit\n");
        scanf("%d", &ch);
```

```
        switch (ch) {
            case 1:
                printf("Enter the value : ");
                scanf("%d", &val);
                insert(node, val);
                break;
            case 2:
                return;
            default:
                printf("Invalid choice\n");
        }
    }
}

void insert(struct Node** node, int value) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->val = value;
    new_node->next = *node;
    *node = new_node;
}

void sortList(struct Node** node) {
    struct Node *temp, *i;
    for (temp = *node; temp != NULL; temp = temp->next) {
        for (i = temp->next; i != NULL; i = i->next) {
            if (i->val < temp->val) {
                int tem = i->val;
                i->val = temp->val;
                temp->val = tem;
            }
        }
    }
}
void display(struct Node** node) {
    struct Node* temp = *node;
    while (temp != NULL) {
        printf("%d->", temp->val);
        temp = temp->next;
    }
    printf("NULL\n");
}
void reverse(struct Node* *node) {
    struct Node* temp = *node;
    struct Node* curr = temp;
```

```
    struct Node* prev = NULL;
    struct Node* nextOne = NULL;

    while(curr != NULL) {
        nextOne = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextOne;
    }
    display(&prev);
}
void concat(struct Node* *node1, struct Node* *node2) {
    struct Node* temp1 = *node1;
    struct Node* temp2 = *node2;

    struct Node* dummy = temp1;
    while(dummy->next != NULL) dummy = dummy->next;

    dummy->next = temp2;
}
```

Output:

```
Create LL 1 :
1. Insert
2. Exit
1
Enter the value : 2
1. Insert
2. Exit
1
Enter the value : 3
1. Insert
2. Exit
1
Enter the value : 4
1. Insert
2. Exit
1
Enter the value : 5
1. Insert
2. Exit
1
Enter the value : 6
1. Insert
2. Exit
2
Create LL 2 :
1. Insert
2. Exit
1
```

```
Enter the value : 3
1. Insert
2. Exit
1
Enter the value : 5
1. Insert
2. Exit
1
Enter the value : 4
1. Insert
2. Exit
1
Enter the value : 2
1. Insert
2. Exit
2
Concatination of two lists is :
6->5->4->3->2->2->4->5->3->NULL
Sorting of this list :
2->2->3->3->4->4->5->5->6->NULL
Reversing of this list :
6->5->5->4->4->3->3->2->2->NULL
```

2. WAP to Implement doubly link list with primitive operations
I.Create a doubly linked list.
II. Insert a new node to the left of the node.
III. Delete the node based on a specific value
IV. Display the contents of the list

Code:

```
#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\n");
        return NULL;
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);

    if (*head == NULL) {
        *head = newNode;
    } else {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    }
}
void insertBeforeNode(struct Node** head, int key, int data) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* newNode = createNode(data);
    struct Node* current = *head;

    while (current) {
        if (current->data == key) {
            if (current->prev) {
                current->prev->next = newNode;
                newNode->prev = current->prev;
            } else {
                *head = newNode;
            }

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

```
        current = current->next;
    }

    printf("Key not found in the list\n");
}


void deleteNode(struct Node** head, int pos) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* current = *head;
    int count = 1;

    while (current && count < pos) {
        current = current->next;
        count++;
    }

    if (current == NULL) {
        printf("Position %d is beyond the length of the list\n", pos);
        return;
    }

    if (current->prev) {
        current->prev->next = current->next;
    } else {
        *head = current->next;
    }

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

    free(current);
    printf("Node at position %d deleted\n", pos);
}

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
```

```
    }

    struct Node* current = head;

    while (current) {
        printf("%d-> ", current->data);
        current = current->next;
    }
    printf("NULL");
}

void freeList(struct Node* head) {
    struct Node* current = head;
    struct Node* nextNode;

    while (current) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
}

int main() {
    struct Node* head = NULL;
    int ch, newData, pos, key;

    while (1) {
        printf("\nMenu\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert before a node\n");
        printf("3. Delete a node\n");
        printf("4. Display list\n");
        printf("5. Free doubly linked list and exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &newData);
                insertAtBeginning(&head, newData);
                break;

            case 2:
```

```
            printf("Enter the value before which you want to insert: ");
            scanf("%d", &key);
            printf("Enter data to insert: ");
            scanf("%d", &newData);
            insertBeforeNode(&head, key, newData);
            break;

        case 3:
            printf("Enter the position you wish to delete: ");
            scanf("%d", &key);
            deleteNode(&head, key);
            break;

        case 4:
            printf("Doubly linked list: ");
            displayList(head);
            break;

        case 5:
            freeList(head);
            printf("Exiting the program\n");
            return 0;

        default:
            printf("Invalid choice\n");
      }
   }

   return 0;
}
```

Output:

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 2
Enter the value before which you want to insert: 3
Enter data to insert: 1
List is empty

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 1
Enter data to insert at the beginning: 4

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 1
Enter data to insert at the beginning: 3

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 2
Enter the value before which you want to insert: 5
Enter data to insert: 1
Key not found in the list
```

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 2
Enter the value before which you want to insert: 3
Enter data to insert: 5

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 5-> 3-> 4-> NULL
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 1
Enter data to insert at the beginning: 6

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 3
Enter the position you wish to delete: 1
Node at position 1 deleted
```

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 5-> 3-> 4-> NULL
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 3
Enter the position you wish to delete: 3
Node at position 3 deleted

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 5-> 3-> NULL
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 5
Exiting the program
```