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

```
#include <stdio.h>

#include <stdlib.h>


struct node{

   int data;

   struct node *next;

};

struct node* head=NULL;

struct node* head2=NULL;


void insert1(int val){

   struct node*  newnode=(struct node*)malloc(sizeof(struct node));

   newnode->data=val;

   newnode->next=head;

   head=newnode;

}


void insert2(int val){

   struct node*  newnode=(struct node*)malloc(sizeof(struct node));

   newnode->data=val;

   newnode->next=head2;

   head2=newnode;

}
```

```
void sort(){
   struct node *current=head;
   struct node *index=NULL;
   int temp;

   if(head==NULL){
      printf("List is empty");
      return;
   }
   else{
      while(current!=NULL){
         index=current->next;
         while(index!=NULL){
            if(current->data > index->data){
               temp=current->data;
               current->data=index->data;
               index->data=temp;
            }
            index=index->next;
         }
         current=current->next;
      }
   }
}
```

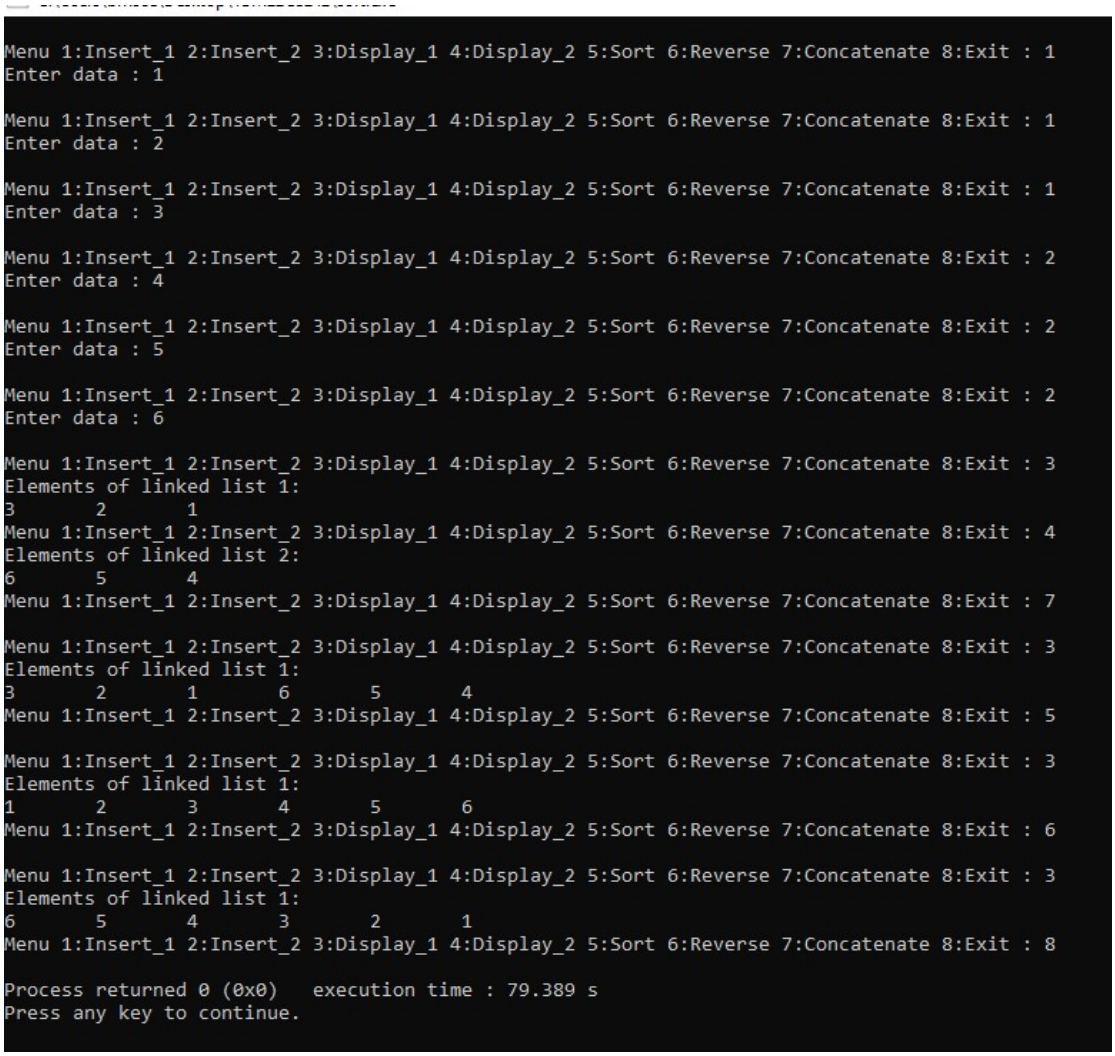
```
void reverse(){
   struct node* temp=head;
   struct node* prev=NULL;
   while(temp!=NULL){
      struct node* front=temp->next;
      temp->next=prev;
      prev=temp;
      temp=front;
   }
   head=prev;
}

void concat(){
   struct node* temp=head;
   while(temp->next!=NULL){
      temp=temp->next;
   }
   temp->next=head2;
}

void display1(){
   struct node*temp=head;
   if(head==NULL){
      printf("List is empty");
      return;
   }
```

```
    while(temp!=NULL){
        printf("%d\t",temp->data);
        temp=temp->next;
    }
}

void display2(){
    struct node*temp=head2;
    if(head2==NULL){
        printf("List is empty");
        return;
    }
    while(temp!=NULL){
        printf("%d\t",temp->data);
        temp=temp->next;
    }
}

int main(){
    int ch,val;
    while(ch!=8){
        printf("\nMenu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse
7:Concatenate 8:Exit : ");
        scanf("%d",&ch);
        switch(ch){
            case 1:
```

```
    printf("Enter data : ");
    scanf("%d",&val);
    insert1(val);
    break;
case 2:
    printf("Enter data : ");
    scanf("%d",&val);
    insert2(val);
    break;
case 3:
    printf("Elements of linked list 1:\n");
    display1();
    break;
case 4:
    printf("Elements of linked list 2:\n");
    display2();
    break;
case 5:
    sort();
    break;
case 6:
    reverse();
    break;
case 7:
    concat();
    break;
```

```
        case 8:
            return 0;
      }
   }
   return 0;
}
```

Output:



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

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *prev;
   struct node *next;
};

struct node *head = NULL;

struct node *insert_begin(struct node *start)
{
   struct node *temp;
   temp = (struct node *)malloc(sizeof(struct node));
   printf("Enter the value to be inserted\n");
   scanf("%d", &temp->data);
   temp->next = NULL;
   temp->prev = NULL;
   if (start == NULL)
   {
      start = temp;
   }

   else
   {
```

```
        temp->next = start;
        start->prev = temp;
        start = temp;
    }
    return start;
}

struct node *insert_end(struct node *start)
{
    struct node *temp;
    temp = (struct node *)malloc(sizeof(struct node));
    printf("Enter the value to be inserted\n");
    scanf("%d", &temp->data);
    temp->next = NULL;
    temp->prev = NULL;

    if (start == NULL)
    {
        start = temp;
    }

    else
    {
        struct node *ptr;
        ptr = start;
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
```

```
        }
        ptr->next = temp;
        temp->prev = ptr;
    }
    return start;
}

struct node *insert_pos(struct node *start)
{
    int n;
    struct node *temp;
    struct node *ptr = start;
    temp = (struct node *)malloc(sizeof(struct node));
    printf("Enter the value to be inserted\n");
    scanf("%d", &temp->data);
    temp->next = NULL;
    temp->prev = NULL;
    printf("Enter the position where the node has to be inserted\n");
    scanf("%d", &n);

    if (n == 1)
    {
        temp->next = start;
        start = temp;
    }
    else
    {
        for (int i = 1; i < n - 1; i++)
```

```
        {
            ptr = ptr->next;
            if (ptr == NULL)
            {
                printf("the node cant be inserted at position -%d\n", n);
            }
        }

        temp->next = ptr->next;
        ptr->next = temp;
    }
    return start;
}

struct node *delete_begin(struct node *start)
{
    if (start == NULL)
    {
        printf("Empty list\n");
        return start;
    }
    else if (start->next == NULL)
    {
        printf("Value deleted=%d\n", start->data);
        free(start);
        start = NULL;
    }
    else
```

```
    {
        struct node *ptr;
        ptr = start;
        start = start->next;
        printf("Value deleted=%d\n", ptr->data);
        free(ptr);
        start->prev = NULL;
    }

    return start;
}

struct node *delete_end(struct node *start)
{
    struct node *ptr = start;
    if (start == NULL)
    {
        printf("\n the list is empty\n");
        return start;
    }
    else if (start->next == NULL)
    {
        printf("The value deleted=%d\n", start->data);
        free(start);
        start = NULL;
    }
    else
    {
```

```
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        printf("the value deleted=%d\n", ptr->data);
        ptr->prev->next = NULL;
        free(ptr);
    }
    return start;
}

struct node *delete_pos(struct node *start)
{
    int n;
    struct node *ptr = start;
    if (start == NULL)
    {
        printf("\n The list is empty\n");
        return start;
    }
    printf("Enter the position to be deleted\n");
    scanf("%d", &n);
    if (n == 1)
    {
        printf("The value deleted=%d\n", start->data);
        start = start->next;
        start->prev = NULL;
        free(ptr);
```

```
    }
    else
    {
        for (int i = 1; i < n - 1; i++)
        {
            ptr = ptr->next;
        }
        printf("The value deleted=%d\n", ptr->next->data);
        ptr->next = ptr->next->next;

        free(ptr->next->prev);
        ptr->next->prev = ptr;
    }
    return start;
}

void display(struct node *start)
{
    struct node *ptr;
    ptr = start;
    if (start == NULL)
    {
        printf("\n list is empty\n");
    }
    else
    {   printf("Elements : ");
        while (ptr != NULL)
        {
```

```
            printf("%d\n", ptr->data);
            ptr = ptr->next;
        }
    }
}

int main()
{
    int ch;
    while (ch!=8)
    {
        printf("Menu: 1:Insert at beginning 2:Insert at end 3:Insert at given position 4:Delete from beginning \n 5:Delete from end 6:Delete at a specific position 7:display 8:Exit \n");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            head = insert_begin(head);
            break;
        case 2:
            head = insert_end(head);
            break;
        case 3:
            head = insert_pos(head);
            break;
        case 4:
             head= delete_begin(head);
            break;
        case 5:
```

```
            head= delete_end(head);
            break;
        case 6:
            head = delete_pos(head);
            break;
        case 7:
            display(head);
            break;
        case 8:
            return 0;
        }
    }
}
```

Output :

```
Menu: 1:Insert at beginning 2:Insert at end 3:Insert at given position 4:Delete from beginning
 5:Delete from end 6:Delete at a specific position 7:display 8:Exit
1
Enter the value to be inserted
1
Menu: 1:Insert at beginning 2:Insert at end 3:Insert at given position 4:Delete from beginning
 5:Delete from end 6:Delete at a specific position 7:display 8:Exit
2
Enter the value to be inserted
2
Menu: 1:Insert at beginning 2:Insert at end 3:Insert at given position 4:Delete from beginning
 5:Delete from end 6:Delete at a specific position 7:display 8:Exit
3
Enter the value to be inserted
3
Enter the position where the node has to be inserted
3
Menu: 1:Insert at beginning 2:Insert at end 3:Insert at given position 4:Delete from beginning
 5:Delete from end 6:Delete at a specific position 7:display 8:Exit
2
Enter the value to be inserted
4
Menu: 1:Insert at beginning 2:Insert at end 3:Insert at given position 4:Delete from beginning
 5:Delete from end 6:Delete at a specific position 7:display 8:Exit
7
Elements : 1
2
3
4
Menu: 1:Insert at beginning 2:Insert at end 3:Insert at given position 4:Delete from beginning
 5:Delete from end 6:Delete at a specific position 7:display 8:Exit
4
Value deleted=1
Menu: 1:Insert at beginning 2:Insert at end 3:Insert at given position 4:Delete from beginning
 5:Delete from end 6:Delete at a specific position 7:display 8:Exit
6
Enter the position to be deleted
2
The value deleted=3
Menu: 1:Insert at beginning 2:Insert at end 3:Insert at given position 4:Delete from beginning
 5:Delete from end 6:Delete at a specific position 7:display 8:Exit
7
Elements : 2
4
Menu: 1:Insert at beginning 2:Insert at end 3:Insert at given position 4:Delete from beginning
 5:Delete from end 6:Delete at a specific position 7:display 8:Exit
8

Process returned 0 (0x0)   execution time : 47.780 s
Press any key to continue.
```