(1) SINGLE LINKED LIST (SORT, REVERSE AND CONCATENATION)

CODE:

```
#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:

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 1

Enter data : 1

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 1

Enter data : 2

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 2

Enter data : 3

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 2

Enter data : 4

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 3

Elements of linked list 1:

2 1

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 4

Elements of linked list 2:

4 3

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 5

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 3

Elements of linked list 1:

1 2

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 6

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 3

Elements of linked list 1:

2 1

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 7

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 3

Elements of linked list 1:

2 1 4 3

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 8

(2) DOUBLY LINKED LIST

CODE:

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

struct node *s1 = 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", start->data);
        free(start);
        start = NULL;
    }
    else
    {
        struct node *ptr;
        ptr = start;
        start = start->next;
        printf("Value deleted=%d", 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", start->data);
        free(start);
        start = NULL;
    }
    else
    {
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        printf("the value deleted=%d", 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", 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", 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
    {
        while (ptr != NULL)
        {
            printf("%d\n", ptr->data);
            ptr = ptr->next;
        }
    }
}

int main()
{
    int choice;
    while (1)
```

```
{
    printf("\n 1. to add in the beginning \n 2. to add at the end\n 3. to insert at the given position\n 4.to delete at the beginning\n 5.to delete at the end\n 6.to delete at a specific position\n 7.display \n 8.Exit\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
       s1 = insert_begin(s1);
       break;
    case 2:
       s1 = insert_end(s1);
       break;
    case 3:
       s1 = insert_pos(s1);
       break;
    case 4:
       s1 = delete_begin(s1);
       break;
    case 5:
       s1 = delete_end(s1);
       break;
    case 6:
       s1 = delete_pos(s1);
       break;
    case 7:
       display(s1);
```

```
            break;


        case 8:
            exit(0);
        }
    }
}
```

OUTPUT:

```
 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
 6.to delete at a specific position
 7.display
 8.Exit
1
Enter the value to be inserted
10


 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
```

6.to delete at a specific position

7.display

8.Exit

2

Enter the value to be inserted

20

1. to add in the beginning

2. to add at the end

3. to insert at the given position

4.to delete at the beginning

5.to delete at the end

6.to delete at a specific position

7.display

8.Exit

2

Enter the value to be inserted

30

1. to add in the beginning

2. to add at the end

3. to insert at the given position

4.to delete at the beginning

5.to delete at the end

6.to delete at a specific position

7.display

8.Exit

7

10

20

30

1. to add in the beginning

2. to add at the end

3. to insert at the given position

4.to delete at the beginning

5.to delete at the end

6.to delete at a specific position

7.display

8.Exit

5

the value deleted=30

1. to add in the beginning

2. to add at the end

3. to insert at the given position

4.to delete at the beginning

5.to delete at the end

6.to delete at a specific position

7.display

8.Exit

7

10

20

1. to add in the beginning

2. to add at the end

3. to insert at the given position

4.to delete at the beginning

5.to delete at the end

6.to delete at a specific position

7.display

8.Exit

3

Enter the value to be inserted

40

Enter the position where the node has to be inserted

2

1. to add in the beginning

2. to add at the end

3. to insert at the given position

4.to delete at the beginning

5.to delete at the end

6.to delete at a specific position

7.display

8.Exit

7

10

40

20

1. to add in the beginning

2. to add at the end

3. to insert at the given position

4.to delete at the beginning

5.to delete at the end

6.to delete at a specific position

7.display

8.Exit

4

Value deleted=10

1. to add in the beginning

2. to add at the end

3. to insert at the given position

4.to delete at the beginning

5.to delete at the end

6.to delete at a specific position

7.display

8.Exit

7

40

20

1. to add in the beginning

2. to add at the end

3. to insert at the given position

4.to delete at the beginning

5.to delete at the end

6.to delete at a specific position

7.display

8.Exit

8