WEEK9:

Q1)Inserting at any position(Linked list)

CODE:

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

struct node
{
	int info;
	struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
	NODE x;
	x = (NODE)malloc(sizeof(struct node));
	if (x == NULL)
	{
		printf("Memory is full!!\n");
		exit(0);
	}
	return x;
}
void freenode(NODE x)
{
	free(x);
}
NODE insert_front(NODE first, int item)
{
	NODE temp;
	temp = getnode();
	temp->info = item;
	temp->link = NULL;
	if (first == NULL)
		return temp;
	temp->link = first;
	first = temp;
	return first;
}
```

```
NODE delete_front(NODE first)
{
	NODE temp;
	if (first == NULL)
	{
		printf("List is empty cannot delete!\n");
		return first;
	}
	temp = first;
	temp = temp->link;
	printf("The item deleted from front of the list is : %d\n", first->info);
	free(first);
	return temp;
}
NODE insert_rear(NODE first, int item)
{
	NODE temp, cur;
	temp = getnode();
	temp->info = item;
	temp->link = NULL;
	if (first == NULL)
		return temp;
	cur = first;
	while (cur->link != NULL)
		cur = cur->link;
	cur->link = temp;
	return first;
}
NODE insert_pos(int item, int pos, NODE first)
{
	NODE temp, cur, prev;
	int count;
	temp = getnode();
	temp ->info = item;
	temp ->link = NULL;
	if (first == NULL && pos == 1)
	{
		return temp;
	}
	if (first == NULL)
	{
		printf("Invalid position\n");
		return first;
```

```
        }
        if (pos == 1)
        {
                temp ->link = first;
                first = temp;
                return temp;
        }
        count = 1;
        prev = NULL;
        cur = first;
        while (cur != NULL && count != pos)
        {
                prev = cur;
                cur = cur ->link;
                count++;
        }
        if (count == pos)
        {
                prev ->link = temp;
                temp ->link = cur;
                return first;
        }
        printf("Invalid position\n ");
        return first;
}
NODE delete_rear(NODE first)
{
        NODE cur, prev;
        if (first == NULL)
        {
                printf("List is empty cannot delete\n");
                return first;
        }
        if (first->link == NULL)
        {
                printf("Item deleted is %d\n", first->info);
                free(first);
                return NULL;
        }
        prev = NULL;
        cur = first;
        while (cur->link != NULL)
        {
```

```
            prev = cur;
            cur = cur->link;
        }
        printf("Item deleted at rear-end is %d\n", cur->info);
        free(cur);
        prev->link = NULL;
        return first;
}
void display(NODE first)
{
        NODE temp;
        if (first == NULL)
            printf("List is EMPTY!\n");
        for (temp = first; temp != NULL; temp = temp->link)
        {
            printf("%d\n", temp->info);
        }
}
void main()
{
        int item, choice, pos;
        NODE first = NULL;

        for (;;)
        {

printf("\n---------------------------\n1:Insert_front\n2:Delete_front\n3:Insert_rear\n4:Insert_pos\n5:Delete_r
ear\n6:Display_list\n7:Exit\n");
            printf("Enter the choice\n");
            scanf("%d", &choice);
            switch (choice)
            {
            case 1:
                printf("Enter the item at front-end\n");
                scanf("%d", &item);
                first = insert_front(first, item);
                break;
            case 2:
                first = delete_front(first);
                break;
            case 3:
                printf("Enter the item at rear-end\n");
                scanf("%d", &item);
```

```
                first = insert_rear(first, item);
                break;
            case 4:
                printf("Enter the item to be inserted at given position\n");
                scanf("%d",&item);
                printf("Enter the position\n");
                scanf("%d",&pos);
                first=insert_pos(item,pos,first);
                break;
            case 5:
                first = delete_rear(first);
                break;
            case 6:
                printf("The list is:\n");
                display(first);
                break;
            case 7:
                exit(0);
                break;
            default:
                printf("INVALID CHOICE!\n");
                break;
            }
        }
}
```

OUTPUT:

```
The list is:
10
20
30

-------------------------
1:Insert_front
2:Delete_front
3:Insert_rear
4:Insert_pos
5:Delete_rear
6:Display_list
7:Exit
Enter the choice
4
Enter the item to be inserted at given position
100
Enter the position
2

-------------------------
1:Insert_front
2:Delete_front
3:Insert_rear
4:Insert_pos
5:Delete_rear
6:Display_list
7:Exit
Enter the choice
6
```

```
Enter the choice
6
The list is:
10
100
20
30

-------------------------
1:Insert_front
2:Delete_front
3:Insert_rear
4:Insert_pos
5:Delete_rear
6:Display_list
7:Exit
Enter the choice
```

Q2)Delete from any position(linked list)

CODE:

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

struct node
{
	int info;
	struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
	NODE x;
	x = (NODE)malloc(sizeof(struct node));
	if (x == NULL)
	{
		printf("Memory is full!!\n");
		exit(0);
	}
	return x;
}
void freenode(NODE x)
{
	free(x);
}
NODE insert_front(NODE first, int item)
{
	NODE temp;
	temp = getnode();
	temp->info = item;
	temp->link = NULL;
	if (first == NULL)
		return temp;
	temp->link = first;
	first = temp;
	return first;
}
NODE delete_front(NODE first)
{
	NODE temp;
	if (first == NULL)
	{
		printf("List is empty cannot delete!\n");
		return first;
	}
	temp = first;
	temp = temp->link;
	printf("The item deleted from front of the list is : %d\n", first->info);
```

```
        free(first);
        return temp;
}
NODE insert_rear(NODE first, int item)
{
        NODE temp, cur;
        temp = getnode();
        temp->info = item;
        temp->link = NULL;
        if (first == NULL)
                return temp;
        cur = first;
        while (cur->link != NULL)
                cur = cur->link;
        cur->link = temp;
        return first;
}
NODE insert_pos(int item, int pos, NODE first)
{
        NODE temp, cur, prev;
        int count;
        temp = getnode();
        temp ->info = item;
        temp ->link = NULL;
        if (first == NULL && pos == 1)
        {
                return temp;
        }
        if (first == NULL)
        {
                printf("Invalid position\n");
                return first;
        }
        if (pos == 1)
        {
                temp ->link = first;
                first = temp;
                return temp;
        }
        count = 1;
        prev = NULL;
        cur = first;
        while (cur != NULL && count != pos)
        {
                prev = cur;
                cur = cur ->link;
                count++;
        }
        if (count == pos)
```

```
        {
                prev ->link = temp;
                temp ->link = cur;
                return first;
        }
        printf("Invalid position\n ");
        return first;
}
NODE delete_rear(NODE first)
{
        NODE cur, prev;
        if (first == NULL)
        {
                printf("List is empty cannot delete\n");
                return first;
        }
        if (first->link == NULL)
        {
                printf("Item deleted is %d\n", first->info);
                free(first);
                return NULL;
        }
        prev = NULL;
        cur = first;
        while (cur->link != NULL)
        {
                prev = cur;
                cur = cur->link;
        }
        printf("Item deleted at rear-end is %d\n", cur->info);
        free(cur);
        prev->link = NULL;
        return first;
}
NODE delete_pos(int pos,NODE first)
{
NODE cur;
NODE prev;
int count,flag=0;
if(first==NULL || pos<0)
{
printf("Invalid position\n");
return NULL;
}
if(pos==1)
{
cur=first;
first=first->link;
freenode(cur);
```

```
return first;
}
prev=NULL;
cur=first;
count=1;
while(cur!=NULL)
{
if(count==pos){flag=1;break;}
count++;
prev=cur;
cur=cur->link;
}
if(flag==0)
{
printf("Invalid position\n");
return first;
}
printf("Item deleted at given position is %d\n",cur->info);
prev->link=cur->link;
freenode(cur);
return first;
}
void display(NODE first)
{
	NODE temp;
	if (first == NULL)
		printf("List is EMPTY!\n");
	for (temp = first; temp != NULL; temp = temp->link)
	{
		printf("%d\n", temp->info);
	}
}
void main()
{
	int item, choice, pos;
	NODE first = NULL;

	for (;;)
	{

printf("\n---------------------------\n1:Insert_front\n2:Delete_front\n3:Insert_rear\n4:Insert_pos\n5:Delete_r
ear\n6:Delete_pos\n7:Display_list\n8:Exit\n");
		printf("Enter the choice\n");
		scanf("%d", &choice);
		switch (choice)
		{
		case 1:
			printf("Enter the item at front-end\n");
			scanf("%d", &item);
```

```
                first = insert_front(first, item);
                break;
            case 2:
                first = delete_front(first);
                break;
            case 3:
                printf("Enter the item at rear-end\n");
                scanf("%d", &item);
                first = insert_rear(first, item);
                break;
            case 4:
                printf("Enter the item to be inserted at given position\n");
                scanf("%d",&item);
                printf("Enter the position\n");
                scanf("%d",&pos);
                first=insert_pos(item,pos,first);
                break;
            case 5:
                first = delete_rear(first);
                break;
            case 6:
                printf("Enter the position\n");
                scanf("%d",&pos);
                first=delete_pos(pos,first);
                break;
            case 7:
                printf("The list is:\n");
                display(first);
                break;
            case 8:
                exit(0);
                break;
            default:
                printf("INVALID CHOICE!\n");
                break;
            }
        }
}
```

OUTPUT:

```
1:Insert_front
2:Delete_front
3:Insert_rear
4:Insert_pos
5:Delete_rear
6:Delete_pos
7:Display_list
8:Exit
Enter the choice
7
The list is:
10
20
30
40

--------------------------
1:Insert_front
2:Delete_front
3:Insert_rear
4:Insert_pos
5:Delete_rear
6:Delete_pos
7:Display_list
8:Exit
Enter the choice
6
Enter the position
2
```

```
Enter the position
2
Item deleted at given position is 20

--------------------------
1:Insert_front
2:Delete_front
3:Insert_rear
4:Insert_pos
5:Delete_rear
6:Delete_pos
7:Display_list
8:Exit
Enter the choice
7
The list is:
10
30
40

--------------------------
```

3A)Sorting linked list:

CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
	int info;
	struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
	NODE x;
	x = (NODE)malloc(sizeof(struct node));
	if (x == NULL)
	{
		printf("MEMORY FULL!!!!\n");
		exit(0);
	}
	return x;
}

NODE order_list(int item,NODE first)
{
NODE temp,prev,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL) return temp;
if(item<first->info)
{
temp->link=first;
return temp;
}
prev=NULL;
cur=first;
while(cur!=NULL&&item>cur->info)
{
prev=cur;
cur=cur->link;
}
prev->link=temp;
temp->link=cur;
return first;
}
```

```
void display(NODE first)
{
	NODE temp;
	if (first == NULL)
		printf("List is EMPTY!!!\n");
	for (temp = first; temp != NULL; temp = temp->link)
	{
		printf("%d\n", temp->info);
	}
}

void main()
{
	int item, choice, pos, i, n;
	NODE first = NULL;
	for (;;)
	{
		printf("1: Insert_front\n2: Dislay sorted list\n3: Exit\n");
		printf("Enter choice:\n");
		scanf("%d", &choice);
		switch (choice)
		{
		case 1:
			printf("Enter the item\n");
			scanf("%d", &item);
			first = order_list(item,first);
			break;
		case 2:
			display(first);
			break;
		case 3:
			exit(0);
		default:
			printf("INVALID INPUT!!!\n");
		}
	}
}
```

OUTPUT:

```
C:\Users\misaf\Desktop\DS LAB\week9>llsortc
1: Insert_front
2: Dislay sorted list
3: Exit
Enter choice:
1
Enter the item
5
1: Insert_front
2: Dislay sorted list
3: Exit
Enter choice:
1
Enter the item
2
1: Insert_front
2: Dislay sorted list
3: Exit
Enter choice:
1
Enter the item
50
1: Insert_front
2: Dislay sorted list
3: Exit
Enter choice:
1
Enter the item
0
```

```
Enter the item
0
1: Insert_front
2: Dislay sorted list
3: Exit
Enter choice:
2
0
2
5
50
1: Insert_front
2: Dislay sorted list
```

3B)Reversing linked list:

CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
	int info;
	struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
	NODE x;
	x = (NODE)malloc(sizeof(struct node));
	if (x == NULL)
	{
		printf("MEMORY FULL!!!!\n");
		exit(0);
	}
	return x;
}
NODE insert_rear(NODE first, int item)
{
	NODE temp, cur;
	temp = getnode();
	temp->info = item;
	temp->link = NULL;
	if (first == NULL)
		return temp;
	cur = first;
	while (cur->link != NULL)
		cur = cur->link;
	cur->link = temp;
	return first;
}
void display(NODE first)
{
	NODE temp;
	if (first == NULL)
		printf("List is EMPTY!!!\n");
	for (temp = first; temp != NULL; temp = temp->link)
	{
		printf("%d\n", temp->info);
	}
}

NODE reverse(NODE first)
```

```
{
	NODE cur, temp;
	cur = NULL;
	while (first != NULL)
	{
		temp = first;
		first = first->link;
		temp->link = cur;
		cur = temp;
	}
	return cur;
}
void main()
{
	int item, choice, pos, i, n;
	NODE first = NULL, a, b;
	for (;;)
	{
		printf("1: Insert_front\n2: Reverse\n3: Dislay\n4: Exit\n");
		printf("Enter choice:\n");
		scanf("%d", &choice);
		switch (choice)
		{
		case 1:
			printf("Enter the item\n");
			scanf("%d", &item);
			first = insert_rear(first, item);
			break;

		case 2:
			first = reverse(first);
			printf("REVERSED!!\n");
			break;
		case 3:
			display(first);
			break;
		default:
			exit(0);
		}
	}
}
```

OUTPUT:

```
1: Insert_front
2: Reverse
3: Dislay
4: Exit
Enter choice:
3
10
20
30
1: Insert_front
2: Reverse
3: Dislay
4: Exit
Enter choice:
2
REVERSED!!
1: Insert_front
2: Reverse
3: Dislay
4: Exit
Enter choice:
3
30
20
10
1: Insert_front
2: Reverse
3: Dislay
4: Exit
Enter choice:
```

3C)Concatenate

CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
	int info;
	struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
	NODE x;
```

```
        x = (NODE)malloc(sizeof(struct node));
        if (x == NULL)
        {
                printf("MEMORY FULL!!!!\n");
                exit(0);
        }
        return x;
}
NODE insert_rear(NODE first, int item)
{
        NODE temp, cur;
        temp = getnode();
        temp->info = item;
        temp->link = NULL;
        if (first == NULL)
                return temp;
        cur = first;
        while (cur->link != NULL)
                cur = cur->link;
        cur->link = temp;
        return first;
}
void display(NODE first)
{
        NODE temp;
        if (first == NULL)
                printf("List is EMPTY!!!\n");
        for (temp = first; temp != NULL; temp = temp->link)
        {
                printf("%d\n", temp->info);
        }
}

NODE concat(NODE first, NODE second)
{
        NODE cur;
        if (first == NULL)
                return second;
        if (second == NULL)
                return first;
        cur = first;
        while (cur->link != NULL)
                cur = cur->link;
        cur->link = second;
        return first;
}
void main()
{
        int item, choice, pos, i, n;
```

```
    NODE firsta = NULL, firstb=NULL;
    for (;;)
    {
        printf("\n1:INSERT_FRONT LIST1\n2:INSERT_FRONT LIST2\n3:DISPLAY LIST1\n4:DISPLAY
LIST2\n5:CONCATENATE AND DISPLAY\n6:EXIT\n");
        printf("Enter choice:\n");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
            printf("Enter the item\n");
            scanf("%d", &item);
            firsta = insert_rear(firsta, item);
            break;
            case 2:
            printf("Enter the item\n");
            scanf("%d", &item);
            firstb = insert_rear(firstb, item);
            break;
            case 3:
            display(firsta);
            break;
            case 4:
            display(firstb);
            break;
            case 5:
            firsta=concat(firsta,firstb);
            display(firsta);
            break;
            case 6:
            exit(0);
            default:printf("INVALID INPUT!!\n");


        }
    }
}
```

OUTPUT:

```
1:INSERT_FRONT LIST1
2:INSERT_FRONT LIST2
3:DISPLAY LIST1
4:DISPLAY LIST2
5:CONCATENATE AND DISPLAY
6:EXIT
Enter choice:
3
10
20

1:INSERT_FRONT LIST1
2:INSERT_FRONT LIST2
3:DISPLAY LIST1
4:DISPLAY LIST2
5:CONCATENATE AND DISPLAY
6:EXIT
Enter choice:
4
30
40
50
```

```
1:INSERT_FRONT LIST1
2:INSERT_FRONT LIST2
3:DISPLAY LIST1
4:DISPLAY LIST2
5:CONCATENATE AND DISPLAY
6:EXIT
Enter choice:
5
10
20
30
40
50
```