Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node* next;
}*new,*prev,*temp,*start,*end;
int isempty(){
        return start==NULL;
}
void createNode(int data){
        new=(struct node*)malloc(sizeof(struct node));
        new->data=data;
        new->next=NULL;
}
void insertBegin(int data){
        createNode(data);
        if(isempty())
                end=start=new;
        else{
                new->next=start;
                start=new;
        }
        printf("\nInserted Successfully");
}
void insertEnd(int data){
        createNode(data);
        if(isempty())
```

```
                end=start=new;
        else{
                end->next=new;
                end=new;
        }
        printf("\nInserted Successfully");
}
void insertPos(int data,int pos){
        int i=1;
        createNode(data);
        if(isempty()){
                end=start=new;
        }
        else{
                temp=start;
                while(temp->next!=NULL&&i<pos-1){
                        temp=temp->next;
                        i++;
                }
                if(temp->next==NULL&&i<pos-1){
                        end->next=new;
                        end=new;
                }
                else if(pos==1){
                        new->next=start;
                        start=new;
                }
                else{
                        new->next=temp->next;
                        temp->next=new;
        }
    }
    printf("\nInserted Successfully");
}
void delete(int data){
    if(isempty())
        printf("\nList is Empty");
    else{
        temp=start;
        while(temp->next!=NULL&&temp->data!=data){
            prev=temp;
            temp=temp->next;
        }
        if(temp->next==NULL&&temp->data!=data){
            printf("\nData Not Found");
            return;
        }
        else if(start->data==data)
            start=start->next;
        else if(end->data==data){
            prev->next=NULL;
            end=prev;
        }
        else
            prev->next=temp->next;
        printf("\n%d is deleted successfully",temp->data);
    }

}
void search(int data){
	int choice,i=1;
	if(isempty())
		printf("\nList is empty");
	else{
		temp=start;
		while(temp->next!=NULL && temp->data!=data){
			temp=temp->next;
			i++;
		}
		if(temp->next==NULL && temp->data!=data)
			printf("\n%d is not found",data);
		else
			printf("\n%d is found in %d position",temp->data,i);
	}

}
void display(){
	temp=start;
	if(isempty())
		printf("\nList is empty");
	else{
		while(temp->next!=NULL){
			printf("\nAddress:%u\tData:%d\tNext:%u",temp,temp->data,temp->next);
			temp=temp->next;
		}
		printf("\nAddress:%u\tData:%d\tNext:%u",temp,temp->data,temp->next);
	}
}
void main(){
	int option,c,element,position;
	do{
		printf("\nSingle Linked List\n1.Insert Begin\n2.Insert End\n3.Insert in Position\n4.Delete\n5.Search\n6.Display\nEnter your option:");
		scanf("%d",&option);
		switch(option){
			case 1:
				printf("Enter Element:");
				scanf("%d",&element);
				insertBegin(element);
				break;
			case 2:
				printf("Enter Element:");
				scanf("%d",&element);
				insertEnd(element);
				break;
			case 3:
				printf("Enter Element:");
				scanf("%d",&element);
				printf("Enter Position:");
				scanf("%d",&position);
				insertPos(element,position);
				break;
			case 4:
				printf("Enter Element:");
				scanf("%d",&element);
				delete(element);
				break;
			case 5:
				printf("Enter Element:");
				scanf("%d",&element);
```

```
                        search(element);
                        break;
                case 6:
                        display();
                        break;
                default:
                        printf("\nInvalid Case");
                        break;
            }
            printf("\nPress 1 to continue: ");
            scanf("%d",&c);
        }while(c==1);
}
```

Circular Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node* next;
}*new,*prev,*temp,*start,*end;
int isempty(){
        return start==NULL;
}
void createNode(int data){
        new=(struct node*)malloc(sizeof(struct node));
        new->data=data;
        new->next=new;
}
void insertBegin(int data){
        createNode(data);
        if(isempty())
                end=start=new;
        else{
                new->next=start;
                end->next=new;
                start=new;
        }
        printf("\nInserted Successfully");
}
void insertEnd(int data){
        createNode(data);
```

```
    if(isempty())
        end=start=new;
    else{
        end->next=new;
        new->next=start;
        end=new;
    }
    printf("\nInserted Successfully");
}
void insertPos(int data,int pos){
    int i=1;
    createNode(data);
    if(isempty()){
        end=start=new;
    }
    else{
        temp=start;
        while(temp->next!=start&&i<pos-1){
            temp=temp->next;
            i++;
        }
        if(temp->next==start&&i<pos-1){
            end->next=new;
            new->next=start;
            end=new;
        }
        else if(pos==1){
            new->next=start;
            end->next=new;
            start=new;
        }
        else{
            new->next=temp->next;
            temp->next=new;
        }
    }
    printf("\nInserted Successfully");
}
void delete(int data){
    if(isempty())
        printf("\nList is Empty");
    else{
        temp=start;
        while(temp->next!=start&&temp->data!=data){
            prev=temp;
            temp=temp->next;
        }
        if(temp->next==start&&temp->data!=data){
            printf("\nData Not Found");
            return;
        }
        else if(start->data==data){
            start=start->next;
            end->next=start;
        }
        else if(end->data==data){
            prev->next=start;
            end=prev;
        }
        else
```

```
                prev->next=temp->next;
        }
        printf("\n%d is deleted successfully",temp->data);
}
void search(int data){
        int choice,i=1;
        if(isempty())
                printf("\nList is empty");
        else{
                temp=start;
                while(temp->next!=start && temp->data!=data){
                        temp=temp->next;
                        i++;
                }
                if(temp->next==start && temp->data!=data)
                        printf("\n%d is not found",data);
                else
                        printf("\n%d is found in %d position",temp->data,i);
        }
}
void display(){
        temp=start;
        if(isempty())
                printf("\nList is empty");
        else{
                while(temp->next!=start){
                        printf("\nAddress:%u\tData:%d\tNext:%u",temp,temp->data,temp->next);
                        temp=temp->next;
                }
                printf("\nAddress:%u\tData:%d\tNext:%u",temp,temp->data,temp->next);
        }
}
void main(){
        int option,c,element,position;
        printf("\nCircular Linked List Implementation");
        do{
                printf("\n1.Insert Begin\n2.Insert End\n3.Insert in
Position\n4.Delete\n5.Search\n6.Display\nEnter your option:");
                scanf("%d",&option);
                switch(option){
                        case 1:
                                printf("Enter Element:");
                                scanf("%d",&element);
                                insertBegin(element);
                                break;
                        case 2:
                                printf("Enter Element:");
                                scanf("%d",&element);
                                insertEnd(element);
                                break;
                        case 3:
                                printf("Enter Element:");
                                scanf("%d",&element);
                                printf("Enter Position:");
                                scanf("%d",&position);
                                insertPos(element,position);
                                break;
                        case 4:
                                printf("Enter Element:");
                                scanf("%d",&element);
                                delete(element);
                                break;
                        case 5:
                                printf("Enter Element:");
                                scanf("%d",&element);
                                search(element);
                                break;
                        case 6:
                                display();
                                break;
                        default:
                                printf("\nInvalid Case");
                                break;
                }
                printf("\nPress 1 to continue: ");
                scanf("%d",&c);
        }while(c==1);
}
```

Doubly Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct node{
	int data;
	struct node *left, *right;
}*new,*prev,*temp,*start,*end;
int isempty(){
	return start==NULL;
}
void createNode(int data){
	new=(struct node*)malloc(sizeof(struct node));
	new->data=data;
	new->left=NULL;
	new->right=NULL;
}
void insertBegin(int data){
	createNode(data);
	if(isempty())
		end=start=new;
	else{
		new->right=start;
		start->left=new;
		start=new;
	}
	printf("\nInserted Successfully");
}
void insertEnd(int data){
        createNode(data);
        if(isempty())
                end=start=new;
        else{
                end->right=new;
                new->left=end;
                end=new;
        }
        printf("\nInserted Successfully");
}
void insertPos(int data,int pos){
        int i=1;
        createNode(data);
        if(isempty()){
                end=start=new;
        }
        else{
                temp=start;
                while(temp->right!=NULL&&i<pos-1){
                        temp=temp->right;
                        i++;
                }
                if(temp->right==NULL&&i<pos-1){
                        end->right=new;
                        new->left=end;
                        end=new;
                }
                else if(pos==1){
                        new->right=start;
                        start->left=new;
                start=new;
            }
            else{
                new->right=temp->right;
                temp->right->left=new;
                temp->right=new;
                new->left=temp;
            }
        }
        printf("\nInserted Successfully");
}
void delete(int data){
        if(isempty())
            printf("\nList is Empty");
        else{
            temp=start;
            while(temp->right!=NULL&&temp->data!=data){
                prev=temp;
                temp=temp->right;
            }
            if(temp->right==NULL&&temp->data!=data){
                printf("\nData Not Found");
                return;
            }
            else if(start->data==data){
                start=start->right;
                start->left=NULL;
            }
            else if(end->data==data){
                prev->right=NULL;
```

```
				end=prev;
			}
			else{
				prev->right=temp->right;
				temp->right->left=prev;
			}
		}
		printf("\n%d is deleted successfully",temp->data);
}
void search(int data){
		int choice,i=1,pos=data;
		if(isempty())
			printf("\nList is empty");
		else{
			temp=start;
			while(temp->right!=NULL && temp->data!=data){
				temp=temp->right;
				i++;
			}
			if(temp->right==NULL && temp->data!=data)
				printf("\n%d is not found",data);
			else
				printf("\n%d is found in %d position",temp->data,i);
			}
}
void display(){
		temp=start;
		if(isempty())
			printf("\nList is empty");
		else{
```

```
                printf("\n       ");
                while(temp->right!=NULL){
                        printf("Left:%u\t|Address:%u | Data:%d
|\tRight:%u\n",temp->left,temp,temp->data,temp->right);
                        temp=temp->right;
                }
                printf("Left:%u\t|Address:%u | Data:%d
|\tRight:%u\n",temp->left,temp,temp->data,temp->right);
        }
}
void main(){
        int option,c,element,position;
        printf("\nDoubly Linked List Implementation");
        do{
                printf("\n1.Insert Begin\n2.Insert End\n3.Insert in
Position\n4.Delete\n5.Search\n6.Display\nEnter your option:");
                scanf("%d",&option);
                switch(option){
                        case 1:
                                printf("Enter Element:");
                                scanf("%d",&element);
                                insertBegin(element);
                                break;
                        case 2:
                                printf("Enter Element:");
                                scanf("%d",&element);
                                insertEnd(element);
                                break;
                        case 3:
                                printf("Enter Element:");
                                scanf("%d",&element);
                                printf("Enter Position:");
                                scanf("%d",&position);
                                insertPos(element,position);
                                break;
                        case 4:
                                printf("Enter Element:");
                                scanf("%d",&element);
                                delete(element);
                                break;
                        case 5:
                                printf("Enter Element:");
                                scanf("%d",&element);
                                search(element);
                                break;
                        case 6:
                                display();
                                break;
                        default:
                                printf("\nInvalid Case");
                                break;
                }
                printf("\nPress 1 to continue: ");
                scanf("%d",&c);
        }while(c==1);
}
```

Circular Queue

```
#include<stdio.h>
#include<stdlib.h>
#define MinSize 3
#define EmptyQueue -1
struct QueueRecord{
        int* list;
        int front,rear,capacity;
};
typedef struct QueueRecord *queue;
queue createQueue(int size){
        queue q = (struct QueueRecord*)malloc(sizeof(queue));
        q->front=q->rear=EmptyQueue;
        q->capacity=size;
        q->list = (int*)malloc(sizeof(int)*q->capacity);
        if(q!=NULL&&q->list!=NULL)
                printf("\nQueue Created Successfully");
        else
                printf("\nError");
        return q;
}
void makeEmpty(queue q){
        q->front=q->rear=EmptyQueue;
}
queue destroyQueue(queue q){
        if(!isempty(q))
                makeEmpty(q);
```

```
		free(q->list);
		free(q);
		q=NULL;
		printf("\nQueue is destroyed successfully");
		return q;
}
void enqueue(int element,queue q){
	if(!isfull(q)){
		q->rear=(q->rear+1)%(q->capacity);
		q->list[q->rear]=element;
		if(q->front==EmptyQueue)
			q->front++;
		printf("\nfront=%d\nrear=%d",q->front,q->rear);
		printf("\n%d inserted successfully",q->list[q->rear]);
	}
	else{
		printf("\nQueue is full");
	}
}
void dequeue(queue q){
	if(!isempty(q)){
		if(q->rear==q->front)
			makeEmpty(q);
		else{
			printf("\n%d is deleted",q->list[q->front]);
			q->front=(q->front+1)%(q->capacity);
		}
	}
	else
		printf("\nQueue is Empty");
```

```
}
int front(queue q){
        if(isempty(q)){
                printf("\nQueue is empty");
                return 0;
        }
        else
                return q->list[q->front];
}
int rear(queue q){
        if(isempty(q)){
                printf("\nQueue is empty");
                return 0;
        }
        else
                return q->list[q->rear];
}
void displayQueue(queue q){
        int i,k;
        if(!isempty(q)){
                printf("\nQueue\n");
                i=q->front;
                do{
                        printf("%d\t",i);
                        printf("%d\n",q->list[i]);
                        i=(i+1)%(q->capacity);
                        if(q->front==q->rear)
                                break;

                }while((i-((q->rear+1)%q->capacity))!=0);
```

```
          }
          else
                    printf("\nQueue is empty");
}
int isempty(queue q){
          return q->front==EmptyQueue&&q->rear==EmptyQueue;
}
int isfull(queue q){
          return (q->rear==q->capacity-1&&q->front==0)||(q->front==q->rear+1);
}
void main(){
          int choice,c,size,element;
          queue q;
          printf("\nQueue Implementation");
          do{
                    printf("\n1.Create Queue\n2.Destroy Queue\n3.Insert element in queue\n4.Delete
element in queue\n5.Display Front\n6.Display Rear\n7.Display Queue\nEnter your choice: ");
                    scanf("%d",&choice);
                    switch(choice){
                              case 1:
                                        if(q==NULL){
                                                  printf("\nCreate Queue\nEnter Queue Size : ");
                                                  scanf("%d",&size);
                                                  if(size>MinSize)
                                                            q=createQueue(size);
                                                  else
                                                            printf("\nQueue Size is Too less");
                                        }
                                        else
                                                  printf("\nQueue is Already Created");
```

```
        break;
case 2:
        if(q!=NULL){
                q=destroyQueue(q);
        }
        else
                printf("\nQueue is Not Exist");
        break;
case 3:
        if(q!=NULL){
                printf("\nInsert element in queue\nEnter the element : ");
                scanf("%d",&element);
                enqueue(element,q);
        }
        else
                printf("\nQueue is Not Exist");
        break;
case 4:
        if(q!=NULL){
                printf("\nDelete Element in Queue");
                dequeue(q);
        }
        else
                printf("\nQueue is Not Exist");
        break;
case 5:
        if(q!=NULL){
                printf("\nDisplay Front element\nFront element is %d",front(q));
        }
        else
```

```
                    printf("\nQueue is Not Exist");
                break;
            case 6:
                if(q!=NULL){
                    printf("\nDisplay Rear element\nRear element is %d",rear(q));
                }
                else
                    printf("\nQueue is Not Exist");
                break;
            case 7:
                if(q!=NULL){
                    displayQueue(q);
                }
                else
                    printf("\nQueue is Not Exist");
                break;
            default:
                printf("\nInvalid Case");
                break;
        }
        printf("\nPress 1 to continue: ");
        scanf("%d",&c);
    }while(c==1);
}
```