Lab Assignment # 01 (File/Pointer/Structures)

Q.1. Code of a function which takes a triangle as input and returns its area.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
int main()
{
	int i;
	float area;
	struct vertex
	{
		float x;
		float y;
	};
	struct triangle
	{
		struct vertex vertices[3];
	};
	struct triangle t;
	printf("Enter vertices of the triangle\n");
	for(i=0;i<3;i++)
		scanf("%f%f",&t.vertices[i].x,&t.vertices[i].y);
```

```
        area=0.5*(t.vertices[0].x*(t.vertices[1].y-
        t.vertices[2].y)+t.vertices[1].x*(t.vertices[2].y-
        t.vertices[0].y)+t.vertices[2].x*(t.vertices[0].y-t.vertices[1].y));
        if(area<0)
                printf("Area of the triangle is %f sq units",-area);
        else
                printf("Area of the triangle is %f sq units",area);
}
```

Test Cases:

Input(n)	Output

Q.2. Given two-line segments, we need to check if these two line segments intersect or not.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
main()
{
	float m1,m2,p,q,c1,c2;
	struct Point
	{
		int x,y;
	};
	struct LineSeg
	{
		struct Point p1,p2;
	}l1,l2;
	printf("Enter end-points of line segment 1\n");
	scanf("%d%d%d%d",&l1.p1.x,&l1.p1.y,&l1.p2.x,&l1.p2.y);
	printf("Enter end-points of line segment 2\n");
	scanf("%d%d%d%d",&l2.p1.x,&l2.p1.y,&l2.p2.x,&l2.p2.y);
	m1= (l1.p2.y-l1.p1.y)/(l1.p2.x-l1.p1.x);
	m2= (l2.p2.y-l2.p1.y)/(l2.p2.x-l2.p1.x);
	if(m1==m2 || m1==-m2)
		printf("Line will not intersect\n");
	else
		printf("Line intersects\n");
	c1= (l1.p2.y - m1*l1.p2.x);
	c2= (l2.p2.y - m2*l2.p2.x);
	q = (c1-c2)/(m2-m1);
	p = m1*q + c1;
	printf("Point of intersection of line is %f and %f",q,p);
	}
```

Test Cases:

Input(n)	Output

Q.3. We need to identify whether two rational numbers are equal or not.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
struct rational
{
        int numerator;
        int denominator;
}num1,num2,l1,l2;
struct rational *inputrational, *outputrational;
void reduce(struct rational *inputrational,struct rational *outputrational)
{
        int i,tmp,p,q;
        p=inputrational->numerator;
        q=inputrational->denominator;
        if(p>q)
                for(i=1;i<=p/2;i++)
                        if(p%i==0 && q%i==0)
                                tmp=i;
        else
                for(i=1;i<=q/2;i++)
                        if(p%i==0 && q%i==0)
                                tmp=i;
        outputrational->numerator=p/tmp;
        outputrational->denominator=q/tmp;
        printf("Reduced form is %d/%d\n",p/tmp,q/tmp);
}
int equal(struct rational num1,struct rational num2)
{
        inputrational=&num1;
        outputrational=&l1;
        reduce(inputrational,outputrational);
        inputrational=&num2;
        outputrational=&l2;
        reduce(inputrational,outputrational);
        if(l1.numerator==l2.numerator && l1.denominator==l2.denominator)
                return 1;
```

```
        else
                return 0;
        }
main()
{
        int b;
        printf("Enter first rational number\n");
        scanf("%d%d",&num1.numerator,&num1.denominator);
        printf("Enter second rational number\n");
        scanf("%d%d",&num2.numerator,&num2.denominator);
        b=equal(num1,num2);
        if(b==1)
                printf("True\n");
        else
                printf("False\n");
}
```

Test Cases:

Input(n)	Output

Q.4. Queries from the Indore election data file:-
a. Enlist winners of the each election.
b. Find the percentage of male candidates as well as female candidates.
c. Calculate the total percentage of votes received by a party (including all constituency) in each year.
d. Write your own two more analysis on the election data ….

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<string.h>
struct election_data{
	char st_name[30];
	int year;
	int ac_no;
	char ac_name[20];
	char ac_type[20];
	char cand_name[100];
	char cand_sex;
	char partyname[30];
	char partyabbre[10];
	long int totalvotpoll;
	long int electors;
};
main()
{
	struct election_data data[280];
	FILE *fp;
	int i=0,j=0,reader,w=0,tmp;
	float male=0,female=0;
	unsigned long long total=0,sum=0;
	fp = fopen("Indore_Election_data_modified.txt","r");
	while(1)
	{
	reader =
fscanf(fp,"%[^\n\t]\t%d\t%d\t%s\t%s\t%[^\n\t]\t%c\t%[^\n\t]\t%s\t%ld\t%ld\n",data[i].st
_name,&data[i].year,&data[i].ac_no,data[i].ac_name,data[i].ac_type,data[i].cand_name,
&data[i].cand_sex,data[i].partyname,data[i].partyabbre,&data[i].totalvotpoll,&data[i].ele
ctors);
	i++;
	if(reader==-1)
```

```
                break;
        }
j=i;
i=0;
while(i<j-1)
{
        w=i;
        tmp=data[i].year;
        while(tmp==data[i+1].year)
        {
                if(data[i+1].totalvotpoll>data[i].totalvotpoll)
                {
                        i++;
                        w=i;
                }
                else
                        i++;
        }
        printf("Winner of election of year %d\tis %s\n",data[w].year,data[w].cand_name);
        printf("Percentage of vote given:
        %f%\n\n",100.00*data[w].totalvotpoll/data[w].electors);
        i++;
}
for(i=0;i<j;i++)
{
        if(data[i].cand_sex=='M')
                male++;
        else if(data[i].cand_sex=='F')
                female++;
}
printf("\n\n\n\n\nPercentage of male candidates is %f%\n",100*male/(male+female));
printf("Percentage of female candidates is %f%\n\n\n\n",100*female/(male+female));
i=0;
while(i<j-1)
{
        sum=0;
        total=0;
        tmp=data[i].year;
        while(tmp==data[i+1].year)
        {
                sum=sum+data[i].totalvotpoll;
                total=total+data[i].electors;
                i++;
        }
printf("Percentage of overall voters in year %d \tis %f%\n",data[i].year,100.0*sum/total);
i++;
```

```
}
}
```

Test Cases:

Q.5. An input file contains 10000 random positive integers one at each line.

(a) Find the minimum and maximum values that can be calculated by summing exactly 9999 of the 10000 integers.
(b) Challenge Part: What if you need to find maximum and minimum of 9990 numbers out of 10000 numbers?

Concept/Pseudocode:

Program:

```
#include<stdio.h>
main()
{
```

```
	FILE *fp;
	int d,i,n;
	unsigned long long a[10000],min=0,max=0,tmp;
	fp=fopen("IntegerList.txt","r");
	while(!feof(fp))
		for(i=0;i<10000;i++)
			fscanf(fp,"%lld",&a[i]);
	for(i=1;i<10000;i++)
	{
		d=i;
		while(d>0&&a[d-1]>a[d])
		{
			tmp=a[d];
			a[d]=a[d-1];
			a[d-1]=tmp;
			d--;
		}
	}
	for(i=0;i<=9998;i++)
		min = min+a[i];
	max = min+a[9999]-a[0];
	printf("Minimum and maximum of 9999 out of 10000 elements are %lld
	%lld\n\n",min,max);
	min=0;
	max=0;
	printf("Enter number n out of 10000 for which you want to find minimum and
	maximum\n");
	scanf("%d",&n);
	for(i=0;i<n;i++)
		min = min+a[i];
	for(i=9999;i>9999-n;i--)
		max= max+a[i];
	printf("Minimum and maximum of %d out of 10000 elements are %lld and
	%lld\n",n,min,max);
	fclose(fp);
}
```

Test Cases:

Input(n)	Output

Lab Assignment # 02 (Linked List)

Q.1. ADT of Singular Linked List (SLL) program.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<malloc.h>
struct Node
{
        int data;
```

```
        struct Node *link;
};
struct Node *start=NULL;
struct Node* createNode()
{
        struct Node *n;
        n=(struct Node*)malloc(sizeof(struct Node));
        return(n);
}
struct Node* insertNode(struct Node *start,int data)
{
        struct Node *temp,*t;
        temp=createNode();
        temp->data=data;
        temp->link=NULL;
        if(start==NULL)
                start=temp;
        else
        {
                t=start;
                while(t->link!=NULL)
                        t=t->link;
                t->link=temp;
        }
 return(start);
}
struct Node* insertnode_at(struct Node *start,int data,int pos)
{
        struct Node *t,*temp,*tem;
        int count=0,i=1;
        temp=createNode();
        temp->data=data;
        t=start;
        while(t!=NULL)
        {       t=t->link;
                count++;
        }
        t=start;
        if(pos>count)
                return NULL;
        while(i<pos-1)
        {
                t=t->link;
                i++;
        }
        if(pos==1)
```

```
        {
                temp->link=t;
                start=temp;
        }
        else{
                tem=t->link;
                t->link=temp;
                temp->link=tem;
        }
        return start;
}
void deleteNode()
{
        struct Node *remove;
        if(start==NULL)
                printf("List is Empty\n");
        else
        {
                remove=start;
                start=start->link;
                free(remove);
        }
}
void printList(struct Node *start)
{
        struct Node *t;
        t=start;
        if(start=NULL)
                printf("List is empty\n");
        else
        {
                while(t!=NULL)
                {
                printf("%d->",t->data);
                t=t->link;
                }
        }
}
```

Q.2. ADT of Doubly Linked List (DLL) program.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *next;
        struct node *prev;
};
typedef struct node node;
node *start=NULL;
node* createnode()
{
        node *t;
        t=(node*)malloc(sizeof(node));
        t->next=NULL;
        t->prev=NULL;
        return t;
}
node* insertNode(node *start,int data)
{
        node *temp,*t,*p;
        temp=createnode();
        temp->data=data;
        temp->prev=NULL;
        temp->next=NULL;
        if(start==NULL)
                {
                temp->prev=start;
                start=temp;
```

```
                }
        else
        {
                t=start;
                while(t->next!=NULL)
                        t=t->next;
                t->next=temp;
                temp->prev=t;
        }
return start;
}
node* deleteNode(node *start)
{
        struct node *remove,*temp;
        temp=start;
        remove=start;
        start=temp->next;
        temp->prev=NULL;
        free(remove);
        return start;
}
node* insertNode_at(node *start,int pos)
{
        node *t=start,*temp;
        int count=0,i=1;
        while(t!=NULL)
        {
                t=t->next;
                count++;
        }
        if(pos>count)
                return NULL;
        t=start;
        while(i<pos-1)
        {
                t=t->next;
                i++;
        }
        temp=createnode();
        scanf("%d",&temp->data);
        if(pos==1)
        {
                temp->next=start;
                start=temp;
                temp->prev=NULL;
        }
```

```
        else
        {
                temp->next=t->next;
                t->next->prev=temp;
                t->next=temp;
                temp->prev=t;
        }
return start;
}
void printList(node *start)
{
        node *t,*temp=start;
        if(start==NULL)
        printf("List is empty\n");
        else
        {
        t=start;
                while(t!=NULL)
                {
                        printf("%d->",t->data);
                        t=t->next;
                }
                while(temp->next!=NULL)
                {
                        temp=temp->next;
                }
                printf("\n");
                while(temp!=NULL)
                {
                        printf("%d->",temp->data);
                        temp=temp->prev;
                }
        }
}
```

Q.3. Write a function int isLinkedListPalindrome(Node *head) that returns 1 if the given list is palindrome, else 0.

Concept/Pseudocode:

Program:

```
int isLinkedListPalindrome(Node *head)
{
        Node *p,*q,*sec_head,*curr,*prev,*next;
        int count=0,count1=0;
        p=head;
        q=head;
        while(1)
        {
                p=p->link->link;
                if(p->link==NULL);
                {
                        sec_head=q->link->link;
                        break;
                }
                else if(p==NULL)
                {
                        sec_head=q->link;
                        break;
                }
                q=q->link;
        }
        q->link=NULL;
        p=head;
        curr=sec_head;
        prev=NULL;
        while(curr!=NULL)
        {
                next=curr->link;
                curr->link=prev;
                prev=curr;
```

```
            curr=next;
            count++;
        }
        q=prev;
        while(p!=NULL && q!=NULL)
        {
            if(p->data==q->data)
                count1++;
            p=p->link;
            q=q->link;
        }
        if(count==count1)
        return 1;
        else
        return 0;
}
```

Test Cases:

Input(n)	Output

Q.4. Function Node * mergeLinkedLists(Node * head1, Node * head2) that takes two sorted linked lists and merges them into one linked list, then returns the head of new linked list.

Concept/Pseudocode:

Program:

```
Node* mergeLinkedLists(Node * head1, Node * head2)
{
        Node *sorting,*new_head=NULL;
        if(head1==NULL)
                return head2;
        else if (head2==NULL)
                return head1;
        else if(head1!=NULL && head2!=NULL)
        {
                if(head1->data<=head2->data)
                {
                        sorting=head1;
                        head1=sorting->next;
                }
                else
                {
                        sorting=head2;
                        head2=sorting->next;
                }
                new_head=sorting;
                while(head1!=NULL && head2!=NULL)
                {
                        if(head1->data<=head2->data)
                        {
                                sorting->next=head1;
                                sorting=head1;
                                head1=sorting->next;
                        }
                        else
                        {
                                sorting->next=head2;
                                sorting=head2;
                                head2=sorting->next;
                        }
                }
                if(head1==NULL)
```

```
                    sorting->next=head2;
                else if(head2==NULL)
                    sorting->next=head1;
            }
        return new_head;
        }
```

Test Cases:

Input(n)	Output

Q.5. void Count(Node* head, int number) function that counts the number of times a given int occurs in a list.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<malloc.h>
struct node
{
    int data;
    struct node *link;
```

```
};
struct node *start=NULL;
struct node* createnode()
{
        struct node *n;
        n=(struct node *)malloc(sizeof(struct node));
        return(n);
}
void insertnode()
{
        struct node *temp,*t;
        temp=createnode();
        printf("Enter a number\n");
        scanf("%d",&temp->data);
        temp->link=NULL;
        if(start==NULL)
                start=temp;
        else
        {
                t=start;
                while(t->link!=NULL)
                        t=t->link;
                t->link=temp;
        }
}
void displaylist(struct node *start)
{
        struct node *t;
        if(start==NULL)
                printf("List is Empty\n");
        else
                {
                t=start;
                while(t!=NULL)
                        {
                        printf("%d->",t->data);
                        t= t->link;
                        }
                }
}
void count(int search)
{
        struct node *t;
        int count=0;
        t=start;
        while(t!=NULL)
```

```
        {
                if(t->data==search)
                count++;
                t=t->link;
        }
        printf("\n%d",count);
}
main()
{
        int n,i=1,search;
        printf("Enter number of elements\n");
        scanf("%d",&n);
        while(i<=n)
        {
                insertnode();
                i++;
        }
        displaylist(start);
        scanf("%d",&search);
        count(search);
}
```

Test Cases:

Input(n)	Output

Q.7. Program for the Indore Election problem of Assignment-1 using linked list.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<string.h>
#include<malloc.h>
struct node{
        char st_name[30];
        int year;
        int ac_no;
        char ac_name[20];
        char ac_type[20];
        char cand_name[100];
        char cand_sex;
        char partyname[30];
        char partyabbre[10];
        long int totalvotpoll;
        long int electors;
        struct node *link;
};
struct node *start=NULL;
struct node* createnode()
{
        struct node *n;
        n=(struct node *)malloc(sizeof(struct node));
        return(n);
};
struct node* insertNode(struct node *start,char st_name[],int year,int ac_no,char
ac_name[],char ac_type[],char cand_name[],char cand_sex,char partyname[],char
partyabbre[],long int totalvotpoll,long int electors)
{
        struct node *temp,*t;
        temp=createnode();
        temp->year=year;
        temp->ac_no=ac_no;
        strcpy(temp->ac_name,ac_name);
        strcpy(temp->ac_type,ac_type);
        strcpy(temp->cand_name,cand_name);
```

```
		temp->cand_sex=cand_sex;
		strcpy(temp->partyname,partyname);
		strcpy(temp->partyabbre,partyabbre);
		temp->totalvotpoll=totalvotpoll;
		temp->electors=electors;
		temp->link=NULL;
		if(start==NULL)
			start=temp;
		else
		{
			t=start;
			while(t->link!=NULL)
				t=t->link;
			t->link=temp;
		}
	 return(start);
	};
	main()
	{
		struct node *temp,*w;
		float male=0,female=0;
		FILE *fp;
		int i=0,fpr,j;
		char st_name[30];
		int year,tmp;
		int ac_no;
		char ac_name[20];
		char ac_type[20];
		char cand_name[100];
		char cand_sex;
		char partyname[30];
		char partyabbre[10];
		long int totalvotpoll;
		long int electors;
		fp = fopen("Indore_Election_data_modified.txt","r");
		while(1)
		{
		fpr=
	fscanf(fp,"%[^\n\t]\t%d\t%d\t%s\t%s\t%[^\n\t]\t%c\t%[^\n\t]\t%s\t%ld\t%ld\n",

	st_name,&year,&ac_no,ac_name,ac_type,cand_name,&cand_sex,partyname,party
abbre,&totalvotpoll,&electors);

		start=
insertNode(start,st_name,year,ac_no,ac_name,ac_type,cand_name,cand_sex,partyname,p
artyabbre,totalvotpoll,electors);
```

```
                i++;
                if(fpr==-1)
                        break;
                }
                fclose(fp);
                j=i;
                temp=start;
                while(temp!=NULL)
                {
                        w->link=temp;
                        tmp=temp->year;
                        while(tmp==temp->year)
                        {
                                if(temp->link->totalvotpoll > temp->totalvotpoll)
                                {
                                        temp=temp->link;
                                        w->link=temp;
                                }
                                else
                                        temp=temp->link;
                        }
                printf("Winner of election of year %d\tis %s\n",w->year,w->cand_name);
                printf("Percentage of vote given: %f% \n\n",100.00*w->totalvotpoll/w-
>electors);

                temp=temp->link;
                }
                temp=start;
                while(temp!=NULL)
                {
                        if(temp->cand_sex=='M')
                                male++;
                        else if(temp->cand_sex=='F')
                                female++;
                        temp=temp->link;
                }
                printf("\n\n\n\n\nPercentage of male candidates is
%f%\n",100*male/(male+female));
                printf("Percentage of female candidates is
%f%\n\n\n\n\n",100*female/(male+female));
        }
```

Test Cases:

Q.8. Perform the following operations on the linked list:

c. Perform pair-wise swapping of nodes of a given list.
Input:
20 -> 18 -> 15 -> 10 -> 8 -> 6 -> 5 -> 3 -> 7 -> NULL
Output:
18 -> 20 -> 10 -> 15 -> 6 -> 8 -> 3 -> 5 -> 7 -> NULL

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<malloc.h>
struct node
{
        int data;
        struct node *link;
};
struct node *start=NULL;
struct node* createNode()
{
        struct node *n;
        n= (struct node *)malloc(sizeof(struct node));
        return(n);
}
struct node* insertNode(struct node *start,int data)
{
```

```
        struct node *temp,*t;
        temp= createNode();
        temp->data = data;
        temp->link=NULL;
        if(start==NULL)
                start=temp;
        else
        {
                t=start;
                while(t->link!=NULL)
                        t=t->link;
                t->link=temp;
        }
        return start;
}
struct node*pairSwap(struct node *start)
{
        if (start == NULL || start->link == NULL)
                return;
        struct node *prev = start;
        struct node *curr = start->link;
      start=curr;
        while (1)
        {
      struct node *next = curr->link;
      curr->link = prev;
      if (next == NULL || next->link == NULL)
      {
        prev->link = next;
        break;
      }
      prev->link = next->link;
      prev = next;
      curr = prev->link;
   }
return start;
}
void displayNode(struct node *start)
{
        struct node *t;
        if(start==NULL)
                printf("List is Empty\n");
        else{
                t=start;
                while(t!=NULL)
                {       printf("%d->",t->data);
```

```
                t=t->link;
            }}
}
main()
{
    int i=1,data,n;
    printf("Enter number of elements\n");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("Enter data\n");
        scanf("%d",&data);
        start=insertNode(start,data);
    }
start=pairSwap(start);
displayNode(start);
}
```

d. Remove alternate nodes from a given linked list.
Input:
20 -> 18 -> 15 -> 10 -> 8 -> 6 -> 5 -> 3 -> NULL
Output:
20 -> 15 -> 8 -> 5 -> NULL

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<malloc.h>
struct Node
{
        int data;
        struct Node *link;
};
struct Node *start=NULL;
struct Node* createNode()
{
        struct Node *n;
        n=(struct Node*)malloc(sizeof(struct Node));
        return(n);
}
struct Node* insertNode(struct Node *start,int data)
{
        struct Node *temp,*t;
        temp=createNode();
        temp->data=data;
        temp->link=NULL;
        if(start==NULL)
                start=temp;
        else
        {
                t=start;
                while(t->link!=NULL)
                        t=t->link;
                t->link=temp;
        }
 return(start);
}
void delete_alternate_Node()
{
        if(start==NULL)
                printf("List is Empty\n");
        struct Node *p=start;
        struct Node *q=p->link;
        while(p!=NULL && q!=NULL)
        {
      p->link=q->link;
      free(q);
      p=p->link;
      if(p!=NULL)
        q=p->link;
        }
```

```
}
void displayList(struct Node *start)
{
        struct Node *t;
        t=start;
        if(start=NULL)
                printf("List is empty\n");
        else
        {
                while(t!=NULL)
                {
                printf("%d->",t->data);
                t=t->link;
                }
        }
}
main()
{
        int data,n,i;
        char a,r;
        printf("Enter number of data\n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("Enter data\n");
                scanf("%d",&data);
                start =insertNode(start,data);
        }
        delete_alternate_Node();
        displayList(start);
}
```

Test Cases:

Input(n)	Output

e. Given a pair linked lists, insert nodes of second linked list into the first linked list at alternate positions. Assume that the first linked list has at least as many elements as the second.

Input:
1 -> 2 -> 3 -> NULL
4 -> 5 -> NULL
Output:
1 -> 4-> 2 -> 5 -> 3 -> NULL

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<malloc.h>
struct node
{
        int data;
        struct node *link;
};
struct node *start1=NULL,*start2=NULL,*start3=NULL;
struct node* createnode()
{
        struct node *n;
        n=(struct node *)malloc(sizeof(struct node));
        return(n);
```

```
}
struct node* insertnode(struct node *start)
{
        struct node *temp,*t;
        temp=createnode();
        printf("Enter a number\n");
        scanf("%d",&temp->data);
        temp->link=NULL;
        if(start==NULL)
                start=temp;
        else
        {
                t=start;
                while(t->link!=NULL)
                        t=t->link;
                t->link=temp;
        }
        return start;
}
struct node* merge(struct node *start1,struct node *start2)
{
        struct node *temp,*t1,*t2;
        if(start1==NULL &&  start2==NULL)
        return(NULL);
        temp=start1;
        t1=start1;
        t2=start2;
        start3=temp;
        t1=t1->link;
        while(t1!=NULL&&t2!=NULL)
        {
                temp->link=t2;
                t2=t2->link;
                temp->link->link=t1;
                t1=t1->link;
                temp=temp->link->link;
        }
temp->link=t1;
return start3;
}
void displaylist(struct node *start)
{
        struct node *t;
        if(start==NULL)
                printf("List is Empty\n");
        else
```

```
                {
                t=start;
                while(t!=NULL)
                        {
                        printf("%d->",t->data);
                        t= t->link;
                        }
                }
}

main()
{
        int n,i=1;
        printf("Enter number of elements\n");
        scanf("%d",&n);
        while(i<=n)
        {
                start1=insertnode(start1);
                i++;
        }
        i=1;
        displaylist(start1);
        printf("\n");
        while(i<=n-1)
        {
                start2=insertnode(start2);
                i++;
        }
        displaylist(start2);
        start3 =merge(start1,start2);
        printf("\n");
        displaylist(start3);
}
```

Test Cases:

Input(n)	Output

Q.9. Write a program to represent a polynomial in variable X using a singly linked list, each node of which contains two kinds of data fields; one to store coefficient and another stores the power on variable X.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<malloc.h>
struct node
{
int coeff;
int pow;
struct node *link;
};
struct node *start=NULL;
struct node *start1=NULL;
struct node *start2=NULL;
struct node* createnode(int coeff,int pow)
{
        struct node *n;
        n = (struct node *)malloc(sizeof(struct node));
        n->coeff=coeff;
        n->pow=pow;
        return(n);
```

```
}
struct node* insertnode(struct node *start,int coeff,int pow)
{
	struct node *temp,*t;
	temp=createnode(coeff,pow);
	temp->link=NULL;
	if(start==NULL)
		start=temp;
	else
	{
		t=start;
		while(t->link!=NULL)
			t=t->link;
		t->link=temp;
	}
return start;
}
void displaylist(struct node *start)
{
	struct node *t;
	if(start==NULL)
		printf("List is Empty\n");
	else
		{
		t=start;
		while(t!=NULL)
			{
				printf("|%d|%d|->",t->coeff,t->pow);
				t= t->link;
			}
	}
}
struct node* add(struct node *start,struct node *start1)
{
	struct node *t,*t1,*t2;
	int coeff,pow;
	t=start;
	t1=start1;
	t2=start2;
	while(t!=NULL&&t1!=NULL)
	{
	if(t->pow==t1->pow)
	{	coeff=(t->coeff)+(t1->coeff);
		pow=t->pow;
		start2=insertnode(start2,coeff,pow);
	}
```

```
        else
        {
        start2=insertnode(start2,t->coeff,t->pow);
        start2=insertnode(start2,t1->coeff,t1->pow);
        }
        t1=t1->link;
        t=t->link;
        }
        return start2;
}
main()
{
        int n,i=1,coeff,pow;
        printf("Enter number of terms\n");
        scanf("%d",&n);
        while(i<=n)
        {
                printf("Enter coefficient and power\n");
                scanf("%d%d",&coeff,&pow);
                start=insertnode(start,coeff,pow);
                i++;
        }
        displaylist(start);
        i=1;
        printf("\n");
         while(i<=n)
        {
                printf("Enter coefficient and power\n");
                scanf("%d%d",&coeff,&pow);
                start1=insertnode(start1,coeff,pow);
                i++;
        }
        displaylist(start1);
        printf("\n");
        start2 =add(start,start1);
        displaylist(start2);
}
```

Test Cases:

Input(n)	Output

Lab Assignment # 03 (Stack & Queue)

Q.1. Design and write a function for the stack as abstract data type using array and linked list.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
/*Stack implementation through array*/
struct arrayStack
{
        int top;
        int capacity;
        int *array;
```

```
};
struct arrayStack* createStack(int cap)
{
	struct arrayStack *stack;
	stack=malloc(sizeof(struct arrayStack));
	stack->capacity=cap;
	stack->top=-1;
	stack->array=malloc(sizeof(int)*stack->capacity);
	return stack;
};
int isFull(struct arrayStack *stack)
{
	if(stack->top==stack->capacity-1)
		return 1;
	else
		return 0;
}
int isEmpty(struct arrayStack *stack)
{
	if(stack->top==-1)
		return 1;
	else
		return 0;
}
void push(struct arrayStack *stack,int data)
{
	if(!isFull(stack))
	{
		stack->top++;
		stack->array[stack->top]=data;
	}
	else
	printf("\n Stack overflowed\n");
}
int pop(struct arrayStack *stack)
{
	int item;
	if(!isEmpty(stack))
	{
		item=stack->array[stack->top];
		stack->top--;
		return item;
	}
	return -1;
}
```

Program:

```
#include<stdio.h>
#include<stdlib.h>
/*implementation of stack using linked list*/
struct linked_stack
{
	int data;
	struct linked_stack *next;
};
struct linked_stack *top=NULL;
typedef struct linked_stack sl;
sl* createStack()
{
	sl *stack;
	stack=(sl *)malloc(sizeof(struct linked_stack));
	return stack;
}
sl* push(int data)
{
	sl *newNode;
	newNode=createStack();
	newNode->data=data;
	newNode->next=top;
	top=newNode;
	return top;
}
int pop()
{
	sl *r;
	r=top;
	int data;
	if(top==NULL)
		printf("Stack is underflow");
	else
	{
		data=r->data;
		top=top->next;
		free(r);
		return r;
	}
}
void display()
{
	sl *p;
	p=top;
	if(top==NULL)
```

```
		printf("Stack is underflow\n");
	while(p!=NULL)
	{
		printf("%d\n",p->data);
		p=p->next;
	}
}
```

Q.2. Design and write a function for the queue as abstract data type using array and linked list.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct arrayQueue
{
	int front,rear;
	int capacity;
	int *array;
};
struct arrayQueue* createQueue(int capacity)
{
```

```
	struct arrayQueue *q;
	q = malloc(sizeof(struct arrayQueue));
	q->capacity=capacity;
	q->rear=q->front=-1;
	q->array=malloc(q->capacity*(sizeof(int)));
	if(!q->array)
		return NULL;
	return q;
}
int isEmptyQueue(struct arrayQueue *q)
{
	return (q->front==-1);
}
int isFullQueue(struct arrayQueue *q)
{
	return((q->rear+1)%q->capacity==q->front);
}
int queueSize()
{
	return((q->capacity-q->front+q->rear+1)%q->capacity);
}
void enQueue(struct arrayQueue *q,int data)
{
	if(isFullQueue(q))
		printf("Queue overflow\n");
	else
	{
		q->rear=(q->rear+1)%q->capacity;
		q->array[q->rear]=data;
		if(q->front==-1)
			q->front=q->rear;
	}
}
int deQueue(struct arrayQueue *q)
{
	int data=-1;
	if(isEmptyQueue(q))
	{
		printf("Queue is empty\n");
		return(-1);
	}
	else
	{
		data=q->array[q->front];
		if(q->front==q->rear)
			q->front=q->rear=-1;
```

```
                else
                        q->front=(q->front+1)%q->capacity;
        }
        return data;
}
void deleteQueue()
{
        if(q)
                if(q->array)
                        free(q->array);
        free(q);
}
```

Program:

```
#include <stdio.h>
#include <stdlib.h>
int count = 0;
struct node
{
        int information;
        struct node *next;
}*start,*end,*temp,*start1;
void create()
{
        start = end = NULL;
}
void queueSize()
{
        printf("\n Queue size is %d", count);
}
void enqueue(int data)
{
        if (end == NULL)
        {
                end = (struct node *)malloc(1*sizeof(struct node));
                end->next = NULL;
                end->information = data;
                start = end;
        }
        else
        {
                temp=(struct node *)malloc(1*sizeof(struct node));
                end->next = temp;
                temp->information = data;
                temp->next = NULL;
```

```
                end = temp;
        }
}
void display()
{
        start1 = start;

        if (start1 == NULL && end == NULL)
        {
                printf("Queue is empty");
                return;
        }
   while (start1 != end)
   {
        printf("%d ", start1->information);
        start1 = start1->next;
   }
   if (start1 == end)
        printf("%d", start1->information);
}
void deQueue()
{
        start1 = start;
        if (start1 == NULL)
        {
           printf("\nEmpty queue");
           return;
        }
        else
        if (start1->next != NULL)
        {
                start1 = start1->next;
                printf("%d, ", start->information);
                free(start);
                start = start1;
        }
        else
        {
                printf("\n Dequed value : %d", start->information);
                free(start);
                start = NULL;
                end = NULL;
        }
      count--;
}
int startelement()
```

```
{
   if ((start != NULL) && (end != NULL))
      return(start->information);
   else
      return 0;
}
void empty()
{
    if ((start == NULL) && (end == NULL))
         printf("\n Queue empty");
   else
         printf("Queue not empty");
}
```

Q.3. Write a function that reverses the order of words in a sentence.

Concept/Pseudocode:

Program:

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

```
#include<stdlib.h>
struct wordsStack
{
	char word[50];
};
char* remove_space(char *s)
{
	char *p;
	int i=0,j=0;
	p=malloc(strlen(s)+1);
	while(*(s+i))
	{	while(*(s+i)==' ')
			i++;
	while(*(s+i)!=' ' && *(s+i)!='\0')
	{
		*(p+j)=*(s+i);
		i++;
		j++;
	}
	if(*(s+i)=='\0'&&*(p+j-1)==' ')
		j--;
	*(p+j)=*(s+i);
	j++;
	}
	return p;
}
int count_words(char *s)
{
	int i=0,count=0;
	while(*(s+i))
	{
		if(*(s+i)==' ')
			count++;
		i++;
	}
	return count+1;
}
char* reverseWordsOfSentence(struct wordsStack *ptr,char *s)
{
char temp[50],*q;
int i=0,j,top=0;
while(*(s+i))
{	j=0;
	while(*(s+i)!=' '&& *(s+i)!='\0')
	{
		temp[j]=*(s+i);
```

```
                j++;
                i++;
        }
        if(*(s+i)==' ')
        i++;
        temp[j]='\0';
        strcpy(ptr[top].word,temp);
        top++;
}
q=(char*)malloc(strlen(s)+1);
*(q+0)='\0';
top--;
while(top)
{
        strcat(q,ptr[top].word);
        strcat(q," ");
        top--;
}
strcat(q,ptr[0].word);
return q;
}
main()
{
        int no_words;
        char str[256];
        struct wordsStack *ptr;
        printf("Enter string\n");
        gets(str);
        strcpy(str,remove_space(str));
        no_words=count_words(str);
        ptr=(struct wordsStack *)calloc(no_words,sizeof(struct wordsStack));
        strcpy(str,reverseWordsOfSentence(ptr,str));
        printf("Reversed String is\n%s \n",str);
}
```

Test Cases:

Input(n)	Output

Q.4. Write a function prefix_to_infix().

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct arrayStack
{
        int top;
        int capacity;
        char *array;
};
struct arrayStack* createStack(int cap)
{
        struct arrayStack  *stack;
        stack=malloc(sizeof(struct arrayStack));
        stack->capacity=cap;
        stack->top=-1;
        stack->array=malloc(sizeof(char)*stack->capacity);
```

```
	return stack;
};
int isFull(struct arrayStack *stack)
{
	if(stack->top==stack->capacity-1)
		return 1;
	else
		return 0;
}
int isEmpty(struct arrayStack *stack)
{
	if(stack->top==-1)
		return 1;
	else
		return 0;
}
void push(struct arrayStack *stack,char data)
{
	if(!isFull(stack))
	{
		stack->top++;
		stack->array[stack->top]=data;
	}
	else
	printf("\n Stack overflowed\n");
}
char pop(struct arrayStack *stack)
{
	char item;
	if(!isEmpty(stack))
	{
		item=stack->array[stack->top];
		stack->top--;
		return item;
	}
	return -1;
}
char str[100];
int check(struct arrayStack *stack)
{
	char data;
	if(isEmpty(stack))
		return 1;
	data=pop(stack);
	push(stack,data);
```

```
		if(data=='/' || data=='^' || data=='*' || data=='+' || data=='-' || data=='%' ||
data==') ')
			return 0;
		else
			return 1;
	}
	void prefix_to_infix()
	{
		int n,i,j,k,l;
		char data,op[50],po,pc,c;
		po='(';
		pc=')';
		struct arrayStack *stack;
		n=strlen(str);
		stack=createStack(n+20);
		for(i=n-1;i>=0;i--)
		{
			data=str[i];
			if(data=='/' || data=='^' || data=='*' || data=='+' || data=='-' ||
data=='%')
			{
				op[0]=pop(stack);
				op[1]=pop(stack);
				if(op[0]!='(' && op[1]!='(')
				{
					push(stack,pc);
					push(stack,op[1]);
					push(stack,data);
					push(stack,op[0]);
					push(stack,po);
				}
				else{
					j=0;
					push(stack,op[1]);
					push(stack,op[0]);
					while(1)
					{
						op[j]=pop(stack);
						if(op[0]=='(')
						{
							if(op[j]==')' && check(stack))
								break;
						}
						else
							if(op[j]=='(' && check(stack))
								break;
```

```
                                        j++;
                                }
                                k=j;
                                j=j+1;
                                while(1)
                                {
                                        op[j]=pop(stack);
                                        if(!isEmpty(stack))
                                        {
                                                c=pop(stack);
                                                push(stack,c);
                                        }
                                        if((op[j]==')' && c!=')') || isEmpty(stack))
                                                break;
                                        j++;
                                }
                                push(stack,pc);
                                j++;
                                for(l=j;l>0;l--)
                                {
                                        if(l==k)
                                                push(stack,data);
                                        push(stack,op[l]);
                                }
                                push(stack,po);
                        }
                }
                else
                        push(stack,data);
        }
        while(!isEmpty(stack))
        {
                str[i]=pop(stack);
                printf("%c",str[i]);
                i++;
        }
        str[i]='\0';
}
main()
{
        gets(str);
        prefix_to_infix();
}
```

Test Cases:

Input(n)	Output

Q.7. Suppose there is circle, and it has N petrol pumps. The petrol pumps are numbered from 0 to N-1. For each petrol pump, you know the amount of petrol available on it and the distance of next petrol pump. Assuming your car run 1 Km in 1 liter of petrol, you need to find out the petrol pump, from where you should start so that you can finish the circle.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct arrayQueue
{
        int front,rear;
        int capacity;
        int *petrol;
```

```
        int *distance;
}*q;
struct arrayQueue* createQueue(int capacity)
{
        struct arrayQueue *q;
        q = malloc(sizeof(struct arrayQueue));
        q->capacity=capacity;
        q->rear=q->front=-1;
        q->petrol=malloc(q->capacity*(sizeof(int)));
        q->distance=malloc(q->capacity*(sizeof(int)));
        return q;
}
int isEmptyQueue(struct arrayQueue *q)
{
        return (q->front==-1);
}
int isFullQueue(struct arrayQueue *q)
{
        return((q->rear+1)%q->capacity==q->front);
}
int queueSize()
{
        return((q->capacity-q->front+q->rear+1)%q->capacity);
}
void enQueue(struct arrayQueue *q,int petrol,int distance)
{
        if(isFullQueue(q))
                printf("Queue overflow\n");
        else
        {
                q->rear=(q->rear+1)%q->capacity;
                q->petrol[q->rear]=petrol;
                q->distance[q->rear]=distance;
                if(q->front==-1)
                        q->front=q->rear;
        }
}
void deQueue(struct arrayQueue *q,int *p,int *d)
{
        if(isEmptyQueue(q))
        {
                printf("Queue is empty\n");
        }
        else
        {
                *p=q->petrol[q->front];
```

```
		*d=q->distance[q->front];
		if(q->front==q->rear)
			q->front=q->rear=-1;
		else
			q->front=(q->front+1)%q->capacity;
	}
}
main()
{
	struct arrayQueue *q;
	int left,n,petrol,distance,i,sop=0,sod=0,start=-1,flag=0,p=0,d=0;
	printf("Enter number of petrol pumps\n");
	scanf("%d",&n);
	q=createQueue(n);
	for(i=1;i<=n;i++)
	{
		scanf("%d%d",&petrol,&distance);
		sop=sop + petrol;
		sod=sod + distance;
		enQueue(q,petrol,distance);
	}
	if(sod>sop)
		printf("Car can never complete marathon");
	else
	{
		while(1)
		{
			while(1)
			{
				deQueue(q,&p,&d);
				start++;
				left=p-d;
				if(left<0)
					enQueue(q,p,d);
				else
					break;
			}
			for(i=start+1;i<n;i++);
			{
				left=left+q->petrol[i]-q->distance[i];
				if(left<0)
				{
					flag=1;
					break;
				}
			}
```

```
                        if(flag!=1)
                                for(i=0;i<start;i++)
                                {
                                        left=left+q->petrol[i]-q->distance[i];
                                        if(left<0)
                                        {
                                                flag=1;
                                                break;
                                        }
                                }
                        enQueue(q,p,d);
                        if(flag!=1)
                                break;
                }
                printf("Car should start from %d petrol pump\n",start+1);
        }
}
```

Test Cases:

Input(n)	Output

Q.8. Write a program to implement the queue using two stack.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *next;
};
struct node **top=NULL;
struct queue
{
        struct node *stack1;
        struct node *stack2;
}*q;
void enqueue (struct queue *q, int x)
{
        push (&q->stack1,x);
}

void dequeue (struct queue *q)
{
        int x;
        if (q->stack1 == NULL && q->stack2 == NULL)
          {
            printf ("queue is empty");
          }
        if (q->stack2 == NULL)
          {
            while (q->stack1 != NULL)
                {
                  x = pop (q->stack1);
                  push (&q->stack2, x);
                }
          }
        x = pop (&q->stack2);
        printf ("%d\n", x);
}

void push (struct node **top, int data)
```

```
{
        struct node *newnode = (struct node *)malloc(sizeof (struct node));
        if (newnode == NULL)
          {
            printf ("Stack overflowed \n");
          }
        else
         {
        newnode->data = data;
        newnode->next = (*top);
        (*top) = newnode;
       }
}
int pop (struct node *top)
{
        int data;
        struct node *t;
        if ((top)==NULL)
          {
            printf ("Stack is underflow \n");
            return;
          }
        else
          {
            t = top;
            data=t->data;
            top = t->next;
            free (t);
            return data;
       }
}

void display (struct node *top1, struct node *top2)
{
        while (top1 != NULL)
          {
            printf ("%d\n", top1->data);
            top1 = top1->next;
          }
        while (top2 != NULL)
          {
            printf ("%d\n", top2->data);
            top2 = top2->next;
          }
}
int main ()
```

```
{
        struct queue *q = (struct queue *) malloc (sizeof (struct queue));
        int f = 0, a;
        char ch='y';
        q->stack1 = NULL;
        q->stack2 = NULL;
        do
         {
           printf ("enter your choice\n1.add to queue\n2.remove from
queue\n3.display\n4.exit\n");
           scanf ("%d", &f);
           switch (f)
                {
                case 1:
                  printf ("enter the element to be added to queue\n");
                  scanf ("%d", &a);
                  enqueue (q, a);
                  break;
                case 2:
                  dequeue (q);
                  break;
                case 3:
                  display (q->stack1, q->stack2);
                  break;
                case 4:
                  exit (1);
                  break;
                default:
                  printf ("invalid\n");
                  break;
                }
        }while(ch=='y');
}
```

Test Cases:

Input(n)	Output

Lab Assignment # 04 (Tree)

Q.1. Design an algorithm which when given a binary search tree and two numbers x and y outputs all the data items z in order with the property that x <= z<= y.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *left;
   struct node *right;
};
typedef struct node Node;
Node* getNode(int x)
```

```
{
   Node *temp = (Node*)malloc(sizeof(Node));
   temp->data = x;
   temp->left = NULL;
   temp->right = NULL;
   return temp;
}
Node * insert(Node* root, int x)
{
   if ( root == NULL){
      root = getNode(x);
   }
   else if (x <= root->data){
      root->left = insert(root->left,x);
   }
   else if ( x> root->data){
      root->right = insert(root->right,x);
   }
   return root;
}
void inorder(Node * root)
{
   if (root !=NULL)
   {
      inorder(root->left);
      printf("%d,",root->data);
      inorder(root->right);
   }
}
void btw(Node *root,int x,int y)
{
        if(root==NULL)
                return;
        else{
                if(root->data>=x)
                {
                        if(root->data<=y)
                        {
                                printf("%d, ",root->data);
                                btw(root->right,x,y);
                        }
                        else
                                btw(root->left,x,y);
                        btw(root->left,x,y);
                }
                else
```

```
                btw(root->right,x,y);
        }
}
main()
{
        Node *root;
        int n,i,data,x,y;
        printf("Enter number of elements\n");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                scanf("%d",&data);
                root=insert(root,data);
        }
        inorder(root);
        printf("\n");
        scanf("%d%d",&x,&y);
        btw(root,x,y);
        printf("\n");
}
```

Test Cases:

Input(n)	Output

Q.4. Write a program for binary search tree with various functions.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *left;
        struct node *right;
};
typedef struct node Node;
Node* getNode(int x)
{
        Node *temp = (Node*)malloc(sizeof(Node));
        temp->data = x;
        temp->left = NULL;
        temp->right = NULL;
        return temp;
}
Node * insert(Node* root, int x)
{
        if ( root == NULL){
      root = getNode(x);
        }
        else if (x <= root->data){
      root->left = insert(root->left,x);
        }
        else if ( x> root->data){
      root->right = insert(root->right,x);
        }
        return root;
}
void inorder(Node * root)
{
        if (root !=NULL)
        {
           inorder(root->left);
```

```
            printf("%d,",root->data);
            inorder(root->right);
        }
}
int contains(Node *root, int i)
{
        int flag=0;
        if(root!=NULL)
        {
                if(root->data>=i)
                {
                        if(root->data==i)
                                flag=1;
                else
                        flag=contains(root->left,i);
                }
            else{
                        flag=contains(root->right,i);
                }
        }
     return flag;
}
int min(Node *root)
{
        int k=-1;
        if(root==NULL)
                k=-1;
        else{
        if(root->left==NULL)
                k=root->data;
      else
                k=min(root->left);
        }
        return k;
}

int max(Node *root)
{
        int k=-1;
        if(root==NULL)
                k=-1;
        else{
                if(root->right==NULL)
                        k=root->data;
        else
                        k=max(root->right);
```

```
	}
	return k;
}
int isComplete(Node* root)
{
	int flag=0,c,l,r;
	if(root==NULL)
		c=1;
	else
	{
		if(root->left!=NULL && root->right!=NULL)
		{
			l=isComplete(root->left);
			r=isComplete(root->right);
			c=l&r;
		}
		else if(root->left==NULL && root->right==NULL)
			c=1;
		else
			c=0;
	}
	return c;
}
Node* min_addr(Node* root)
{
	if(root->left!=NULL)
		root=min_addr(root->left);
	return root;
}
Node* delete(Node* root,int i)
{
	Node* temp;
	int b;
	b=contains(root,i);
	if(root==NULL || b==0)
		return root;
	else if(b==1){
		if(i<root->data )
			root->left=delete(root->left,i);
		else if(i>root->data)
			root->right=delete(root->right,i);
		else
		{
			if(root->left==NULL && root->right==NULL)
			{
				free(root);
```

```
                root=NULL;
                return root;
            }
            else if(root->left==NULL)
            {
                temp=root;
                root=root->right;
                free(temp);
                return root;
            }
            else if(root->right==NULL)
            {
                temp=root;
                root=root->left;
                free(temp);
                return root;
            }
            else
            {
                temp=min_addr(root->right);
                root->data=temp->data;
                root->right=delete(root->right,temp->data);
                return root;
            }
        }
    }
    return root;
}

int main()
{
    int choice,data,b;
    Node *root;
    while(1)
    {

    printf("1.Contains\n2.Insert\n3.Delete\n4.Min\n5.Max\n6.Complete\n7.In
order traversal\n0.To exit\n");
            scanf("%d",&choice);
            switch(choice)
            {
                case 0: exit(0);
                        break;
                case 1: printf("Enter the element\n");
                        scanf("%d",&data);
                        b=contains(root,data);
```

```
                                        if(b==1)
                                                printf("True\n");
                                        else
                                                printf("False\n");
                                        break;
                                case 2: printf("Enter the element to be inserted\n");
                                        scanf("%d",&data);
                                        root=insert(root,data);
                                        break;
                                case 3: printf("Enter the data to be deleted\n");
                                        scanf("%d",&data);
                                        root=delete(root,data);
                                        break;
                                case 4: data=min(root);
                                        printf("Minimum element is
        %d\n",data);
                                        break;
                                case 5: data=max(root);
                                        printf("Maximum element is
        %d\n",data);
                                        break;
                                case 6: b=isComplete(root);
                                        if(b==1)
                                                printf("True\n");
                                        else
                                                printf("False\n");
                                        break;
                                case 7: inorder(root);
                                        printf("\n");
                                        break;
                                default: printf("Invalid choice");
                                         break;
                        }
                }

        }
```

Q.5. Write a function to find the longest path of the tree.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   int ht;
   struct node *left;
   struct node *right;
};
typedef struct node Node;
Node* getNode(int x)
{
   Node *temp = (Node*)malloc(sizeof(Node));
   temp->data = x;
   temp->left = NULL;
   temp->ht=1;
   temp->right = NULL;
   return temp;
}
Node * insert(Node* root, int x)
{
   if ( root == NULL){
      root = getNode(x);
   }
   else if (x <= root->data){
      root->left = insert(root->left,x);
      root->left->ht=root->ht+1;
   }
   else if ( x> root->data){
      root->right = insert(root->right,x);
```

```
        root->right->ht=root->ht+1;
    }
    return root;
}
void inorder(Node * root)
{
    if (root !=NULL)
    {
        inorder(root->left);
        printf("%d,",root->data);
        inorder(root->right);
    }
}
int height(Node *T)
{
    int lh,rh;
    if(T==NULL)
        return(0);
    lh=height(T->left);
    rh=height(T->right);
    if(lh>rh)
        return(1+lh);
    else
            return 1+rh;
}
void maxLength(Node* root,int *maxl, int *maxr)
{
            int l=1,r,lm=-1,rm=-1;
            if(root==NULL)
                    return;
            else{
            if(root->left!=NULL)
            {
                    l=height(root->left);
                    r=height(root->right);
                    if(*maxl+*maxr < l+r)
                    {
                            *maxl=l;
                            *maxr=r;
                    }
                    maxLength(root->left,maxl,maxr);
            }
            if(root->right!=NULL)
            {
                    l=height(root->left);
                    r=height(root->right);
```

```
                    if(*maxl+*maxr < l+r)
                    {
                            *maxl=l;
                            *maxr=r;
                    }
                    maxLength(root->right,maxl,maxr);
            }
            }
}
int main()
{
   int i,n,j,maxl=0,maxr=0;
   Node * root = NULL;
   printf("Enter number of elements\n");
   scanf("%d",&j);
   for(i=1; i<=j; i++)
   {
         scanf("%d",&n);
      root = insert(root,n);
   }
   printf("\nInorder : ");
   inorder(root);
   maxLength(root,&maxl,&maxr);
   printf("\nLongest path of tree is : %d\n",maxl+maxr);
}
```

Test Cases:

Input(n)	Output

Q.6. Given a binary search tree, find the maximum vertical level and horizontal level sum in binary tree.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        int hd;
        int vd;
        struct node *left;
        struct node *right;
};
typedef struct node Node;
Node* getNode(int x)
{
        Node *temp = (Node*)malloc(sizeof(Node));
        temp->data = x;
        temp->left = NULL;
        temp->right = NULL;
        temp->hd=0;
        temp->vd=0;
        return temp;
}
Node* insert(Node* root, int x)
{
        if ( root == NULL){
            root = getNode(x);
        }
   else if (x <= root->data){
            root->left = insert(root->left,x);
```

```
            root->left->hd=root->hd+1;
            root->left->vd=root->vd+1;
    }
    else if ( x> root->data){
            root->right = insert(root->right,x);
            root->right->hd=root->hd-1;
            root->right->vd=root->vd+1;
    }
        return root;
}
void inorder(Node* root)
{
        if (root !=NULL)
        {
                inorder(root->left);
                printf("%d, ",root->data);
                inorder(root->right);
    }
}
void sum(Node *root,int *s,int hd)
{
        if(root==NULL)
                return;
        else{
                if(root->hd==hd)
                {
                        *s = *s + root->data;
                }
            sum(root->left,s,hd);
            sum(root->right,s,hd);
         }
}
int verticalSum(Node* root)
{
        int max,hd=0,s=0;
        Node* p=root;
        if(root==NULL)
                return;
        else{
                sum(p,&s,hd);
                max=s;
                hd++;
                while(p->left!=NULL)
                {
                        s=0;
                        sum(p,&s,hd);
```

```
                        if(s>max)
                                max=s;
                        p->left=p->left->left;
                        hd++;
                }
                hd=-1;
                while(p->right!=NULL)
                {
                        s=0;
                        sum(p,&s,hd);
                        if(s>max)
                                max=s;
                        p->right=p->right->right;
                        hd--;
                }
        }
        return max;
}
int height(Node *root)
{
        int lh,rh,h;
        if(root==NULL)
                return 0;
        lh=height(root->left);
        rh=height(root->right);
        if(lh>rh)
                h=1+lh;
        else
                h=1+rh;
        return h;
}
void hsum(Node *root,int *s,int vd)
{
        if(root==NULL)
                 return;
        else{
                if(root->vd==vd)
                {
                        *s = *s + root->data;
                }
         hsum(root->left,s,vd);
         hsum(root->right,s,vd);
         }
}
int horiSum(Node* root,int h)
{
```

```
int max=-1,vd=0,s=0;
        if(root==NULL)
                return;
        else{
                while(vd<=h)
                {
                        s=0;
                        hsum(root,&s,vd);
                        if(s>max)
                                max=s;
                        vd++;
                }
        }
        return max;
}
main()
{
        Node *root,*p;
        int n,i,data,vsum,h;
        printf("Enter number of elements\n");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                scanf("%d",&data);
                root=insert(root,data);
        }
        inorder(root);
        p=root;
        h=height(p);
        vsum=horiSum(root,h);
        printf("\nMaximum horizontal sum=%d",vsum);
        vsum=verticalSum(p);
        printf("\nMaximum vertical sum=%d\n",vsum);
}
```

Test Cases:

Input(n)	Output

Q.7. Given two arrays that represent in-order and post-order traversals of a full binary tree, construct the binary tree.

Concept/Pseudocode:

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
        int data;
        struct node *left;
        struct node *right;
};
struct node* newNode (int data)
{
        struct node* temp = (struct node *) malloc(sizeof(struct node));
        temp->data = data;
        temp->left = temp->right = NULL;
        return temp;
}
struct node* constructTree(int pre[],int post[],int* preIndex,int l,int h, int
size)
{
```

```
        if (*preIndex >= size || l > h)
                return NULL;
        struct node* root = newNode ( pre[*preIndex] );
    ++*preIndex;
    if (l==h)
        return root;
    int i;
    for (i=l;i<=h;i++)
        if (pre[*preIndex] == post[i])
                break;
    if (i <= h)
    {
                root->left = constructTree(pre, post, preIndex, l, i, size);
                root->right = constructTree(pre, post, preIndex, i + 1, h,
size);
    }
    return root;
}
struct node *construct_Tree (int pre[], int post[], int size)
{
        int preIndex = 0;
        return constructTree(pre,post,&preIndex,0,size-1,size);
}
void printInorder (struct node* node)
{
        if (node == NULL)
                return;
        printInorder(node->left);
        printf("%d ", node->data);
        printInorder(node->right);
}
int main ()
{
        int n,i;
        printf("Enter number of elements\n");
        scanf("%d",&n);
        int pre[n];
        int post[n];
        printf("Enter elements in pre order\n");
        for(i=0;i<n;i++)
                scanf("%d",&pre[i]);
        printf("Enter elements in post order\n");
        for(i=0;i<n;i++)
                scanf("%d",&post[i]);
        struct node *root = construct_Tree(pre,post,n);
        printf("Inorder traversal of the constructed tree: \n");
```

```
        printInorder(root);
        return 0;
}
```

Test Cases:

Input(n)	Output

Q.8. Write a program of Heap Sort.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct heap
{
	int *array;
	int count;
	int capacity;
};
typedef struct heap heap;
heap* createHeap(int capacity)
{
	heap *h;
	h=(heap *)malloc(sizeof(heap));
	h->count=0;
	h->capacity=capacity;
	h->array=(int *)malloc(sizeof(int)*h->capacity);
	return h;
}
int leftchild(heap *h,int i)
{
	int l;
	l=2*i+1;
	if(l>=h->count)
		return -1;
	return l;
}
int rightchild(heap *h,int i)
{
	int r;
	r=2*i+2;
	if(r>=h->count)
		return -1;
	return r;
}
void percolateDown(heap *h,int i)   //Heapify
{
	int l,r,max,temp;
	l=leftchild(h,i);
	r=rightchild(h,i);
	if(l!=-1 && h->array[l]>h->array[i])
		max=l;
	else
		max=i;
	if(r!=-1 && h->array[r]>h->array[max])
		max=r;
	if(max!=i)
```

```
        {
                temp=h->array[i];
                h->array[i]=h->array[max];
                h->array[max]=temp;
        }
        if(max==i)
                return;
        else
                percolateDown(h,max);
}
void buildHeap(heap *h,int a[],int n)
{
        int i;
        if(h==NULL)
                return;
        for(i=0;i<n;i++)
                h->array[i]=a[i];
        h->count=n;
        for(i=(n-3)/2;i>=0;i--)
                percolateDown(h,i);
}
void heapSort(int a[],int n)
{
        heap *h;
        int oldsize,i,temp;
        h= createHeap(n);
        buildHeap(h,a,n);
        oldsize=h->count;
        for(i=n-1;i>0;i--)
        {
                temp=h->array[0];
                h->array[0]=h->array[h->count-1];
                h->array[h->count-1]=temp;
                h->count--;
                percolateDown(h,0);
        }
        h->count=oldsize;
        printf("Sorted array is \n");
        for(i=0;i<n;i++)
                printf("%d, ",h->array[i]);
}
main()
{
        int i,n;
        printf("Enter number of elements\n");
        scanf("%d",&n);
```

```
        int a[n];
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        heapSort(a,n);
}
```

Test Cases:

Input(n)	Output

Q.9. Write a program to convert the min heap to max heap.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct heap
{
        int *array;
        int count;
        int capacity;
};
typedef struct heap heap;
heap* createHeap(int capacity)
{
        heap *h;
        h=(heap *)malloc(sizeof(heap));
        h->count=0;
        h->capacity=capacity;
        h->array=(int *)malloc(sizeof(int)*h->capacity);
        return h;
}
int leftchild(heap *h,int i)
{
        int l;
        l=2*i+1;
        if(l>=h->count)
                return -1;
        return l;
}
int rightchild(heap *h,int i)
{
        int r;
        r=2*i+2;
        if(r>=h->count)
                return -1;
        return r;
}
void percolateDown(heap *h,int i)   //Heapify
{
        int l,r,max,temp;
        l=leftchild(h,i);
        r=rightchild(h,i);
        if(l!=-1 && h->array[l]>h->array[i])
                max=l;
        else
                max=i;
        if(r!=-1 && h->array[r]>h->array[max])
```

```
                max=r;
        if(max!=i)
        {
                temp=h->array[i];
                h->array[i]=h->array[max];
                h->array[max]=temp;
        }
        if(max==i)
                return;
        else
                percolateDown(h,max);
}
void buildHeap(heap *h,int a[],int n)
{
        int i;
        if(h==NULL)
                return;
        for(i=0;i<n;i++)
                h->array[i]=a[i];
        h->count=n;
        for(i=(n-3)/2;i>=0;i--)
                percolateDown(h,i);
}
main()
{
        int n,i;
        heap *h;
        printf("Enter number of elements\n");
        scanf("%d",&n);
        int a[n];
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        h=createHeap(n);
        buildHeap(h,a,n);
        for(i=0;i<n;i++)
                printf("%d, ",h->array[i]);
}
```

Test Cases:

Input(n)	Output

Lab Assignment # 05 (Graph)

Q.1. To check whether there is a path between two given cells in the maze, it suffices to check that there is a path between the corresponding two vertices in the graph.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
int v,e,o,d;
void input()
{
	printf("Enter number of nodes and number of edges\n");
	scanf("%d%d",&v,&e);
	printf("Enter origin and destination cells\n");
	scanf("%d%d",&o,&d);
}
int max(int a,int b)
{
	if(a>b)
		return a;
	else
		return b;
```

```
}
void warshall(int a[v][v])
{
	int i,j,k;
	for(k=0;k<v;k++)
		for(i=0;i<v;i++)
			for(j=0;j<v;j++)
				a[i][j]=max(a[i][j],a[i][k]&&a[k][j]);
}
main()
{
	input();
	int i,j,u,l;
	int a[v][v];
	for(i=0;i<v;i++)
		for(j=0;j<v;j++)
			a[i][j]=0;
	printf("Enter node number in pairs that connects an
edge\n");
	for(i=0;i<e;i++)
	{
		scanf("%d%d",&u,&l);
		a[u][l]=1;
		a[l][u]=1;
	}
	warshall(a);
	if(a[o][d]==1)
		printf("Possible\n");
	else
		printf("Not possible\n");
}
```

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct graph
{
        int v;
        int e;
        int d,o;
        int **adj;
};
typedef struct graph graph;
struct arrayStack
{
        int top;
        int capacity;
        int *array;
};
struct arrayStack* createStack(int cap)
{
        struct arrayStack *stack;
        stack=malloc(sizeof(struct arrayStack));
        stack->capacity=cap;
        stack->top=-1;
        stack->array=malloc(sizeof(int)*stack->capacity);
        return stack;
};
int isFull(struct arrayStack *stack)
{
        if(stack->top==stack->capacity-1)
                return 1;
        else
                return 0;
}
int isEmpty(struct arrayStack *stack)
{
        if(stack->top==-1)
                return 1;
        else
                return 0;
}
void push(struct arrayStack *stack,int data)
{
        if(!isFull(stack))
        {
```

```
                stack->top++;
                stack->array[stack->top]=data;
        }
        else
        printf("\n Stack overflowed\n");
}
int pop(struct arrayStack *stack)
{
        int item;
        if(!isEmpty(stack))
        {
                item=stack->array[stack->top];
                stack->top--;
                return item;
        }
        return -1;
}
graph* adjMatrixofgraph()
{
        int u,v,i;
        graph *g;
        g=(graph *)malloc(sizeof(graph));
        printf("Enter number of nodes and number of edges\n");
        scanf("%d%d",&g->v,&g->e);
        g->adj=(int **)malloc(sizeof(int)*g->v);
        for(i=0;i<g->v;i++)
                g->adj[i]=(int *)malloc(sizeof(int)*g->v);
        for(u=0;u<g->v;u++)
                for(v=0;v<g->v;v++)
                        g->adj[u][v]=0;
        if(g->e!=0)
        printf("Enter node number in pairs that connects an
edge\n");
        for(i=0;i<g->e;i++)
        {
                scanf("%d%d",&u,&v);
                g->adj[u][v]=1;
                g->adj[v][u]=1;
        }
        printf("Enter origin node and destinaton node to check
path\n");
        scanf("%d%d",&g->o,&g->d);
        return g;
}
int check(graph *g)
{
```

```
        int flag=0,n,i,u=0,p,j=0;
        struct arrayStack *stack;
        n=g->v;
        stack=createStack(n);
        int a[n];
        for(i=0;i<n;i++)
                a[i]=1;
        push(stack,u);
        if(g->o > g->d)
        {
                p=g->o;
                g->o=g->d;
                g->d=p;
        }
        while(!isEmpty(stack))
        {
                p=pop(stack);
                a[p]=3;
                j=0;
                for(i=0;i<g->v;i++)
                {
                        if(g->adj[p][i]==1 && a[i]!=3 && a[i]!=2)
                        {
                                push(stack,i);
                                a[i]=2;
                                j=1;
                                if(i==g->d && (a[g->o]==2 || a[g-
>o]==3))
                                        return 1;
                        }
                }
                if(j==0 && a[g->o]!=1)
                        return 0;
                else if(j==0 && p<n-1)
                        push(stack,p+1);
        }
        return 1;
}
main()
{
        int b;
        graph *g;
        g=adjMatrixofgraph();
        b=check(g);
        if(b==1)
                printf("Possible\n");
```

```
        else
                printf("Not possible\n");
    }
```

Test Cases:

Input(n)	Output

Q.2 You construct the following directed graph: vertices correspond to courses, there is a directed edge(u, v) is the course u should be taken before the course v. Then, it is enough to check whether the resulting graph contains a cycle.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct graph
{
	int v;
	int e;
	int **adj;
};
typedef struct graph graph;
struct arrayStack
{
	int top;
	int capacity;
	int *array;
};
struct arrayStack* createStack(int cap)
{
	struct arrayStack *stack;
	stack=malloc(sizeof(struct arrayStack));
	stack->capacity=cap;
	stack->top=-1;
	stack->array=malloc(sizeof(int)*stack->capacity);
	return stack;
};
int isFull(struct arrayStack *stack)
{
	if(stack->top==stack->capacity-1)
		return 1;
	else
		return 0;
}
int isEmpty(struct arrayStack *stack)
{
	if(stack->top==-1)
		return 1;
	else
		return 0;
}
void push(struct arrayStack *stack,int data)
{
	if(!isFull(stack))
	{
		stack->top++;
		stack->array[stack->top]=data;
```

```
		}
		else
		printf("\n Stack overflowed\n");
	}
	int pop(struct arrayStack *stack)
	{
		int item;
		if(!isEmpty(stack))
		{
			item=stack->array[stack->top];
			stack->top--;
			return item;
		}
		return -1;
	}
	graph* adjMatrixofgraph()
	{
		int u,v,i;
		graph *g;
		g=(graph *)malloc(sizeof(graph));
		printf("Enter number of nodes and number of edges\n");
		scanf("%d%d",&g->v,&g->e);
		g->adj=(int **)malloc(sizeof(int)*g->v);
		for(i=0;i<g->v;i++)
			g->adj[i]=(int *)malloc(sizeof(int)*g->v);
		for(u=0;u<g->v;u++)
			for(v=0;v<g->v;v++)
				g->adj[u][v]=0;
		printf("Enter node number in pairs that connects an
edge\n");
		for(i=0;i<g->e;i++)
		{
			scanf("%d%d",&u,&v);
			g->adj[u][v]=1;
		}
		return g;
	}
	int checkConsistency(graph *g)
	{
		int n,i,u=0,p,j=0;
		struct arrayStack *stack;
		n=g->v;
		stack=createStack(n);
		int a[n];
		for(i=0;i<n;i++)
			a[i]=1;
```

```
                push(stack,u);
                while(!isEmpty(stack))
                {
                        p=pop(stack);
                        a[p]=3;
                        j=0;
                        for(i=0;i<g->v;i++)
                        {
                                if(g->adj[p][i]==1 && (a[i]==2 || a[i]==3))
                                        return 1;
                                else if(g->adj[p][i]==1 && a[i]!=3 &&
a[i]!=2)
                                {
                                        push(stack,i);
                                        a[i]=2;
                                        j=1;
                                }
                        }
                        if(j==0 && p<n-1)
                        push(stack,p+1);
                }
                return 0;
        }
        main()
        {
                int b;
                graph *g;
                g=adjMatrixofgraph();
                b=checkConsistency(g);
                if(b==1)
                        printf("Inconsistent\n");
                else
                        printf("Consistent\n");
        }
```

Test Cases:

Input(n)	Output

Q.4. Compute the minimum number of flight segments to get from one city to another one.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
#define INF 9999
struct graph
{
        int v;
        int e,o,d;
        int **adj;
```

```
};
typedef struct graph graph;
graph* adjMatrixofgraph()
{
        int u,v,i;
        graph *g;
        g=(graph *)malloc(sizeof(graph));
        printf("Enter number of cities and number of flights\n");
        scanf("%d%d",&g->v,&g->e);
        g->adj=(int **)malloc(sizeof(int)*g->v);
        for(i=0;i<g->v;i++)
                g->adj[i]=(int *)malloc(sizeof(int)*g->v);
        for(u=0;u<g->v;u++)
                for(v=0;v<g->v;v++)
                        g->adj[u][v]=INF;
        printf("Enter cities in pairs that are connected by
flights\n");
        for(i=0;i<g->e;i++)
        {
                scanf("%d%d",&u,&v);
                g->adj[u][v]=1;
                g->adj[v][u]=1;
        }
        printf("Enter cities for which you want to find number of
flight segments\n");
        scanf("%d%d",&g->o,&g->d);
        return g;
}
int compute(graph *g)
{
        int i,count=0,min,f[g->v],v[g->v],nc;
        for(i=0;i<g->v;i++)
        {
                f[i]=g->adj[g->o][i];
                v[i]=0;
        }
        f[g->o]=0;
        v[g->o]=1;
        count=1;
        while(count<g->v-1)
        {
                min=INF;
                for(i=0;i<g->v;i++)
                        if(f[i]<min && v[i]!=1)
                        {
                                min=f[i];
```

```
                                        nc=i;
                                }
                        v[nc]=1;
                        for(i=0;i<g->v;i++)
                                if(!v[i])
                                        if((min+g->adj[nc][i])< f[i])
                                                f[i]=min+g->adj[nc][i];
                count++;
                }
                return f[g->d];
}
main()
{
                int b;
                graph *g;
                g=adjMatrixofgraph();
                b=compute(g);
                printf("%d\n",b);
}
```

Test Cases:

Input(n)	Output

Q.5. Goal is to build roads between some pairs of the given cities such that there is a path between any two cities and the total length of the roads is minimized.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
struct graph
{
	int v;
	int e;
	float **adj;
};
float weight(int x1,int y1,int x2,int y2)
{
	int d1,d2;
	float d;
	d1=x1-x2;
	d1=d1*d1;
	d2=y1-y2;
	d2=d2*d2;
	d=sqrt(d1+d2);
	return d;
}
typedef struct graph graph;
graph* adjMatrixofgraph()
{
	int u,v,i;
	graph *g;
	g=(graph *)malloc(sizeof(graph));
	printf("Enter number of nodes\n");
	scanf("%d",&g->v);
	int x[g->v],y[g->v];
	g->adj=(float **)malloc(sizeof(float)*g->v);
	for(i=0;i<g->v;i++)
		g->adj[i]=(float *)malloc(sizeof(float)*g->v);
	printf("Enter coordinates of vertices\n");
```

```
        for(i=0;i<g->v;i++)
                scanf("%d%d",&x[i],&y[i]);
        for(u=0;u<g->v;u++)
                for(v=u+1;v<g->v;v++)
                        g->adj[u][v]=weight(x[u],y[u],x[v],y[v]);

        return g;
}
float shortest(graph *g)
{
        int u,v,flag;
        float sum=0,min=10000;
        for(u=0;u<g->v;u++)
        {
                min=10000;
                flag=0;
                for(v=u+1;v<g->v;v++)
                {
                        if(min>g->adj[u][v])
                        {
                                min=g->adj[u][v];
                                flag=1;
                        }
                }
                if(flag==1)
                sum=sum+min;
        }
        return sum;
}
main()
{
        graph *g;
        float s;
        g=adjMatrixofgraph();
        s=shortest(g);
        printf("%f\n",s);
}
```

Test Cases:

Input(n)	Output

Q.7. Write a program to detect the cycle in the given a directed graph.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct graph
{
        int v;
        int e;
        int **adj;
};
typedef struct graph graph;
struct arrayStack
{
        int top;
        int capacity;
        int *array;
};
struct arrayStack* createStack(int cap)
{
```

```
        struct arrayStack *stack;
        stack=malloc(sizeof(struct arrayStack));
        stack->capacity=cap;
        stack->top=-1;
        stack->array=malloc(sizeof(int)*stack->capacity);
        return stack;
};
int isFull(struct arrayStack *stack)
{
        if(stack->top==stack->capacity-1)
                return 1;
        else
                return 0;
}
int isEmpty(struct arrayStack *stack)
{
        if(stack->top==-1)
                return 1;
        else
                return 0;
}
void push(struct arrayStack *stack,int data)
{
        if(!isFull(stack))
        {
                stack->top++;
                stack->array[stack->top]=data;
        }
        else
        printf("\n Stack overflowed\n");
}
int pop(struct arrayStack *stack)
{
        int item;
        if(!isEmpty(stack))
        {
                item=stack->array[stack->top];
                stack->top--;
                return item;
        }
        return -1;
}
graph* adjMatrixofgraph()
{
        int u,v,i;
        graph *g;
```

```
		g=(graph *)malloc(sizeof(graph));
		printf("Enter number of nodes and number of edges\n");
		scanf("%d%d",&g->v,&g->e);
		g->adj=(int **)malloc(sizeof(int)*g->v);
		for(i=0;i<g->v;i++)
			g->adj[i]=(int *)malloc(sizeof(int)*g->v);
		for(u=0;u<g->v;u++)
			for(v=0;v<g->v;v++)
				g->adj[u][v]=0;
		printf("Enter node number in pairs that connects an
edge\n");
		for(i=0;i<g->e;i++)
		{
			scanf("%d%d",&u,&v);
			g->adj[u][v]=1;
		}
		return g;
	}
	int check(graph *g)
	{
		int n,i,u=0,p,j=0;
		struct arrayStack *stack;
		n=g->v;
		stack=createStack(n);
		int a[n];
		for(i=0;i<n;i++)
			a[i]=1;
		push(stack,u);
		while(!isEmpty(stack))
		{
			p=pop(stack);
			a[p]=3;
			j=0;
			for(i=0;i<g->v;i++)
			{
				if(g->adj[p][i]==1 && (a[i]==2 || a[i]==3))
					return 1;
				else if(g->adj[p][i]==1 && a[i]!=3 &&
a[i]!=2)
				{
					push(stack,i);
					a[i]=2;
					j=1;
				}
			}
			if(j==0 && p<n-1)
```

```
                push(stack,p+1);
        }
        return 0;
}
main()
{
        int b;
        graph *g;
        g=adjMatrixofgraph();
        b=check(g);
        if(b==1)
                printf("True\n");
        else
                printf("False\n");
}
```

Test Cases:

Input(n)	Output

Q.8. Write a program to find the given graph is a connected graph or a disconnected graph.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct graph
{
        int v;
        int e;
        int **adj;
};
typedef struct graph graph;
graph* adjMatrixofgraph()
{
        int u,v,i;
        graph *g;
        g=(graph *)malloc(sizeof(graph));
        printf("Enter number of nodes and number of edges\n");
        scanf("%d%d",&g->v,&g->e);
        g->adj=(int **)malloc(sizeof(int)*g->v);
        for(i=0;i<g->v;i++)
                g->adj[i]=(int *)malloc(sizeof(int)*g->v);
        for(u=0;u<g->v;u++)
                for(v=0;v<g->v;v++)
                        g->adj[u][v]=0;
        printf("Enter node number in pairs that connects an
edge\n");
        for(i=0;i<g->e;i++)
        {
                scanf("%d%d",&u,&v);
                g->adj[u][v]=1;
                g->adj[v][u]=1;
        }
        return g;
}
int connect(graph *g)
{
        int u,v,count;
```

```
        for(u=0;u<g->v;u++)
        {
        count=0;
                for(v=0;v<g->v;v++)
                        if(g->adj[u][v]==0)
                                count++;
        if(count==g->v || g->e < g->v-1)
                return 0;
        else
                return 1;
        }
}
main()
{
        int b;
        graph *g;
        g=adjMatrixofgraph();
        b=connect(g);
        if(b==1)
                printf("True\n");
        else
                printf("False\n");
}
```

Test Cases:

Input(n)	Output

Q.9. Write a program BFS and DFS graph traversal algorithm for the given directed and undirected graph.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct graph
{
        int v;
        int e;
        int **adj;
};
typedef struct graph graph;
struct arrayQueue
{
        int front,rear;
        int capacity;
        int *array;
};
struct arrayQueue* createQueue(int capacity)
{
        struct arrayQueue *q;
        q = malloc(sizeof(struct arrayQueue));
        q->capacity=capacity;
        q->rear=q->front=-1;
        q->array=malloc(q->capacity*(sizeof(int)));
        return q;
}
int isEmptyQueue(struct arrayQueue *q)
{
        return (q->front==-1);
}
int isFullQueue(struct arrayQueue *q)
```

```
{
        return((q->rear+1)%q->capacity==q->front);
}
void enQueue(struct arrayQueue *q,int data)
{
        if(isFullQueue(q))
                printf("Queue overflow\n");
        else
        {
                q->rear=(q->rear+1)%q->capacity;
                q->array[q->rear]=data;
                if(q->front==-1)
                        q->front=q->rear;
        }
}
int deQueue(struct arrayQueue *q)
{
        int data=-1;
        if(isEmptyQueue(q))
        {
                printf("Queue is empty\n");
                return(-1);
        }
        else
        {
                data=q->array[q->front];
                if(q->front==q->rear)
                        q->front=q->rear=-1;
                else
                        q->front=(q->front+1)%q->capacity;
        }
        return data;
}
graph* adjMatrixofgraph()
{
        int flag=0,u,v,i;
        graph *g;
        g=(graph *)malloc(sizeof(graph));
        printf("Enter number of nodes and number of edges\n");
        scanf("%d%d",&g->v,&g->e);
        g->adj=(int **)malloc(sizeof(int)*g->v);
        for(i=0;i<g->v;i++)
                g->adj[i]=(int *)malloc(sizeof(int)*g->v);
        for(u=0;u<g->v;u++)
                for(v=0;v<g->v;v++)
                        g->adj[u][v]=0;
```

```
		printf("If graph is directed press 1\n");
		scanf("%d",&flag);
		printf("Enter node number in pairs that connects an
edge\n");
		for(i=0;i<g->e;i++)
		{
			scanf("%d%d",&u,&v);
			g->adj[u][v]=1;
			if(flag!=1)
			g->adj[v][u]=1;
		}
		return g;
	}
	void bfs(graph *g)
	{
		int n,i,u=0,p,j,flag;
		struct arrayQueue *queue;
		n=g->v;
		queue=createQueue(n);
		int a[n];
		for(i=0;i<n;i++)
			a[i]=1;
		enQueue(queue,u);
		while(!isEmptyQueue(queue))
		{
			p=deQueue(queue);
			flag=0;
			printf("%d, ",p);
			a[p]=3;
			for(i=0;i<g->v;i++)
			{
				if(g->adj[p][i]==1 && a[i]!=3 && a[i]!=2)
				{
					enQueue(queue,i);
					a[i]=2;
					flag=1;
				}
			}
			if(flag==0 && p<n-1 && a[p+1]==1)
					enQueue(queue,p+1);
		}
	}
	main()
	{
		graph *g;
		g=adjMatrixofgraph();
```

```
        bfs(g);
}
```

Test Cases:

Input(n)	Output

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct graph
{
        int v;
        int e;
        int **adj;
};
typedef struct graph graph;
struct arrayStack
{
```

```
        int top;
        int capacity;
        int *array;
};
struct arrayStack* createStack(int cap)
{
        struct arrayStack *stack;
        stack=malloc(sizeof(struct arrayStack));
        stack->capacity=cap;
        stack->top=-1;
        stack->array=malloc(sizeof(int)*stack->capacity);
        return stack;
};
int isFull(struct arrayStack *stack)
{
        if(stack->top==stack->capacity-1)
                return 1;
        else
                return 0;
}
int isEmpty(struct arrayStack *stack)
{
        if(stack->top==-1)
                return 1;
        else
                return 0;
}
void push(struct arrayStack *stack,int data)
{
        if(!isFull(stack))
        {
                stack->top++;
                stack->array[stack->top]=data;
        }
        else
        printf("\n Stack overflowed\n");
}
int pop(struct arrayStack *stack)
{
        int item;
        if(!isEmpty(stack))
        {
                item=stack->array[stack->top];
                stack->top--;
                return item;
        }
```

```
		return -1;
	}
	graph* adjMatrixofgraph()
	{
		int u,v,i,flag=0;
		graph *g;
		g=(graph *)malloc(sizeof(graph));
		printf("Enter number of nodes and number of edges\n");
		scanf("%d%d",&g->v,&g->e);
		printf("If graph is directed press 1\n");
		scanf("%d",&flag);
		g->adj=(int **)malloc(sizeof(int)*g->v);
		for(i=0;i<g->v;i++)
			g->adj[i]=(int *)malloc(sizeof(int)*g->v);
		for(u=0;u<g->v;u++)
			for(v=0;v<g->v;v++)
				g->adj[u][v]=0;
		printf("Enter node number in pairs that connects an
edge\n");
		for(i=0;i<g->e;i++)
		{
			scanf("%d%d",&u,&v);
			g->adj[u][v]=1;
			if(flag!=1)
			g->adj[v][u]=1;
		}
		return g;
	}
	void dfs(graph *g)
	{
		int n,i,u=0,p,flag;
		struct arrayStack *stack;
		n=g->v;
		stack=createStack(n);
		int a[n];
		for(i=0;i<n;i++)
			a[i]=1;
		push(stack,u);
		while(!isEmpty(stack))
		{
			p=pop(stack);
			printf("%d, ",p);
			flag=0;
			a[p]=3;
			for(i=0;i<g->v;i++)
			{
```

```
                        if(g->adj[p][i]==1 && a[i]!=3 && a[i]!=2)
                        {
                                push(stack,i);
                                a[i]=2;
                                flag=1;
                        }
                }
                if(flag==0 && p<n-1 && a[p+1]==1)
                        push(stack,p+1);
        }
}
main()
{
        graph *g;
        g=adjMatrixofgraph();
        dfs(g);
}
```

Test Cases:

Input(n)	Output

Q.10. Write a program to find a Mother Vertex in a Graph.

Concept/Pseudocode:

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct graph
{
	int v;
	int e;
	int **adj;
};
typedef struct graph graph;
graph* adjMatrixofgraph()
{
	int u,v,i;
	graph *g;
	g=(graph *)malloc(sizeof(graph));
	printf("Enter number of nodes and number of edges\n");
	scanf("%d%d",&g->v,&g->e);
	g->adj=(int **)malloc(sizeof(int)*g->v);
	for(i=0;i<g->v;i++)
		g->adj[i]=(int *)malloc(sizeof(int)*g->v);
	for(u=0;u<g->v;u++)
		for(v=0;v<g->v;v++)
			g->adj[u][v]=0;
	printf("Enter node number in pairs that connects an edge\n");
	for(i=0;i<g->e;i++)
	{
		scanf("%d%d",&u,&v);
```

```
                g->adj[u][v]=1;
                g->adj[v][u]=1;
        }
        return g;
}
int motherVertex(graph *g)
{
        int i,j,count=0,max=-1,mother,k;
        for(i=0;i<g->v;i++)
        {
        count=0;
        k=0;
        for(j=0;j<g->v;j++)
                {
                        if(g->adj[i][j]==1)
                                count++;
                        else if(g->adj[i][j]==0)
                                k++;
                }
        if(k==g->v)
                return -1;
        if(count>max)
                mother=i;
        }
        return mother;
}
main()
{
        int b;
        graph *g;
        g=adjMatrixofgraph();
        b=motherVertex(g);
        printf("%d\n",b);
}
```

Test Cases:

Input(n)	Output