1) Write a C program to accept and sort n elements in ascending order by using bubble sort. Non-recursive

Program :

```
// Write a C program to accept and sort n elememts in ascending order by using bubble sort.
//Complexity of Bubble sort is Worst case = Average case = Best case = 0(n^2)
#include<stdio.h>
//void bubble_sort(int a[100],int n);
 int main()
{
        int i,j,n,a[100],temp;

        printf("How many numbers you wants to enter : ");
        scanf("%d",&n);

        printf("\nEnter a element \n");
        for (i=0; i<n; i++)
                scanf("%d",&a[i]);

        //bubble_sort(a,n);

        or (i=0; i<n-1; i++)
                for (j=0; j<n-i-1; j++)
                        if (a[j] > a[j+1])
                        {
                                temp = a[j];
                                a[j] = a[j+1];
                                a[j+1] = temp;
                        }

        printf("After bubble sorting\n");
        for (i=0; i<n; i++)
                printf("%d\n",a[i]);

        return 1;
}

/*void bubble_sort(int a[100],int n)
{
        int i,j,temp;
        for (i=0; i<n-1; i++)
                for (j=0; j<n-i-1; j++)
                        if (a[j] > a[j+1])
                        {
                                temp = a[j];
                                a[j] = a[j+1];
                                a[j+1] = temp;
                        }
}*/
```

2) Write a C program to accept and sort n elements in ascending order by using insertion sort. Non-recursive

Program :

```
//Write a C program to accept and sort n elememts in ascending order by using insertion sort.
//Complexity of Bubble sort is Worst case = Average case = O(n^2), Best case = O(n)
#include<stdio.h>
//void insertion_sort(int a[100], int n);
 int main()
{
        int a[100],i,j,k,temp,n,p;

        printf("How many numbers you wants to enter : ");
        scanf("%d",&n);

        printf("\nEnter a element \n");
        for (i=0; i<n; i++)
                scanf("%d",&a[i]);

        //insertion_sort(a,n);

        for (i=0; i<n; i++)
                printf("%d\n",a[i]);


        for (i=1; i<n; i++)
        {
                temp = a[i];
                for (j=i-1; j>=0; j--)
                {
                        if (a[j] > temp)
                                a[j+1] = a[j];
                        else
                                break;
                }
                a[j+1] = temp;
        }

        printf("\nAfter insertion sort \n");
        for (i=0; i<n; i++)
                printf("%d\n",a[i]);

        return 1;
}

/*void insertion_sort(int a[100], int n)
{
        int i,j,temp;

        for (i=1; i<n; i++)
        {
                temp = a[i];
                for (j=i-1; j>=0; j--)
                {
                        if (a[j] > temp)
                                a[j+1] = a[j];
                        else
                                break;
                }
                a[j+1] = temp;
        }
}*/
```

3) Write a program in C to accept 5 numbers from the user and sort the numbers in ascending order by using Merge sort. recursive

Program :

```
//Write a program in C to accept 5 numbers from the user and sort the numbers in ascending order by using Merge sort.

//It is working on divide and conquer strategy.
//Generally this sort want's only two function.
//time :- Best Case = Worst Case = 0(n log n).
//It is stable sorting process.
#include<stdio.h>
void merge(int a[100],int lb,int mid,int ub)
{
	int i=lb,j=mid+1,k=lb,b[100];

	while (i<=mid && j<=ub)
	{
		if(a[i] <= a[j])
			b[k++] = a[i++];
		else
			b[k++] = a[j++];
	}
	while (i<=mid)
		b[k++] = a[i++];

	while (j<=ub)
		b[k++] = a[j++];

	for (k=lb; k<=ub;k++)
		a[k] = b[k];
}

void merge_sort(int a[100],int lb,int ub)
{
	int mid;
	if (lb<ub)
	{
		mid = (lb+ub)/2;
		merge_sort(a,lb,mid);
		merge_sort(a,mid+1,ub);
		merge(a,lb,mid,ub);
	}
}

void main()
{
	int a[100],n,i;

	printf("Enter 5 elememts \n");
```

```
        for (i=0; i<5; i++)
                scanf("%d",&a[i]);

        merge_sort (a,0,5-1);

        printf("\nAfter sorting \n");
        for (i=0; i<5; i++)
                printf("%d\n",a[i]);
}
```

4) Write a C program to sort a random array of n integers by using Merge Sort algorithm in ascending order. Recursive

Program :

```
//Write a C program to sort a random array of n integers by using Merge Sort algorithm in ascending order.

//It is working on divide and conquer strategy.
//Generally this sort want's only two function.
//time :- Best Case = Worst Case = 0(n log n).
//It is stable sorting process.
#include<stdio.h>
#include<time.h>
void gen(int a[100],int n)
{
	int i;
	srand(time(0));
	for (i=0; i<n; i++)
		a[i] = rand()%100;
}
void merge(int a[100],int lb,int mid,int ub)
{
	int i=lb,j=mid+1,k=lb,b[100];

	while (i<=mid && j<=ub)
	{
		if(a[i] <= a[j])
			b[k++] = a[i++];
		else
			b[k++] = a[j++];
	}
	while (i<=mid)
		b[k++] = a[i++];

	while (j<=ub)
		b[k++] = a[j++];

	for (k=lb; k<=ub;k++)
		a[k] = b[k];
}

void merge_sort(int a[100],int lb,int ub)
{
	int mid;
	if (lb<ub)
	{
		mid = (lb+ub)/2;
		merge_sort(a,lb,mid);
		merge_sort(a,mid+1,ub);
		merge(a,lb,mid,ub);
	}
}

void main()
{
    int a[100],n,i;

    printf("How many numbers you wants in array to sort : ");
    scanf("%d",&n);

    gen(a,n);

    merge_sort (a,0,n-1);

    printf("\nAfter sorting \n");
    for (i=0; i<n; i++)
        printf("%d\n",a[i]);
}
```

5) Write a C program to accept n elements from user store it in an array. Accept a value from the user and use Non- recursive binary search method to check whether the value is present in array or not. Display proper message in output.

Program :

```
//Write a C program to accept n elements from user store it in an array. Accept a value from the user and use Non- recursive binary search method to check whether the value is present in array or not. Display proper message in output.

//Input of binary search is must be ascending or descending order.
//Time Complexity of Warst case is O(log n) and Best case is O(1).

#include<stdio.h>

void bubble_sort(int a[],int n)
{
	int i,j,temp;

	for (i=1; i<n; i++)
		for (j=0; j<n-i; j++)
			if (a[j+1] < a[j])
			{
				temp = a[j+1];
				a[j+1] = a[j];
				a[j] = temp;
			}

}

void binary_search (int a[100], int low, int high, int num)
{
	int mid;

	while (low <= high)
	{
		mid = (low+high)/2;
		if (a[mid] == num)
		{
			printf("Element is found at %d position...\n",mid+1);
			break;
		}

		else if (a[mid] > num)
			high = mid-1;

		else
			low = mid+1;
	}
	if (low > high)
		printf("Element not found\n");
}
 void main()
{
        int a[100],n,i,num;
        printf("How many elements you want's to enter : ");
        scanf("%d",&n);

        printf("\nNOTE : Number is must be greater than previous number.\n");

        printf("Enter elements \n");

        for (i=0; i<n; i++)
        {
                scanf("%d",a[i]);
                if (i > 0 && a[i-1] >= a[i])
                {
                        printf("Enter larger number than %d.\n",a[i-1]);
                        i--;
                }
        }

        printf("Enter a element for search : ");
        scanf("%d",&num);

        bubble_sort(a,n);
        binary_search(a,0,n-1,num);
}
```

6) Write a C program to accept n elements from user store it in an array. Accept a value from the user and use recursive binary search method to check whether the value is present in array or not.

Program :

```
//Write a C program to accept n elements from user store it in an array. Accept a value from the user and use recursive binary search method to check whether the value is present in array or not.
//For binary search input must be in asceding or descending order.

#include<stdio.h>

void bubble_sort(int a[], int n)
{
	int i,j,temp;

	for (i=1; i<n; i++)
		for (j=0; j<n-i; j++)
			if (a[j] > a[j+1])
			{
				temp = a[j];
				a[j] = a[j+1];
				a[j+1] = temp;
			}

	for (i=0; i<n ;i++)
		printf("%d \n",a[i]);

}

void binary_search(int a[], int low, int high, int num)
{
	int mid;

	mid = (low+high)/2;
	if(low<=high)
	{
		if(a[mid] == num)
		{
			printf("Element is found at %d position\n",mid+1);
			return;
		}

		else if (a[mid] > num)
			binary_search(a,low,mid-1,num);
		else
			binary_search(a,mid+1,high,num);
	}
	else
	{
		printf("Element is not found...");
		return;
```

```
        }

}

 void main()
{
        int a[100],i,num,n;

        printf("How many elements you wants to enter : ");
        scanf("%d",&n);

        printf("\nNOTE : Number is must be greater than previous number.\n");

        printf("Enter elements \n");

        for (i=0; i<n; i++)
        {
                scanf("%d",a[i]);
                if (i > 0 && a[i-1] >= a[i])
                {
                        printf("Enter larger number than %d.\n",a[i-1]);
                        i--;
                }
        }

        printf("Enter a element for search : ");
        scanf("%d",&num);

        bubble_sort(a,n);
        binary_search(a,0,n,num);
}
```

7) Write a C program to implement a Singly linked list with following operations create() , display(),insert(),delete()

Program :

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

void create(node **r)
{
	int n,i;
	node *temp,*newnode;
	printf("How many nodes u wants to enter :");
	scanf("%d",&n);

	for (i = 0; i < n; i++)
	{
		newnode = (node *)malloc(sizeof(node));
		scanf("%d",&newnode->data);
		if (*r == NULL)
		{
			*r = newnode;
			temp = newnode;
		}
		else
		{
			temp->next = newnode;
			temp = temp->next;
		}
	}
}

void display(node *r)
{
	while (r!=NULL)
	{
		printf("%d\n",r->data);
		r = r->next;
	}
}

int count(node *r)
{
	int cnt=0;

	while (r!=NULL)
	{
		cnt++;
		r = r->next;
	}

	return cnt;
}


void insert(node **r)
{
	int n,i;
	node *newnode,*temp = *r;
	printf("Enter a position to insert : ");
	scanf("%d",&n);

	newnode = (node *)malloc(sizeof(node));
	scanf("%d",&newnode->data);
	newnode->next = NULL;

	if (n<0 || count(*r)+1 < n )
		printf("Position invalid\n");
	else if(n==1)
	{
		newnode->next = *r;
		*r = newnode;
	}
	else if (n==count(*r))
	{
		while (temp->next->next!=NULL)
		{
			temp = temp->next;
		}

		newnode->next = temp->next;
		temp->next = newnode;
	}
	else if(n == count(*r)+1)
	{
		while (temp->next!=NULL)
		{
			temp = temp->next;
		}
		temp->next = newnode;
	}

	else
	{
		for (i = 1; i < n-1; i++)
		{
			temp = temp->next;
		}
		newnode->next = temp->next;
		temp->next = newnode;
	}
}

void delete(node **r)
{
	int n,i;
	node *temp,*del;

	printf("Enter a position to delete : ");
	scanf("%d",&n);

	if(n<0 || n>count(*r))
		printf("Invalid position\n");
	else if(n==1)
	{
		temp=*r;
		*r = (*r)->next;
		temp->next=NULL;
		free(temp);
	}
	else if(n==count(*r))
	{
		temp = *r;
		while (temp->next->next!=NULL)
		{
			temp = temp->next;
		}
		del = temp->next;
		temp->next=NULL;
		del->next=NULL;
		free(del);
	}
	else
	{
		temp = *r;
		for (i = 1; i < n-1; i++)
		{
			temp = temp->next;
		}
		del = temp->next;
		temp->next = del->next;
		del->next=NULL;
		free(del);
	}
}

void main (int argc, char *argv[])
{
	int ch;
	node *root=NULL;
```

```
        printf("Create a LL : ");
        create(&root);

        while(1)
        {
                printf("1. Insert\n2. Delete\n3. Display\4.Exit\nEnter a choice");
                scanf("%d",&ch);

                switch(ch)
                {
                        case 1 : insert(&root);
                                 break;
                        case 2 : delete(&root);
                                 break;
                        case 3 : display(root);
                                 break;
                        case 4 : return;
                        default : printf("Invalid choice\n");
                }
        }
}
```

8) Write a C program to implement a Singly Circular linked list with following operations create(), display(),search(),length()

Program :

```
//Write a C program to implement a Singly Circular linked list with following operations create(),
display(), search(),length().

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

typedef struct node
{
	int data;
	struct node *next;
}node;

void create(node **r)
{
	int n,i;
	node *temp;

	printf("How many nodes you wants to insert : ");
	scanf("%d",&n);
	if ( n == 0)
		return;

	printf("Enter data\n");
	for (i=1, *r=temp=(node *)malloc(sizeof(node)); i<n; i++,temp = temp->next)
	{
		scanf("%d",&temp->data);
		temp->next = (node *)malloc(sizeof(node));
	}
	scanf("%d",&temp->data);
	temp->next = *r;
	*r = temp;
}

void display(node *r)
{
	node *temp = r;

	if (r == NULL)
	{
		printf("List is Empty");
		return;
	}

	do
	{
		printf("%d\n",temp->next->data);
		temp = temp->next;
```

```
        }while(temp!=r);

}

void search(node *r)
{
        int num,i;
        node *temp = r;

        printf("Enter a data to search : ");
        scanf("%d",&num);

        do
        {
                if(temp->next->data == num)
                {
                        printf("Data is found \n");
                        return;
                }
                temp = temp->next;
        }while (temp!=r);

        printf("Data is not found\n");
}


void length(node *r)
{
        node *temp = r;
        int cnt=0;

        do
        {
                cnt++;
                temp = temp->next;
        }while (temp!=r);

        printf("Length of linked list is %d\n",cnt);

}

 void main()
{
        node *root=NULL;
        int ch;

        create(&root);

        while(1)
        {
                printf("1. Display\n2. Search\n3. Length\n4. Exit\nEnter choice : ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1 : display(root);
                                 break;
                        case 2 : search(root);
                                 break;
                        case 3 : length(root);
                                 break;
                        case 4 : printf("Thank You!...\n");
                                 return 1;
                        default : printf("Invalid choice\n");
                }
        }

}
```

9) Write a C program to implement a Doubly linked list with following operations create() , display(), insert(),delete()

Program :

```
//Write a C program to implement a Doubly linked list with following operations create() ,
display(), insert(),delete().

//NOTE : Instead of n you can count() for counting a nodes in DLL and validation of of position.

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

typedef struct node
{
        int data;
        struct node *pre , *next;
}node;

/*int count(node *r)
{
        int cnt=0;
        while (r!=NULL)
        {
                cnt++;
                r = r->next;
        }

        return cnt;
}*/

int  create (node **r)
{
        int n,i;
        node *temp;
        printf("How many node you want's to insert : ");
        scanf("%d",&n);
        if(n == 0)
                return 0;

        printf("Enter data : \n");
        for (i=1,*r=temp = (node *)malloc(sizeof(node)); i<n; i++,temp = temp->next)
        {
                scanf("%d",&temp->data);
                temp->next = (node *)malloc(sizeof(node));
                temp->next->pre = temp;
        }
        scanf("%d",&temp->data);
        temp->next=NULL;
        (*r)->pre=NULL;

        return n;
```

```
}

void display(node *r)
{
    if (r==NULL)
    {
        printf("List is Empty...\n");
        return;
    }

    while(r!=NULL)
    {
        printf("%d\n",r->data);
        r = r->next;
    }

}

int insert(node **r, int n)
{
    int pos,i;
    node *temp=*r,*new=NULL;
    printf("Enter a position : ");
    scanf("%d",&pos);

    if (pos<=0 || pos>n+1)
    {
        printf("Position invalid...\n");
        return n;
    }
    new = (node *)malloc(sizeof(node));
    printf("Enter a data : ");
    scanf("%d",&new->data);
    if (pos == 1 && (*r)==NULL)
    {
        (*r) = new;
        (*r)->next = NULL;
        (*r)->pre = NULL;

        return n+1;
    }
    if(pos == 1)
    {
        new->next = *r;
        new->next->pre = new;
        new->pre=NULL;
        *r=new;
        return n+1;
    }
    if(pos == n+1)
    {
        while(temp->next!=NULL)
```

```
                        temp = temp->next;

                temp->next = new;
                new->pre=temp;
                new->next=NULL;
                return n+1;
        }
        for (i=1;i<pos-1;i++,temp=temp->next);

        new->next = temp->next;
        new->pre=temp;
        temp->next=new;

        return n+1;
}

int delete(node **r,int n)
{
        int i,pos;
        node *temp=*r,*del=NULL;

        printf("Enter a position : ");
        scanf("%d",&pos);
        if((*r) == NULL)
        {
                printf("List is Empty...\n");
                return 0;
        }
        if (pos<=0 || pos > n )
        {
                printf("Position Invalid...\n");
                return n;
        }
        if (pos == 1 && temp->next == NULL)
        {
                free(*r);
                *r = NULL;

                return n-1;
        }
        if (pos == 1)
        {
                (*r) = temp->next;
                (*r)->pre = NULL;
                temp->next = NULL;
                temp->pre=NULL;
                free(temp);

                return n-1;

        }
        if (pos == n)
        {
                while(temp->next->next!=NULL)
                        temp = temp->next;

                temp->next->pre=NULL;
                temp->next=NULL;

                return n-1;
        }

        for(i=1;i<pos-1; i++,temp=temp->next);

        del = temp->next;
        temp->next = del->next;
        del->next->pre = temp;
        del->pre = NULL;
        del->next =NULL;
        free(del);

        return n-1;
}

 void main()
{
        node *root=NULL;
        int ch,n;
        printf("Create a linked list : \n");
        n=create(&root);

        while (1)
        {
                printf("1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter choice : ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1 : n=insert(&root,n);
                                 break;
                        case 2 : n=delete(&root,n);
                                 break;
                        case 3 : display(root);
                                 break;
                        case 4 : printf("Thank You!..\n");
                                 return 1;
                        default : printf("Invalid choice\n");
                }
        }

}
```

10) Write a C program to implement a Doubly Circular linked list with following operations create() and display(), append(),delete()

Program :

```
//Write a C program to implement a Doubly Circular linked list with following operations create()
and display(), append(),delete().

//NOTE : Instead of n you can count() for counting a nodes in CDLL and validation of of position.

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

typedef struct node
{
        int data;
        struct node *pre, *next;
}node;

/*int count(node *r)
{
        int cnt=0;
        node temp=r;

        do
        {
                cnt++;
                temp = temp->next;
        }while(temp!=r);

        return cnt;
}*/

int create(node **r)
{
        int n,i;
        node *temp;

        printf("How many nodes you want's to insert : ");
        scanf("%d",&n);

        if (n == 0)
                return 0;

        printf("Enter a nodes : \n");
        for(i=1, *r=temp=(node *)malloc(sizeof(node)); i<n; i++, temp=temp->next)
        {
                 scanf("%d",&temp->data);
                 temp->next=(node *)malloc(sizeof(node));
                 temp->next->pre = temp;
        }
        scanf("%d",&temp->data);
```

```
    temp->next = *r;
    (*r)->pre = temp;
    //(*r)=temp;

    return n;
}

void display(node *r)
{
    node *temp=r;

    if (r == NULL)
        printf("Link list is Empty\n");
    else
    {
        while (temp->next!=r)
        {
            printf("%d\n",temp->data);
            temp = temp->next;
        }
        printf("%d\n",temp->data);
    }
}

int append(node **r, int n)
{
    node *new;

    new=(node *)malloc(sizeof(node));
    printf("Enter a data : ");
    scanf("%d",&new->data);

    if (*r == NULL)
    {
        (*r) = new;
        new->next=(*r);
        new->pre=(*r);
    }
    else
    {
        new->pre = (*r)->pre;
        new->next = (*r);
        (*r)->pre->next = new;
        (*r)->pre = new;
    }

    return n+1;

}

int delete(node **r,int n)
{
```

```
int i,pos;
node *temp,*del;

printf("Enter a position : ");
scanf("%d",&pos);

if((*r) == NULL)
{
        printf("Link list is Empty\n");
         return n;
}
if(pos<=0 || pos>n)
{
        printf("Position invalid...\n");
        return n;
}
if(pos==1 && (*r)->next == (*r))
{
        (*r)->next=(*r)->pre=NULL;
        free(*r);
        *r=NULL;
        return n-1;

}
if(pos==1)
{
        temp=(*r)->next;
        temp->pre = (*r)->pre;
        (*r)->pre->next = temp;
        (*r)->next=NULL;
        (*r)->pre=NULL;
        free(*r);
        *r = temp;

        return n-1;
}
if (pos==n)
{
        temp=(*r)->pre;
        temp->pre = (*r)->pre;
        temp->pre->next=(*r);
        temp->next=temp->pre=NULL;
        free(temp);

        return n-1;

}

for (i=1,temp=(*r); i<pos-1; i++,temp=temp->next);

del = temp->next;
temp->next=del->next;
```

```
        del->next->pre = temp;
        del->next=NULL;
        del->pre=NULL;
        free(del);

        return n-1;
}

 void main()
{
        node *root=NULL;
        int ch,n;

        printf("Create a linked list : \n");
        n=create(&root);

        while (1)
        {
                printf("1. Append\n2. Delete\n3. Display\n4. Exit\nEnter choice : ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1 : n = append(&root,n);
                                 break;
                        case 2 : n = delete(&root,n);
                                 break;
                        case 3 : display(root);
                                 break;
                        case 4 : printf("Thank You!...\n");
                                 return 1;
                        default : printf("Invalid choice");
                }
        }

}
```

11)Write a C program to implement Static implementation of stack of integers with following operation:
a) Initialize()
b) push()
c) pop()
d) isempty()
e) isfull()
f) display()
g) peek()

Program :

```
/*1) Write a C program to implement Static implementation of stack of integers with following
operation:
a) Initialize()
b) push()
c) pop()
d) isempty()
e) isfull()
f) display()
g) peek()*/


#include <stdio.h>
typedef struct stack
{
	int top;
	int number[100];
}stack;

void initialize(int *top)
{
	*top = -1;
}

void push(stack *s,int num)
{
	if (isfull(s->top))
	{
		printf("Stack is full/overflow...\n");
		return;
	}
	++s->top;
	s->number[s->top] = num;
}

void pop(stack *s)
{
	if (isempty(s->top))
	{
		printf("Stack is empty/underflow...\n");
		return;
```

```
    }
    printf("%d\n",s->number[s->top]);
    s->top--;
}

int isempty(int top)
{
    if (top==-1)
        return 1;
    return 0;
}

int isfull(int top)
{
    if (top==100)
        return 1;
    return 0;
}

void display(stack s)
{
    if (isempty(s.top))
        printf("Stack is empty/underflow...\n");
    else
    {
        int i;
        for(i=s.top; i>-1; i--)
            printf("%d\n",s.number[i]);
    }
}

void peek(stack s)
{
    if (isempty(s.top))
        printf("Stack is empty/underflow...\n");
    else
        printf("%d\n",s.number[s.top]);
}

 void main()
{
    int ch,n;
    stack s;

    initialize(&s.top);
    while (1)
    {
        printf("1. Push\n2. Pop\n3. IsEmpty\n4. IsFull\n5. Display\n6. Peek\n7. Exit\nEnter a
choice : ");
        scanf("%d",&ch);

        switch (ch)
```

```
                {
                        case 1 : printf("Enter a number to push");
                                 scanf("%d",&n);
                                 push(&s,n);
                                 break;
                        case 2 : pop(&s);
                                 break;
                        case 3 : (isempty(s.top) == 1)? printf("Yes\n") : printf("No\n");
                                 break;
                        case 4 : (isfull(s.top) == 1)? printf("Yes\n") : printf("No\n");
                                 break;
                        case 5 : display(s);
                                 break;
                        case 6 : peek(s);
                                 break;
                        case 7 : printf("Thank You!...\n");
                                return 1;
                        default : printf("Invalid Choice...\n");
                }
        }
}
```

12)Write a C program to implement Dynamic implementation of stack of integers with following
operation:
h) Initialize()
i) push()
j) pop()
k) isempty()
l) isfull()
m) display()
n) peek()

Program :

```
/*1) Write a C program to implement Dynamic implementation of stack of integers with following
operation:
a) Initialize()
b) push()
c) pop()
d) isempty()
e) isfull()
f) display()
g) peek()*/

#include <stdio.h>
#include <stdlib.h>
typedef struct stack
{
	int data;
	struct stack *next;
}stack;

int isempty(stack *top)
{
	if (top == NULL)
		return 1;
	return 0;
}

int isfull(stack *temp)
{
	if(temp!=NULL)
		return 0;
	return 1;
}

void initialize(stack **top)
{
	*top = NULL;
}

void push(stack **top,int num)
{
    stack *temp;
    temp = (stack *)malloc(sizeof(stack));
    if (isfull(temp))
    {
        printf("Stack is full/overflow...\n");
        return;
    }

    if (*top == NULL)
    {
        temp->next = NULL;
        temp->data=num;
        *top = temp;
    }
    else
    {
        temp->next = *top;
        temp->data=num;
        *top=temp;
    }
    temp = NULL;
    free(temp);
}

void pop(stack **top)
{
    if (isempty(*top))
        printf("Stack is empty/underflow...\n");
    else
    {
        printf("%d\n",(*top)->data);
        (*top) = (*top)->next;
    }
}


void display(stack *top)
{
    if(isempty(top))
    {
        printf("Stack is empty/underflow...\n");
        return;
    }

    do
    {
        printf("%d\n",top->data);
        top =top->next;
    }while(top!=NULL);
}

void peek(stack *top)
```

```
{
        if(isempty(top))
                printf("Stack is empty/underflow...\n");
        else
                printf("%d\n",top->data);
}

 void main()
{
        int ch,n;
        stack *top;
        initialize(&top);

        while (1)
        {
                printf("1. Push\n2. Pop\n3. IsEmpty\n4. Display\n5. Peek\n6. Exit\nEnter a choice  : 
");
                scanf("%d",&ch);

                switch (ch)
                {
                        case 1 : printf("Enter a number to push : ");
                                 scanf("%d",&n);
                                 push(&top,n);
                                 break;
                        case 2 : pop(&top);
                                 break;
                        case 3 : (isempty(top) == 1)? printf("Yes\n") : printf("No\n");
                                 break;
                        case 4 : display(top);
                                 break;
                        case 5 : peek(top);
                                 break;
                        case 6 : printf("Thank You!...\n");
                                return 1;
                        default : printf("Invalid Choice...\n");
                }
        }

}
```

Following programs are supposed to be solved or written by all.
1. Write a C program to create binary search tree of integers and perform following operations:
• Preordertraversal
• Post ordertraversal

Program :

```
#include<stdio.h>
#include <stdlib.h>
typedef struct node
{
	int data;
	struct node *left,*right;
}node;

void create(node **r)
{
	int n,i;
	node *cur,*newnode,*par;
	printf("How many nodes u wants to enter :");
	scanf("%d",&n);

	for (i = 0; i < n; i++)
	{
		newnode = (node *)malloc(sizeof(node));
		scanf("%d",&newnode->data);
		if (*r == NULL)
		{
			*r = newnode;
			cur = newnode;
			cur->left = cur->right = NULL;
		}
		else
		{
			cur = *r;
			while (cur!=NULL)
			{
				par = cur;
				if (cur->data >= newnode->data)
				{
					cur = cur->left;
				}
				else
				{
					cur = cur->right;
				}
			}
			if (par->data >= newnode->data)
			{
				par->left = newnode;
			}
```

```
                        else
                        {
                                par->right = newnode;
                        }
                }
        }
}

void preorder(node *r)
{
        if (r == NULL)
                return;
        printf("%d ",r->data);
        preorder(r->left);
        preorder(r->right);
}

void postorder(node *r)
{
        if (r == NULL)
                return;
        preorder(r->left);
        preorder(r->right);
        printf("%d ",r->data);
}


void main (int argc, char *argv[])
{

        node *root=NULL;
        printf("Create a BST : ");
        create(&root);

        printf("\nPREORDER : ");
        preorder(root);

        printf("\nPOSTORDER : ");
        postorder(root);
}
```

2. Write a C program to read a graph as adjacency matrix and display the adjacency matrix.

Program :

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

struct AdjListNode
{
     int dest;
     struct AdjListNode* next;
};

// A structure to represent an adjacency list
struct AdjList
{
     struct AdjListNode *head;
};

struct Graph
{
     int V;
     struct AdjList* array;
};

struct AdjListNode* newAdjListNode(int dest)
{
     struct AdjListNode* newNode =
     (struct AdjListNode*) malloc(sizeof(struct AdjListNode));
     newNode->dest = dest;
     newNode->next = NULL;
     return newNode;
}
struct Graph* createGraph(int V)
{
     struct Graph* graph =
     (struct Graph*) malloc(sizeof(struct Graph));
     graph->V = V;

     graph->array =
     (struct AdjList*) malloc(V * sizeof(struct AdjList));

     int i;
     for (i = 0; i < V; ++i)
     graph->array[i].head = NULL;
     return graph;
}
void addEdge(struct Graph* graph, int src, int dest)
{
     struct AdjListNode* newNode = newAdjListNode(dest);
     newNode->next = graph->array[src].head;
     graph->array[src].head = newNode;
     newNode = newAdjListNode(src);
```

```
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}

void printGraph(struct Graph* graph)
{
    int v;
    for (v = 0; v < graph->V; ++v)
    {
        struct AdjListNode* pCrawl = graph->array[v].head;
        printf("\n Adjacency list of vertex %d\n head ", v);
        while (pCrawl)
        {
            printf("-> %d", pCrawl->dest);
            pCrawl = pCrawl->next;
        }
        printf("\n");
    }
}

void main()
{
    int V = 5;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);
    printGraph(graph);
}
```

3. Add a function in Q2 (above question) to count total degree, indegree and outdegree of the graph.

Program :

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

  • void main()

  • {

  •    int option;

  •    do

  •    {

  •         printf("\n A Program to represent a Graph by using an ");

  •         printf("Adjacency Matrix method \n ");

  •         printf("\n 1. Directed Graph ");

  •         printf("\n 2. Un-Directed Graph ");

  •         printf("\n 3. Exit ");

  •         printf("\n\n Select a proper option : ");

  •         scanf("%d", &option);

  •         switch(option)

  •         {

  •             case 1 : dir_graph();

  •                      break;

  •             case 2 : undir_graph();

  •                      break;

  •             case 3 : exit(0);

  •         } // switch

  •     }while(1);

  • }

  • int dir_graph()

  • {
```

```
•     int adj_mat[50][50];
•     int n;
•     int in_deg, out_deg, i, j;
•     printf("\n How Many Vertices ? : ");
•     scanf("%d", &n);
•     read_graph(adj_mat, n);
•     printf("\n Vertex \t In_Degree \t Out_Degree \t Total_Degree ");
•     for (i = 1; i <= n ; i++ )
•     {
•         in_deg = out_deg = 0;
•         for ( j = 1 ; j <= n ; j++ )
•         {
•             if ( adj_mat[j][i] == 1 )
•                 in_deg++;
•         }
•         for ( j = 1 ; j <= n ; j++ )
•             if (adj_mat[i][j] == 1 )
•                 out_deg++;
•             printf("\n\n %5d\t\t\t%d\t\t%d\t\t%d\n\
  n",i,in_deg,out_deg,in_deg+out_deg);
•     }
•     return;
• }
•
• int undir_graph()
• {
•     int adj_mat[50][50];
•     int deg, i, j, n;
•     printf("\n How Many Vertices ? : ");
•     scanf("%d", &n);
•     read_graph(adj_mat, n);
•     printf("\n Vertex \t Degree ");
•     for ( i = 1 ; i <= n ; i++ )
•     {
•         deg = 0;
•         for ( j = 1 ; j <= n ; j++ )
•             if ( adj_mat[i][j] == 1)
•                 deg++;
•         printf("\n\n %5d \t\t %d\n\n", i, deg);
•     }
•     return;
• }
•
• int read_graph ( int adj_mat[50][50], int n )
• {
•     int i, j;
•     char reply;
•     for ( i = 1 ; i <= n ; i++ )
•     {
•         for ( j = 1 ; j <= n ; j++ )
•         {
•             if ( i == j )
•             {
•                 adj_mat[i][j] = 0;
•                 continue;
•             }
•             printf("\n Vertices %d & %d are Adjacent ? (Y/N) :",i,j);
•             scanf("%c", &reply);
```

```
            if ( reply == 'y' || reply == 'Y' )
                adj_mat[i][j] = 1;
            else
                adj_mat[i][j] = 0;
        }
    }
    return;
}
```