Experiment no 1.

**Program :**

    #include <stdio.h>

    int main() {

        int a[20],elem,again,n,i,choice,j,temp;

        printf("Enter number of elements :");

        scanf("%d",&n);

        printf("Enter the elements :\n");

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

        do{

            printf("Enter your choice :\n1. search\t2. update\t3. insert\t4. delete\t5. sort in ascending order\t6. sort in decending order\n");

            scanf("%d",&choice);

            switch (choice)

            {

            case 1:

                printf("Enter the element to be searched :");

                scanf("%d",&elem);

                for(i=0;i<n;i++)

                {

                    if(a[i]==elem)

                    {

                        printf("Index of required element is %d\n",i);

                        break;

                    }

                }

                if(i==n) printf("Not found\n");

                break;

            case 2:

                printf("Enter the index of element to be updated :");

                scanf("%d",&i);

                printf("update %d with :",a[i]);

                scanf("%d",&elem);

                a[i]=elem;

                break;

            case 3:

                printf("Enter the index where element to be inserted :");

                scanf("%d",&j);

                printf("Enter the element to be inserted :");

                scanf("%d",&elem);

                for ( i = ++n; i > j; i--)

                {

                    temp=a[i];

                    a[i]=a[i-1];

                    a[i-1]=temp;

                }

                a[j]=elem;

                break;

            case 4:

                printf("Enter the index of element to be deleted :");

                scanf("%d",&j);

                n--;

                for ( i = j; i < n; i++)

                {

                    temp=a[i];

                    a[i]=a[i+1];

                    a[i+1]=temp;

                }

                break;

            case 5:

                printf("Sorting arrray in ascending order\n");

                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;

                        }

                    }

                }

                break;

            case 6:

                printf("Sorting arrray in decending order\n");

                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;

                        }

                    }

                }

                break;

            default:

                printf("Invalid Choice !\n");

            }

        printf("\nNew array : ");

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

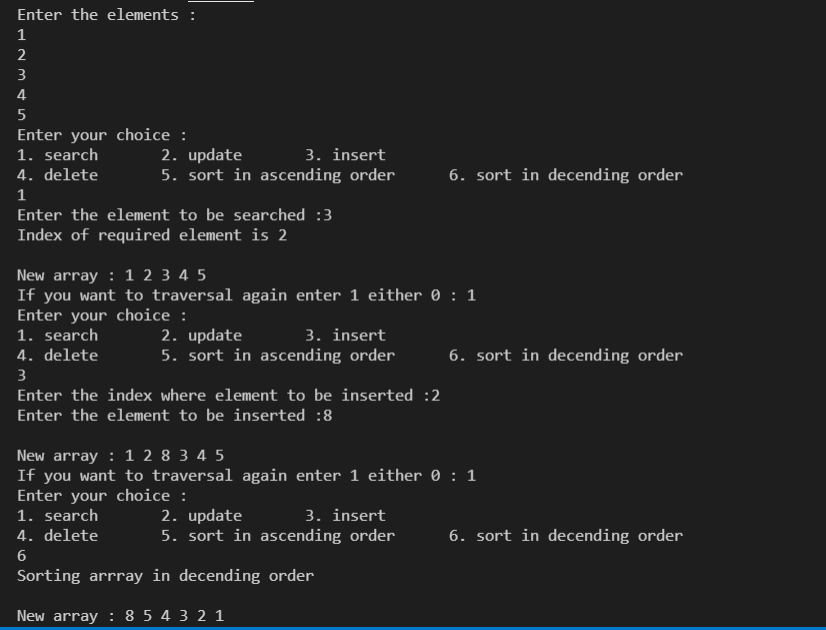
        printf("\nIf you want to traversal again enter 1 either 0 : ");

        scanf("%d",&again);

        }while(again==1);

    }

**Output :**



Experiment no 2.

**Program :**

#include <stdio.h>

#define MAX 100

    char stack[MAX];

    int top = -1,i,x;

    void push(char);

    char pop(void);

    int priority(char);

    void infixToPostfix(char exp[])

    {

        char postfix[100];

        int i=0,j=0;

        while(exp[i]!=0)

        {

            if((exp[i] >= 'a' && exp[i] <= 'z') || (exp[i] >= 'A' && exp[i] <= 'Z') || (exp[i] >= '0' && exp[i] <= '9'))

            {

                postfix[j]=exp[i];

                j++;

            }

            else if(exp[i]=='(')

            {

                push(exp[i]);

            }

            else if(exp[i]==')')

            {

                while((x = pop()) != '(')

                    {

                        postfix[j]=x;

                        j++;

                    }

            }

            else {

                while(priority(exp[i]) <= priority(stack[top])) {

                    postfix[j]=pop();

                    j++;

                }

                push(exp[i]);

            }

            i++;

        }

        while(top != -1) {

            postfix[j]=pop();

            j++;

        }

        postfix[j]=0;

        printf("The postfix of given expression is : %s\n",postfix);

    }

    int main()

    {

        char exp[100],postfix[100];

        printf("Enter the Infix Expression :");

        gets(exp);

        infixToPostfix(exp);

        return 0;

    }

    int priority(char c)

    {

        if(c=='^'||c=='$') return 3;

        else if(c=='\*'||c=='/') return 2;

        else if(c=='+'||c=='-') return 1;

        else return 0;

    }

    void push(char x)

    {

        if (top == MAX - 1)

        {

            printf("\nSTACK is over flow");

            exit(0);

        }

        else

        {

            top++;

            stack[top] = x;

        }

    }

    char pop()

    {

        if (top == -1)

        {

            printf("\nStack is under flow");

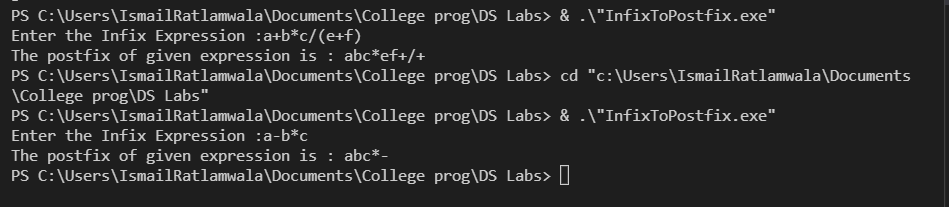
            exit(0);

        }

        else return (stack[top--]);

    }

**Output :**



Experiment no 3.

**Program :**

#include<stdio.h>

    #define size 100

    int top=-1;

    int arr[size];

    void push(int data){

        if( top!= size-1){

            top++;

            arr[top]=data;

        }

        else{

            printf("Satck overflow!!");

        }

    }

    int pop(){

        if( top!=-1){

            int c = arr[ top];

            top--;

            return c;

        }

    }

    int postfixEvaluation(char exp[]){

        for(int i=0;i<strlen(exp);i++){

            if(exp[i]>='0'&& exp[i]<='9'){

                push(exp[i]-'0');

            }

            else{

                int op2=pop();

                int op1=pop();

                switch(exp[i]){

                    case '+':

                        push(op1+op2);

                        break;

                    case '-':

                        push(op1-op2);

                        break;

                    case '\*':

                        push(op1\*op2);

                        break;

                    case '/':

                        push(op1/op2);

                        break;

                    case '^':

                        push(pow(op1,op2));

                        break;

                }

            }

        }

        return  arr[top];

    }

    int main(){

        char exp[size];

        printf("Enter the postfix expression: ");

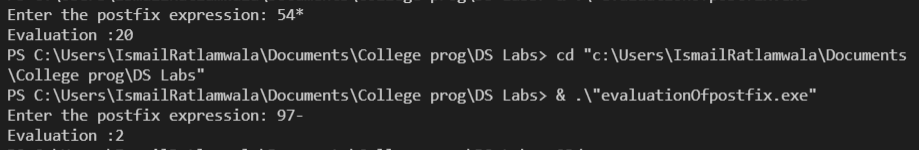
        gets(exp);

        int result = postfixEvaluation(exp);

        printf("Evaluation :%d\n",result);

    }

**Output :**



Experiment no 4.

**Program :**

#include<stdio.h>

    #include<stdlib.h>

    struct Queue

    {

        int capcacity;

        int rear,front;

        int \*arr;

    };

    typedef struct Queue \*PtrToNode;

    typedef PtrToNode queue;

    queue createQueue(int max){

        queue q;

        q=(struct Queue\*)malloc(sizeof(struct Queue));

        q->capcacity=max;

        q->front=-1,q->rear=-1;

        q->arr=(int\*)malloc(max\*sizeof(int));

        return q;

    }

    int isfull(queue q){

        return q->rear == q->capcacity-1;

    }

    int isempty(queue q){

        return q->rear ==-1 && q->front==-1;

    }

    queue enqueue(queue q){

        if(isfull(q)) printf("Overflow !\n");

        else{

            q->rear++;

            printf("Enter the element to be inserted : ");

            scanf("%d",&q->arr[q->rear]);

            if(q->front==-1) q->front=0;

        }

        return q;

    }

    queue dequeue(queue q){

        if(isempty(q)) printf("underflow !\n");

        else if(q->front==q->rear) {

            printf("Element dequeued : %d\n", q->arr[q->front]);

            q->front=q->rear=-1;

        }

        else{

            printf("Element dequeued : %d\n", q->arr[q->front]);

            q->front++;

        }

        return q;

    }

    void display(queue q){

        if(isempty(q)) printf("Queue is Empty\n");

        else{

            printf("Elements in queue : ");

            for (int i = q->front; i <= q->rear; i++) printf("%d ",q->arr[i]);

            printf("\n");

        }

    }

    int main()

    {

        int max = 10,choice;

        queue q;

        q=createQueue(max);

        do{

            printf("\n1.Enqueue  2.Dequeue  3.Display  4.Exit \n");

            printf("Enter your choice : ");

            scanf("%d",&choice);

            switch (choice)

            {

                case 1:

                    q=enqueue(q);

                    break;

                case 2:

                    q=dequeue(q);

                    break;

                case 3:

                    display(q);

                    break;

                case 4:

                    break;

                default:

                    printf("Invalid choice !");

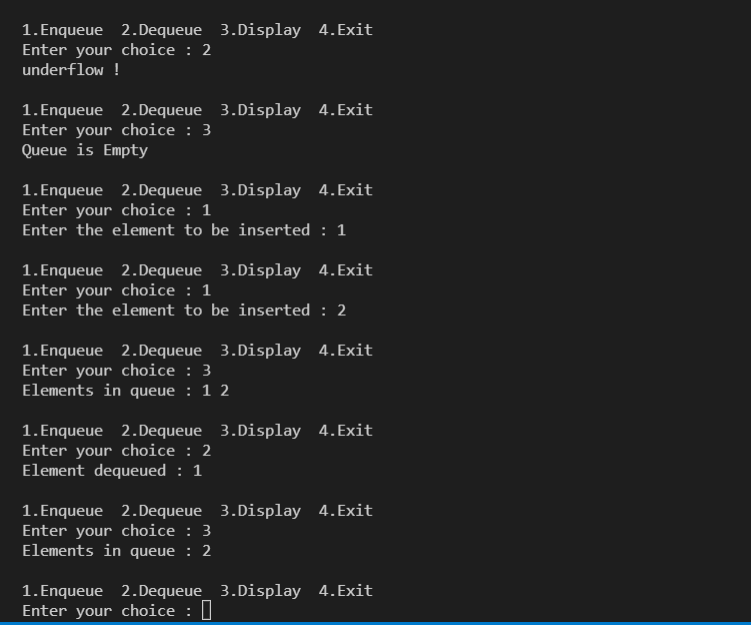
            }

        }while(choice!=4);

        return 0;

    }

**Output :**



Experiment no 5.

**Program :**

#include<stdio.h>

    #include<stdlib.h>

    struct Queue

    {

        int capcacity;

        int rear,front;

        int \*arr;

    };

    typedef struct Queue \*PtrToNode;

    typedef PtrToNode queue;

    queue createQueue(int max){

        queue q;

        q=(struct Queue\*)malloc(sizeof(struct Queue));

        q->capcacity=max;

        q->front=-1,q->rear=-1;

        q->arr=(int\*)malloc(max\*sizeof(int));

        return q;

    }

    int circularIncrement(int n,int max){

        n++;

        //printf("\ncirInc: %d\n",(n==max) ? 0 : n);

        return (n==max) ? 0 : n;

    }

    int isfull(queue q){

        return (q->rear+1)%q->capcacity==q->front;

    }

    int isempty(queue q){

        return q->rear ==-1 && q->front==-1;

    }

    queue enqueue(queue q){

        if(isfull(q)) printf("Overflow !\n");

        else{

            q->rear=circularIncrement(q->rear,q->capcacity);

            printf("Enter the element to be inserted : ");

            scanf("%d",&q->arr[q->rear]);

            if(q->front==-1) q->front=0;

        }

        return q;

    }

    queue dequeue(queue q){

        if(isempty(q)) printf("underflow !\n");

        else if(q->front==q->rear) {

            printf("Element dequeued : %d\n", q->arr[q->front]);

            q->front=q->rear=-1;

        }

        else{

            printf("Element dequeued : %d\n", q->arr[q->front]);

            q->front=circularIncrement(q->front,q->capcacity);

        }

        return q;

    }

    void display(queue q){

        if(isempty(q)) printf("Queue is Empty\n");

        else{

            printf("Elements in queue : front =%d rear =%d\n",q->front,q->rear);

            int i=q->front;

            while (1)

            {

                printf("%d ",q->arr[i]);

                if(i==q->rear) break;

                i=circularIncrement(i,q->capcacity);

            }

            printf("\n");

        }

    }

    int main(int argc, char const \*argv[])

    {

        int max = 4,choice;

        queue q;

        q=createQueue(max);

        do{

            printf("\n1.Enqueue  2.Dequeue  3.Display  4.Exit \n");

            printf("Enter your choice : ");

            scanf("%d",&choice);

            switch (choice)

            {

                case 1:

                    q=enqueue(q);

                    break;

                case 2:

                    q=dequeue(q);

                    break;

                case 3:

                    display(q);

                    break;

                case 4:

                    break;

                default:

                    printf("Invalid choice ^\_^");

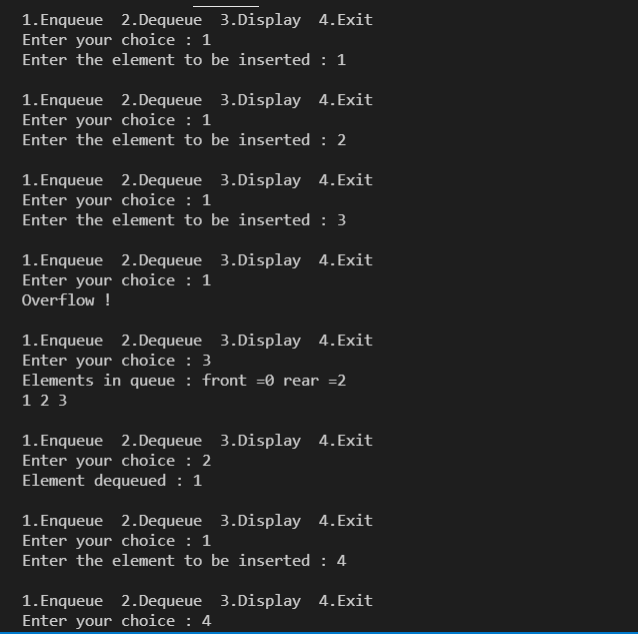
            }

        }while(choice!=4);

        return 0;

    }

**Output :**



Experiment no 6.

**Program :**

#include<stdio.h>

    #include<stdlib.h>

    struct node

    {

        int data;

        struct node \*next;

    };

    struct node \*head=NULL,\*newnode,\*ptr;

    struct node \*createnode(){

        //creating a new node

        newnode = (struct node\*)malloc(sizeof(struct node)); //allocates mem to new node, newnode contains base add of new node

        printf("Enter the data : ");

        scanf("%d",&newnode->data);

        newnode->next=NULL;

        return newnode;

    }

    int createll(){

        int length=0;

        while (1)

        {

            newnode=createnode();

            if(newnode->data==-1) {

                printf("Linked List created /n");

                return length;

            };

            //new node is created

            length++;

            //Linking this node to the List

            if(head==NULL) head= ptr= newnode;

            else{

                ptr->next=newnode;   //linking previous node's next to this (new)node

                ptr= newnode;        //updating our pointer to this (new)node

            }

        }

    }

    void display(){

        ptr = head;

        printf("Elements in list are :");

        while (ptr!=NULL)

        {

            printf("%d ",ptr->data);

            ptr=ptr->next;

        }

        printf("\n");

    }

    void insertstart(){

        newnode=createnode();

        newnode->next = head;

        head = newnode;

    }

    void insertend(){

        newnode=createnode();

        ptr=head;

        while (ptr->next!=NULL)

        {

            ptr=ptr->next;

        }

        ptr->next=newnode;

    }

    void insertmidway(int after,int length,int beforeOrAfter){

        if(after>length) printf("Invalid index\n");

        else{

            ptr=head;

            for (int i = 1; i <= after-beforeOrAfter; i++)

            {

                ptr=ptr->next;

            }

            newnode=createnode();

            newnode->next= ptr->next;

            ptr->next=newnode;

        }

    }

    void deleteStart(){

        ptr = head->next;

        free(head);

        head = ptr;

    }

    void deleteEnd(){

        ptr = head;

        while (ptr->next->next!=NULL)

        {

            ptr=ptr->next;

        }

        free(ptr->next->next);

        ptr->next=NULL;

    }

    void delete(int at,int length){

        if(at>length) printf("Invalid index\n");

        else{

            struct node \*temp;

            ptr=head;

            for (int i = 1; i <= at-2; i++)

            {

                ptr=ptr->next;

            }

            temp=ptr->next->next;

            free(ptr->next);

            ptr->next=temp;

        }

    }

    void deletell(){

        struct node \*temp;

        ptr=head;

        head=NULL;

        while (ptr!=NULL)

        {

            temp=ptr;

            ptr=ptr->next;

            free(temp);

        }

    }

    void sort(int n){

        for (int i = 0; i < n-1; i++){

            ptr=head;

            for(int j=0;j<n-i-1;j++){

                if(ptr->data > ptr->next->data){

                    int temp= ptr->data;

                    ptr->data=ptr->next->data;

                    ptr->next->data=temp;

                }

                ptr=ptr->next;

            }

        }

        printf("After sorting in ascending order,\n");

    }

    int main()

    {

        int length,at,choice;

        do{

            printf("\nMAIN MENU :\n");

            printf("1. To Create a list \t\t\t2. To Add a Node at the beginning \n3. To Add Node at the end \t\t4. To Add Node after the given node \n5. To Add Node before the given node \t6. To delete a Node at the beginning \n7. To delete a node from the end \t8. Delete a given Node\n9. Display the Linked List\t\t10. Delete entire List\n11. Sort the list \t\t\t12. Exit\nEnter the choice:");

            scanf("%d", &choice);

            switch (choice)

            {

            case 1:

                length= createll();

                break;

            case 2:

                insertstart();

                length++;

                break;

            case 3:

                insertend();

                length++;

                break;

            case 4:

                printf("add a new node before :\n");

                scanf("%d",&at);

                insertmidway(at,length,1);// 2 for before and 1 for after

                length++;

                break;

            case 5:

                printf("add a new node before :\n");

                scanf("%d",&at);

                insertmidway(at,length,2);// 2 for before and 1 for after

                length++;

                break;

            case 6:

                deleteStart();

                break;

            case 7:

                deleteEnd();

                break;

            case 8:

                printf("Delete node at index :\n");

                scanf("%d",&at);

                delete(at,length);

                length--;

                break;

            case 9:

                display();

                break;

            case 10:

                deletell();

                length=0;

                break;

            case 11:

                sort(length);

                display();

                break;

            }

        }while (choice != 12);

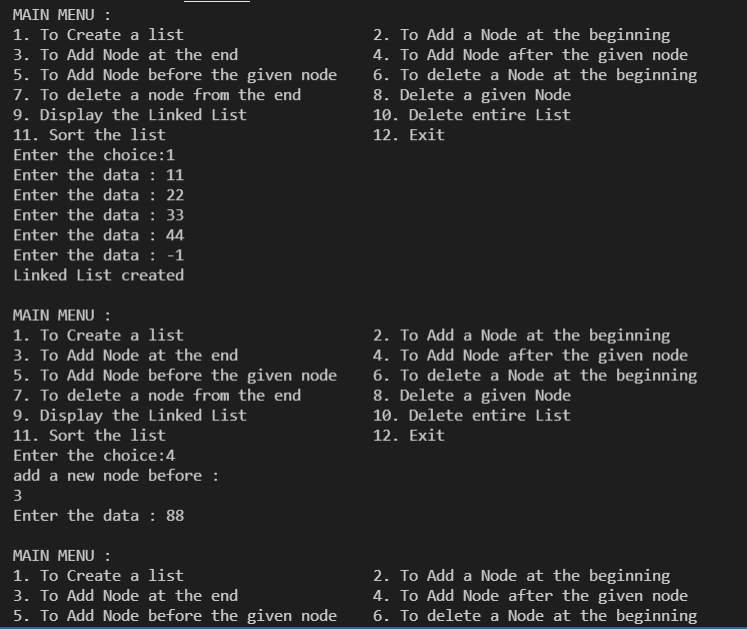
        deletell();

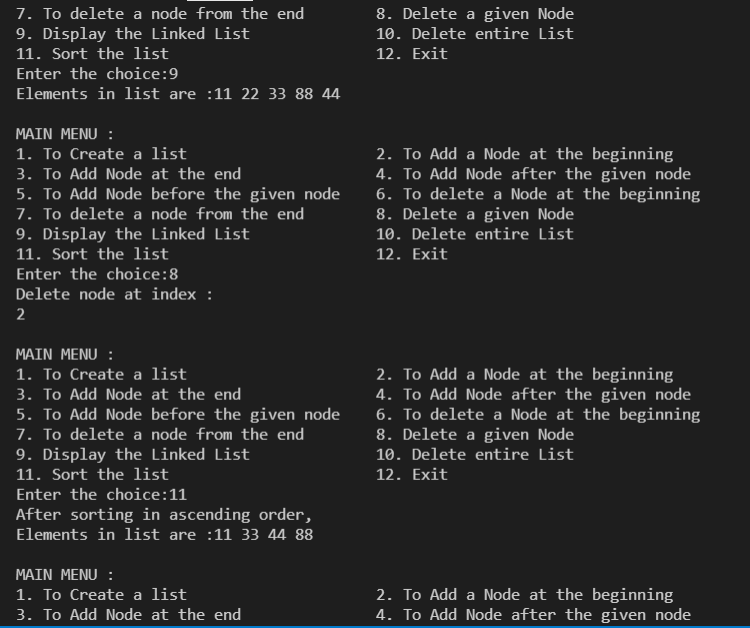
        display();

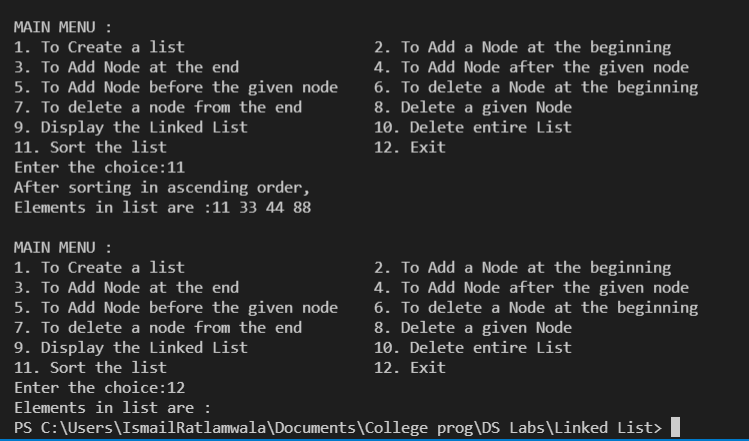
        return 0;

    }

**Output :**







Experiment no 7.

**Program :**

#include<stdio.h>

    #include<stdlib.h>

    struct node

    {

        int data;

        struct node \*next;

    };

    struct node \*head=NULL,\*newnode,\*ptr;

    struct node \*createnode(){

        //creating a new node

        newnode = (struct node\*)malloc(sizeof(struct node)); //allocates mem to new node, newnode contains base add of new node

        printf("Enter the data : ");

        scanf("%d",&newnode->data);

        newnode->next=head;

        return newnode;

    }

    int createll(){

        int length=0;

        while (1)

        {

            newnode=createnode();

            if(newnode->data==-1) {

                printf("Linked List created \n");

                return length;

            };

            length++;

            if(head==NULL) head= ptr= newnode;

            else{

                ptr->next=newnode;

                ptr= newnode;

            }

        }

    }

    void display(){

        if(head==NULL) printf("Empty List\n");

        else{

            ptr = head;

        printf("Elements in list are : ");

        while (ptr->next!=head)

        {

            printf("%d ",ptr->data);

            ptr=ptr->next;

        }

        printf("%d ",ptr->data);

        printf("\n");

        }

    }

    void insertstart(){

        newnode=createnode();

        newnode->next = head;

        ptr = head;

        while (ptr->next!=head) ptr=ptr->next;

        ptr->next=newnode;  //updates the value of head in last node

        head = newnode;

    }

    void insertend(){

        newnode=createnode();

        ptr=head;

        while (ptr->next!=head)

        {

            ptr=ptr->next;

        }

        ptr->next=newnode;

    }

    void insertmidway(int at,int length,int beforeOrAfter){

        if(at>length) printf("Invalid index\n");

        else{

            ptr=head;

            for (int i = 1; i <= at-beforeOrAfter; i++)

            {

                ptr=ptr->next;

            }

            newnode=createnode();

            newnode->next= ptr->next;

            ptr->next=newnode;

        }

    }

    void deleteStart(){

        struct node \*temp = head->next;

        free(head);

        ptr = temp;

        while (ptr->next!=head) ptr=ptr->next;

        ptr->next=temp;  //updates the value of head in last node

        head = temp;

    }

    void deleteEnd(){

        ptr = head;

        while (ptr->next->next!=head)

        {

            ptr=ptr->next;

        }

        free(ptr->next);

        ptr->next=head;

    }

    void delete(int at,int length){

        if(at>length) printf("Invalid index\n");

        else{

            struct node \*temp;

            ptr=head;

            for (int i = 1; i <= at-2; i++)

            {

                ptr=ptr->next;

            }

            temp=ptr->next->next;

            free(ptr->next);

            ptr->next=temp;

        }

    }

    void deletell(){

        struct node \*temp;

        ptr=head;

        while (ptr!=head)

        {

            temp=ptr;

            ptr=ptr->next;

            free(temp);

        }

        head=NULL;

    }

    void sort(int n){

        for (int i = 0; i < n-1; i++){

            ptr=head;

            for(int j=0;j<n-i-1;j++){

                if(ptr->data > ptr->next->data){

                    int temp= ptr->data;

                    ptr->data=ptr->next->data;

                    ptr->next->data=temp;

                }

                ptr=ptr->next;

            }

        }

        printf("After sorting in ascending order,\n");

    }

    int main()

    {

        int length,at,choice;

        do{

            printf("\n-----------------------------MAIN MENU--------------------------------------\n");

            printf("1. To Create a list \t\t\t2. To Add a Node at the beginning \n3. To Add Node at the end \t\t4. To Add Node after the given node \n5. To Add Node before the given node \t6. To delete a Node at the beginning \n7. To delete a node from the end \t8. Delete a given Node\n9. Display the Linked List\t\t10. Delete entire List\n11. Sort the list \t\t\t12. Exit\nEnter the choice:");

            scanf("%d", &choice);

            switch (choice)

            {

            case 1:

                length= createll();

                break;

            case 2:

                insertstart();

                length++;

                break;

            case 3:

                insertend();

                length++;

                break;

            case 4:

                printf("add a new node before :\n");

                scanf("%d",&at);

                insertmidway(at,length,1);// 2 for before and 1 for after

                length++;

                break;

            case 5:

                printf("add a new node before :\n");

                scanf("%d",&at);

                insertmidway(at,length,2);// 2 for before and 1 for after

                length++;

                break;

            case 6:

                deleteStart();

                break;

            case 7:

                deleteEnd();

                break;

            case 8:

                printf("Delete node at index :\n");

                scanf("%d",&at);

                delete(at,length);

                length--;

                break;

            case 9:

                display();

                break;

            case 10:

                deletell();

                length=0;

                break;

            case 11:

                sort(length);

                break;

            case 12:

                exit(0);

            }

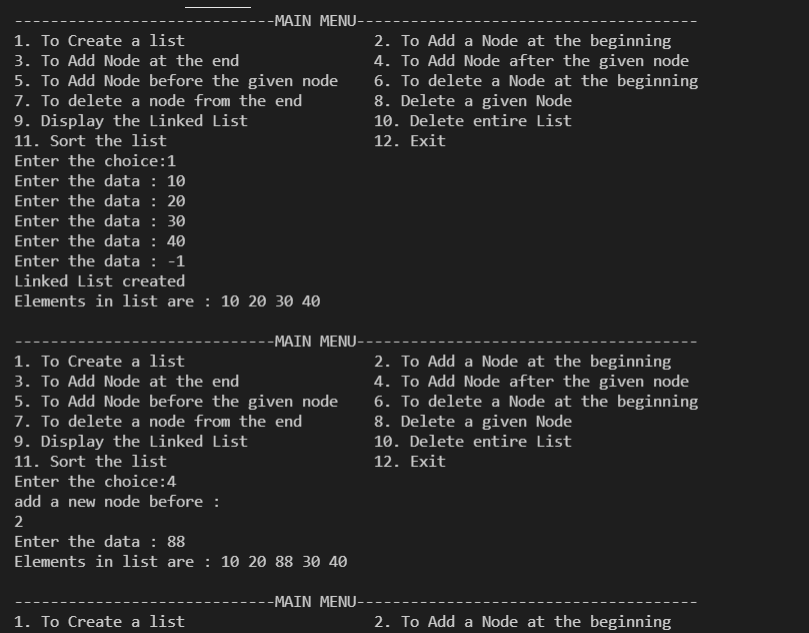
            display();

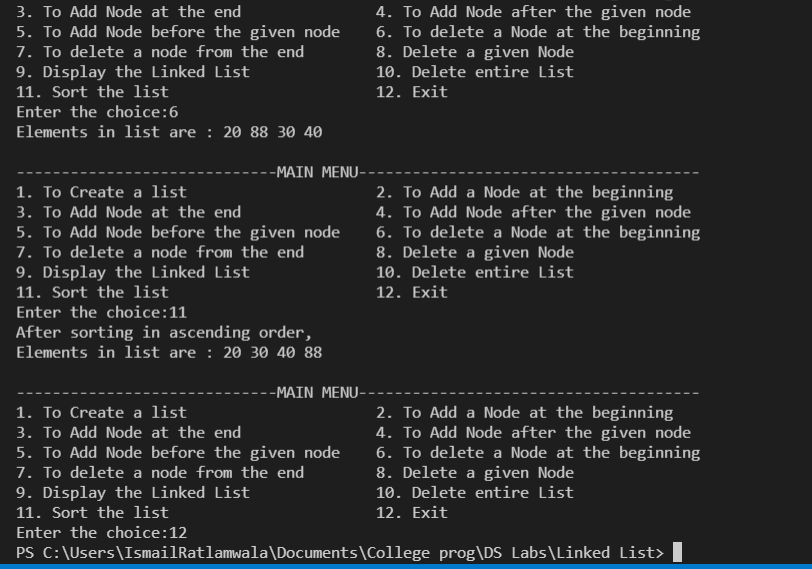
        }while (1);

        return 0;

    }

**Output :**





Experiment no 8.

**Stack using LL:**

**Program :**

    #include<stdio.h>

    #include<stdlib.h>

    struct stack

    {

        int data;

        struct stack \*next;

    };

    struct stack \*top=NULL,\*ptr;

    struct stack \*createnode(){

        struct stack \*newnode;

        newnode = (struct stack\*)malloc(sizeof(struct stack));

        printf("Enter the number to be pushed onto the stack : ");

        scanf("%d",&newnode->data);

        newnode->next=NULL;

        return newnode;

    }

    void push(){

        struct stack \*newnode=createnode();

        newnode->next=top;

        top=newnode;

    }

    void pop(){

        if(top==NULL) printf("Underflow\n");

        else{

            printf("Poped element : %d\n",top->data);

            ptr=top;

            top=top->next;

            free(ptr);

        }

    }

    void display(){

        if(top==NULL) printf("No elements in the stack\n");

        else{

            ptr=top;

            printf("Elements in stack : ");

            while (ptr!=NULL)

            {

                printf("%d ",ptr->data);

                ptr=ptr->next;

            }

            printf("\n");

        }

    }

    int main()

    {

        do{

            int choice;

            printf("\n1. PUSH   2. POP   3. DISPLAY   4. EXIT\nEnter your choice : ");

            scanf("%d",&choice);

            switch (choice)

            {

            case 1:

                push();

                break;

            case 2:

                pop();

                break;

            case 3:

                display();

                break;

            case 4:

                exit(0);

                break;

            default:

                printf("Invalid choice !\n");

                break;

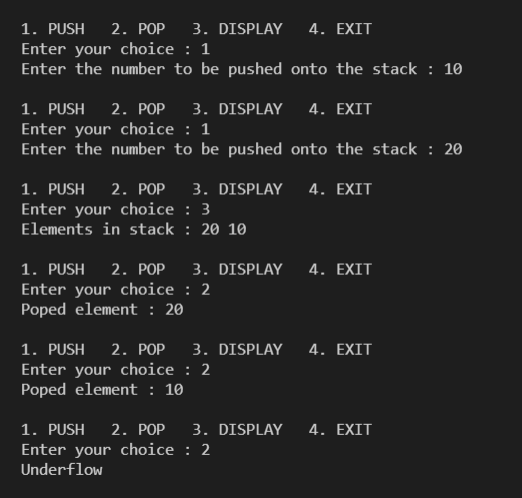
            }

        }while(1);

        return 0;

    }

**Output :**



**Queue using linked list :**

**Program :**

    #include<stdio.h>

    #include<stdlib.h>

    struct queue

    {

        int data;

        struct queue \*next;

    };

    struct queue \*front=NULL,\*ptr,\*temp;

    struct queue \*createnode(){

        struct queue \*newnode;

        newnode = (struct queue\*)malloc(sizeof(struct queue));

        printf("Enter the number to enqueued : ");

        scanf("%d",&newnode->data);

        newnode->next=NULL;

        return newnode;

    }

    void enqueue(){

        struct queue \*newnode=createnode();

        if(front==NULL) front=ptr=newnode;

        else{

            ptr->next=newnode;

            ptr=newnode;

        }

    }

    void dequeue(){

        if(front==NULL) printf("Underflow\n");

        else{

            printf("Dequeued element : %d\n",front->data);

            front=front->next;

        }

    }

    void display(){

        struct queue ;

        if(front==NULL) printf("No elements in the queue\n");

        else{

            temp=front;

            printf("Elements in queue : ");

            while (temp!=NULL)

            {

                printf("%d ",temp->data);

                temp=temp->next;

            }

            printf("\n");

        }

    }

    int main()

    {

        do{

            int choice;

            printf("\n1. ENQUEUE   2. DEQUEUE   3. DISPLAY   4. EXIT\nEnter your choice : ");

            scanf("%d",&choice);

            switch (choice)

            {

            case 1:

                enqueue();

                break;

            case 2:

                dequeue();

                break;

            case 3:

                display();

                break;

            case 4:

                exit(0);

                break;

            default:

                printf("Invalid choice !\n");

                break;

            }

        }while(1);

        return 0;

    }

**Output :**

Experiment no 9.

**Program :**

    #include<stdio.h>

    #include<stdlib.h>

    struct node{

        int data ;

        struct node \*right,\*left;

    };

    struct node \*createNode(int data){

        struct node\* tree;

        tree=(struct node\*)malloc(sizeof(struct node));

        tree->left=tree->right=NULL;

        tree->data=data;

        return tree;

    }

    struct node \*createBST(int data){

            return createNode(data);

    }

    void insert(struct node \*root,int data){

        if(data < root->data){

            if(root->left==NULL){

                root->left=createNode(data);

                return NULL;

            }

            else{

                insert(root->left,data);

            }

        }

        else if(data > root->data){

            if(root->right==NULL){

                root->right=createNode(data);

                return NULL;

            }

            else{

                insert(root->right,data);

            }

        }

    }

    struct node \*preorder(struct node \*root){

        if(root==NULL) return NULL;

        printf("%d ",root->data);

        preorder(root->left);

        preorder(root->right);

    }

    struct node \*inorder(struct node \*root){

        if(root==NULL) return NULL;

        inorder(root->left);

        printf("%d ",root->data);

        inorder(root->right);

    }

    struct node \*postorder(struct node \*root){

        if(root==NULL) return NULL;

        postorder(root->left);

        postorder(root->right);

        printf("%d ",root->data);

    }

    int smallest(struct node \*root){

        if(root->left==NULL) {       //go to left extreme

            return root->data;

        }

        smallest(root->left);

    }

    int largest(struct node \*root){

        if(root->right==NULL) {       //go to right extreme

            return root->data;

        }

        largest(root->right);

    }

    void mirror(struct node \*root){

        if(root==NULL) return NULL;

        mirror(root->left);

        mirror(root->right);

        struct node \*temp = root->left;

        root->left=root->right;

        root->right=temp;

    }

    int countNodes(struct node\*root){

        if(root==NULL) return 0;

        return(countNodes(root->left)+countNodes(root->right)+1);

    }

    int internalNodeCount(struct node\*root){

        if(root==NULL) return 0;

        if((root->right==NULL && root->left==NULL)) return 0;

        return(internalNodeCount(root->left)+internalNodeCount(root->right)+1);

    }

    int LeafNodeCount(struct node\*root){

        if(root==NULL) return 0;

        if(root->right==NULL && root->left==NULL) return 1;

        return(LeafNodeCount(root->right) + LeafNodeCount(root->left));

    }

    int height(struct node \*root){

        if(root==NULL) return 0;

        int leftHeight = height(root->left);

        int rightHeight = height(root->right);

        return(\_\_max(leftHeight,rightHeight)+1);

    }

    struct node\* deleteNode(struct node\*root,int data){

        if(data<root->data)

            root->left= deleteNode(root->left,data);

        else if(data>root->data)

            root->right= deleteNode(root->right,data);

        else{

            if(root->right==NULL && root->left==NULL){

                free(root);

                return NULL;

            }

            else if(root->right==NULL || root->left==NULL){

                struct node\* temp;

                if(root->right!=NULL) temp =root->right;

                else temp =root->left;

                free(root);

                return temp;

            }

            else{

                int inorderSucc= smallest(root->right);

                root->data=inorderSucc;

                root->right= deleteNode(root->right,inorderSucc);

            }

        }

    }

    struct node\* deleteTree(struct node\*root){

        if(root==NULL) return NULL;

        if(root->right==NULL && root->left==NULL){

            printf("%d ",root->data);

            free(root);

            return NULL;

        }

        root->left =deleteTree(root->left);

        root->right= deleteTree(root->right);

        deleteTree(root);

    }

    int main()

    {

        struct node \*root=NULL;

        int option,val;

        do{

            printf("\n\n----------------------------- MAIN MENU ------------------------------------------");

            printf("\n 1. Insert Element");

            printf("\t\t\t 2. Preorder Traversal");

            printf("\n 3. Inorder Traversal");

            printf("\t\t\t 4. Postorder Traversal");

            printf("\n 5. Find the smallest element");

            printf("\t\t 6. Find the largest element");

            printf("\n 7. Delete an element");

            printf("\t\t\t 8. Count total number of nodes");

            printf("\n 9. Count total number of leaf nodes");

            printf("\t 10. Count total number of internal nodes");

            printf("\n 11. Determine the height of the tree");

            printf("\t 12. Find the mirror image of the tree");

            printf("\n 13. Delete the tree");

            printf("\t\t\t 14. Exit\n");

            printf("\n Enter your option : ");

            scanf("%d", &option);

            switch (option)

            {

                case 1:

                    printf("\n Enter the value of the new node : ");

                    scanf("%d", &val);

                    if(root==NULL) {

                    root=createBST(val);

                    }

                    else insert(root,val);

                    break;

                case 2:

                    printf("\n The elements of the tree in preorder are : ");

                    preorder(root);

                    break;

                case 3:

                    printf("\n The elements of the tree in inorder are : ");

                    inorder(root);

                    break;

                case 4:

                    printf("\n The elements of the tree in postorder are : ");

                    postorder(root);

                    break;

                case 5:

                    printf("\n Smallest element is :%d",smallest(root));

                    break;

                case 6:

                    printf("\n Largest element is : %d",largest(root));

                    break;

                case 7:

                    printf("\n Enter the element to be deleted : ");

                    scanf("%d", &val);

                    deleteNode(root, val);

                    break;

                case 8:

                    printf("\n Total no. of nodes = %d", countNodes(root));

                    break;

                case 9:

                    printf("\n Total no. of leaf nodes = %d",LeafNodeCount(root));

                    break;

                case 10:

                    printf("\n Total no. of internal nodes = %d",internalNodeCount(root));

                    break;

                case 11:

                    printf("\n The height of the tree = %d", height(root)-1);

                    break;

                case 12:

                    mirror(root);

                    break;

                case 13:

                    root=deleteTree(root);

                    break;

            }

        } while(option != 14);

        return 0;

    }

**Output :**

