**EXPERIMENT 1**

#include<iostream>

using namespace std;

int main()

{

int a[20],n,i,m,pos,e,ch,temp=0;

cout<<"Enter the size of array: ";

cin>>n;

for(i=0;i<n;i++)

{

cin>>a[i];

}

cout<<"\nEnter choice:\n1 for insertion\n2 for deletion\n3 for searching\n4 for display\n";

cin>>ch;

while(ch!=0)

{

switch(ch)

{

case 1:

cout<<"Enter the position and element you want to insert\n";

cin>>pos>>e;

m=n-pos-1;

for(i=n;i>=m;i--)

{

a[i+1]=a[i];

}

a[pos]=e;

n=n+1;

break;

case 2:

cout<<"Enter the position of element you want to delete\n";

cin>>pos;

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

{

a[i]=a[i+1];

}

n=n-1;

break;

case 3:

temp=0;

cout<<"\nEnter the element to search: ";

cin>>e;

for(i=0;i<n;i++)

{

if(a[i]==e)

{

cout<<"\nPosition is: "<<i+1;

temp=1;

break;

}

}

if(temp==0)

cout<<"\nElement not found";

break;

case 4:

for(i=0;i<n;i++)

{

cout<<a[i]<<" ";

}

cout<<endl;

break;

}

cout<<"\nEnter choice again: ";

cin>>ch;

}

}

**EXPERIMENT 2**

#include<iostream>

using namespace std;

int main()

{

int a[20],n,i,e,temp=0;

cout<<"Enter the size of array: ";

cin>>n;

for(i=0;i<n;i++)

{

cin>>a[i];

}

for(i=0;i<n;i++)

{

cout<<a[i]<<" ";

}

cout<<endl;

cout<<"\nEnter the element to search: ";

cin>>e;

for(i=0;i<n;i++)

{

if(a[i]==e)

{

cout<<"\nPosition of "<<e<<" is: "<<i+1;

temp=1;

break;

}

}

if(temp==0)

cout<<"\nElement not found\n";

return 0;

}

**EXPERIMENT 3**

#include<iostream>

using namespace std;

int main()

{

int a[20],n,i,j,e,flag=0,temp;

intmid,min,max;

cout<<"Enter the size of array: ";

cin>>n;

for(i=0;i<n;i++)

{

cin>>a[i];

}

cout<<"Array is:\n";

for(i=0;i<n;i++)

{

cout<<a[i]<<" ";

}

cout<<endl;

for(i=0;i<n;i++)

{

for(j=0;j<n-1;j++)

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

cout<<"\nAfter sorting the array: \n";

for(i=0;i<n;i++)

{

cout<<a[i]<<" ";

}

cout<<endl;

cout<<"\nEnter the element to search: ";

cin>>ele;

min=0;

max=n-1;

mid=(min+max)/2;

if(min<max)

{

for(i=0;i<n;i++)

{

if(a[mid]==ele)

{

cout<<"Position of "<<ele<<" is: "<<mid+1;

flag=1;

break;

}

else if(a[mid]>ele)

{

max=mid-1;

}

else if(a[mid]<ele)

{

min=mid+1;

}

mid=(min+max)/2;

}

}

if(flag==0)

cout<<"Element not found ";

return 0;

}

**Experiment 4**

#include<iostream>

using namespace std;

int main()

{

int a[20],n,i,j,ele,flag=0,temp;

intmid,min,max;

cout<<"Enter the size of array: ";

cin>>n;

for(i=0;i<n;i++)

{

cin>>a[i];

}

cout<<"Array is:\n";

for(i=0;i<n;i++)

{

cout<<a[i]<<" ";

}

cout<<endl;

for(i=0;i<n;i++)

{

for(j=0;j<n-1;j++)

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

cout<<"\nAfter sorting the array: \n";

for(i=0;i<n;i++)

{

cout<<a[i]<<" ";

}

cout<<endl;

cout<<"\nEnter the element to search: ";

cin>>ele;

min=0;

max=n-1;

mid=(min+max)/2;

if(min<max)

{

for(i=0;i<n;i++)

{

if(a[mid]==ele)

{

cout<<"Position of "<<ele<<" is: "<<mid+1;

flag=1;

break;

}

else if(a[mid]>ele)

{

max=mid-1;

}

else if(a[mid]<ele)

{

min=mid+1;

}

mid=(min+max)/2;

}

}

if(flag==0)

cout<<"Element not found ";

return 0;

}

**Experiment 5**

#include<bits/stdc++.h>

using namespace std;

#define MAX 6

int Stack[MAX];

int top = -1;

void push(int x);

void pop();

void display();

int main ()

{

int choice, val;

choice = -1;

while(choice !=4)

{

cin>> choice;

switch(choice)

{

case 1:// cout<<”Enter the element to push”;

cin>>val;

push(val);

break;

case 2: // cout<<”Enter the element to pop”;

pop();

break;

case 3:// cout<<”Display the stack elements”;

display();

break;

default:

break;

}

cout<<"choice please"<<endl;

}

return 0;

}

void push(int x)

{ if(top==MAX)

cout<<"overflow "<<endl;

else

{

Stack[top+1]=x;

top++;

}

}

void pop()

{ if(top==-1)

cout<<"underflow sir!"<<endl;

else

{cout<<"deleted element is"<<" "<<Stack[top--]<<endl;

}

}

void display()

{for(int i=top;i>-1;i--)

cout<<Stack[i]<<" ";

cout<<endl;

}

**Experiment 6**

#include<iostream>

using namespace std;

class convert

{

char a[20];

int top;

public:

convert()

{

top=-1;

}

void push(int x)

{

a[++top]=x;

}

char pop()

{

return a[top--];

}

int priority(char p)

{

if(p=='(' )

return 0;

else if(p=='+'||p=='-')

return 1;

else if(p=='\*'|| p=='/')

return 2;

else if(p=='^')

return 3;

}

voidin\_po(char exp[20])

{

char \*e;

char x;

e=exp;

while(\*e!='\0')

{

if(isalnum(\*e))

{

cout<<\*e;

}

else if(\*e=='(')

{

push(\*e);

}

else if(\*e==')')

{

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

cout<<x;

}

else

{

while(priority(a[top])>=priority(\*e))

cout<<pop();

push(\*e);

}

e++;

}

while(top!=-1)

{

x=pop();

cout<<x;

}

}

};

class s1

{

public:

char a[20];

int top;

s1()

{

top=-1;

}

void push(int x)

{

a[++top]=x;

}

int pop()

{

return a[top--];

}

voidin\_po(char exp[20])

{

char \*e;

char x;

intnum;

e=exp;

while(\*e!='\0')

{

if(isalnum(\*e))

{

num=\*e-48;

push(num);

}

else

{

int a=pop();

int b=pop();

int c;

switch(\*e)

{

case '+':

c=a+b;

push(c);

break;

case '-':

c=a-b;

push(c);

break;

case '/':

c=a/b;

push(c);

break;

case '\*':

c=a\*b;

push(c);

break;

case '^':

c=a^b;

push(c);

break;

}

}

e++;

}

while(top!=-1)

{

cout<<pop();

}

}

};

int main()

{

intch;

cout<<"Enter choice:\n1 for Infix to postfix conversion\n2 for Postfix Evaluation\n";

cin>>ch;

if(ch==1)

{

convert r;

charss[20];

cin>>ss;

r.in\_po(ss);

}

else if(ch==2)

{

s1 r;

charss[20];

cin>>ss;

r.in\_po(ss);

}

}

**Experiment 7**

#include<iostream>

using namespace std;

struct node

{

int info;

struct node \*next;

};

structlinked\_list

{

node \*front,\*rear;

linked\_list()

{

front=NULL;

rear=NULL;

}

voidinsert\_beg();

void display();

node\* createnode(int);

void del();

};

node\* linked\_list::createnode(int x)

{

node \*temp=new node;

temp->info=x;

temp->next=NULL;

return temp;

}

voidlinked\_list::insert\_beg()

{

intele;

cin>>ele;

node \*p,\*ptr;

p=createnode(ele);

if(front==NULL)

{

front=p;

rear=p;

}

else

{

rear->next=p;

rear=p;

}

}

voidlinked\_list::display()

{

rear->next=front;

node \*temp=front;

if(front==NULL)

{

cout<<"Underflow";

cout<<"\nQueue is empty";

}

else

{

while(temp->next!=front)

{

cout<<temp->info<<"->";

temp=temp->next;

}

cout<<temp->info;

}

}

voidlinked\_list::del()

{

if(front==rear)

{

cout<<"Deleted element is: "<<front->info<<endl;

front=NULL;

}

else if(front==NULL)

{

cout<<"Underflow";

cout<<"\nQueue is empty";

}

else

{

cout<<"Deleted element is: "<<front->info<<endl;

node\* temp=front;

front=front->next;

delete temp;

}

}

int main()

{

intch,n,count=0;

linked\_list l1;

cout<<"Enter the size of queue ";

cin>>n;

cout<<"\nEnter choice\n1.Insert\n2.Display\n3.Delete\n";

cin>>ch;

while(ch)

{

if(ch==1 )

{

if(count==n)

{

cout<<"Overflow";

}

else

{

l1.insert\_beg();

count++;

}

}

else if(ch==2)

{

l1.display();

}

else if(ch==3)

{

l1.del();

count--;

}

cout<<"\nEnter choice again: ";

cin>>ch;

}

}

**Experiment 8**

#include<iostream>

using namespace std;

void swap(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition (intarr[], int low, int high)

{

int pivot = arr[high];

int i =low - 1;

for (int j = low; j <= high- 1; j++)

{

if (arr[j] <= pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

voidquickSort(intarr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main()

{

int a[100],i,j,m,s;

cout<<"Enter no of elements to be entered in array: ";

cin>>s;

for(i=0;i<s;i++)

{

cin>>a[i];

}

quickSort(a,0,s-1);

for (i=0;i<s;i++)

{

cout<<a[i]<<" ";

}

}

**Experiment 9**

#include<iostream>

using namespace std;

void swap(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition (intarr[], int low, int high)

{

int pivot = arr[high];

int i =low - 1;

for (int j = low; j <= high- 1; j++)

{

if (arr[j] <= pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

voidquickSort(intarr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main()

{

int a[100],i,j,m,s;

cout<<"Enter no of elements to be entered in array: ";

cin>>s;

for(i=0;i<s;i++)

{

cin>>a[i];

}

quickSort(a,0,s-1);

for (i=0;i<s;i++)

{

cout<<a[i]<<" ";

}

}

**Experiment 11**

#include<iostream>

using namespace std;

struct node

{

int data;

node \*left;

node \*right;

};

node \*createnode(int x)

{

node \*temp=new node;

temp->data=x;

temp->left=temp->right=NULL;

return temp;

}

node\* insertion(node \*root,int value)

{

if(root==NULL)

{

root=createnode(value);

}

else if(value < root->data)

root->left=insertion(root->left,value);

else if(value > root->data)

root->right=insertion(root->right,value);

return root;

}

voidinorder(struct node \*root)

{

if (root != NULL)

{

inorder(root->left);

cout<<root->data<<" ";

inorder(root->right);

}

}

voidpreorder(struct node \*root)

{

if (root != NULL)

{

cout<<root->data<<" ";

preorder(root->left);

preorder(root->right);

}

}

voidpostorder(struct node \*root)

{

if (root != NULL)

{

postorder(root->left);

postorder(root->right);

cout<<root->data<<" ";

}

}

node\* searchnode(node \*root,int x)

{

if(root==NULL){

cout<<"Node not found";

return NULL;

}

else if(root->data==x)

{

cout<<"\nNode exists\n";

return NULL;

}

else if(x < root->data)

root->left=searchnode(root->left,x);

else

root->right=searchnode(root->right,x);

}

node\* delnode(node \*root,int x)

{

if(root==NULL)

cout<<"Node not found";

else if(root->data==x && root->right==NULL && root->left==NULL)

{

delete root;

root=NULL;

return root;

}

else if(x < root->data)

root->left=delnode(root->left,x);

else

root->right=delnode(root->right,x);

}

intfindheight(node \*root)

{

intlh,rh;

if(root==NULL)

return 0;

else

{

lh=findheight(root->left);

rh=findheight(root->right);

if(lh>rh)

return (lh+1);

else

return (rh+1);

}

}

intfindsize(node \*root)

{

if(root==NULL)

return 0;

else

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

}

intcheckbst(node \*root)

{

if(root==NULL)

return 1;

if(root->left!=NULL && root->left->data > root->data)

return 0;

if(root->right!=NULL && root->right->data < root->data)

return 0;

if(!checkbst(root->left) || !checkbst(root->right))

return 0;

return 1;

}

int count=0;

intchildnodes(node \*root)

{

if(root==NULL)

return 0;

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

count++;

else{

childnodes(root->left);

childnodes(root->right);

}

return count;

}

intinternalnodes(node \*root)

{

if(root==NULL)

return 0;

if(root->left!=NULL || root->right!=NULL)

count++;

childnodes(root->left);

childnodes(root->right);

return count;

}

node\* heightofnode(node \*root,int x)

{

if(root==NULL){

cout<<"Node not found";

return NULL;

}

else if(root->data==x)

{

int x=findheight(root);

cout<<"\nHeight of the node is: "<<x-1<<endl;

return NULL;

}

else if(x < root->data)

root->left=heightofnode(root->left,x);

else

root->right=heightofnode(root->right,x);

}

int main()

{

node \*root = NULL;

node \*ptr;

intx,n,ch;

cout<<"Enter choice\n1.Insert\n2.Search\n3.Traverse\n4.Delete\n5.Height or Depth of tree\n6.Size of the tree\n7.BST or not\n8.No of child nodes\n9.No of internal nodes\n10.Height of node\n";

cin>>ch;

while(ch)

{

switch(ch)

{

case 1:

cout<<"Enter the no of nodes: ";

cin>>n;

for(int i=0;i<n;i++)

{

cin>>x;

root=insertion(root,x);

}

break;

case 2:

cout<<"\nEnter an element you want to search: ";

cin>>x;

ptr=root;

ptr=searchnode(ptr,x);

break;

case 3:

ptr=root;

cout<<"Inorder: ";

inorder(ptr);

cout<<"\nPreorder: ";

preorder(ptr);

cout<<"\nPostorder: ";

postorder(ptr);

break;

case 4:

cout<<"\nEnter an element you want to delete: ";

cin>>x;

ptr=root;

delnode(ptr,x);

break;

case 5:

ptr=root;

int h;

h=findheight(ptr);

cout<<"Height of the tree is: "<<h-1<<endl;

break;

case 6:

ptr=root;

x=findsize(ptr);

cout<<"Size of the tree: "<<x<<endl;

break;

case 7:

ptr=root;

x=checkbst(ptr);

if(x==0)

cout<<"\nTree is not a BST\n";

else

cout<<"\nIt is a BST\n";

break;

case 8:

ptr=root;

x=childnodes(ptr);

if(x==0)

cout<<"\nNo child nodes present\n";

else

cout<<"Child nodes are: "<<x<<endl;

count=0;

break;

case 9:

ptr=root;

x=internalnodes(ptr);

if(x==0)

cout<<"\nEmpty tree\n";

else

cout<<"\nInternal nodes are: "<<x+1<<endl;

count=0;

break;

case 10:

cout<<"\nEnter the node you want to find the path for: ";

cin>>x;

ptr=root;

heightofnode(ptr,x);

break;

}

cout<<"\nEnter choice: ";

cin>>ch;

}

return 0;

}