**Experiment 5**

#include&lt;bits/stdc++.h&gt;

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&gt;&gt; choice;

switch(choice)

{

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

cin&gt;&gt;val;

push(val);

break;

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

pop();

break;

case 3:// cout&lt;&lt;”Display the stack elements”;

display();

break;

default:

break;

}

cout&lt;&lt;&quot;choice please&quot;&lt;&lt;endl;

}

return 0;

}

void push(int x)

{ if(top==MAX)

cout&lt;&lt;&quot;overflow &quot;&lt;&lt;endl;

else

{

Stack[top+1]=x;

top++;

}

}

void pop()

{ if(top==-1)

cout&lt;&lt;&quot;underflow sir!&quot;&lt;&lt;endl;

else

{cout&lt;&lt;&quot;deleted element is&quot;&lt;&lt;&quot; &quot;&lt;&lt;Stack[top--]&lt;&lt;endl;

}

}

void display()

{for(int i=top;i&gt;-1;i--)

cout&lt;&lt;Stack[i]&lt;&lt;&quot; &quot;;

cout&lt;&lt;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;

}

void in\_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--];

}

void in\_po(char exp[20])

{

char \*e;

char x;

int num;

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()

{

int ch;

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

cin>>ch;

if(ch==1)

{

convert r;

char ss[20];

cin>>ss;

r.in\_po(ss);

}

else if(ch==2)

{

s1 r;

char ss[20];

cin>>ss;

r.in\_po(ss);

}

}

**Experiment 7**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node \*next;

};

struct linked\_list

{

node \*front,\*rear;

linked\_list()

{

front=NULL;

rear=NULL;

}

void insert\_beg();

void display();

node\* createnode(int);

void del();

};

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

{

node \*temp=new node;

temp->data=x;

temp->next=NULL;

return temp;

}

void linked\_list::insert\_beg()

{

int ele;

cin>>ele;

node \*p,\*ptr;

p=createnode(ele);

if(front==NULL)

{

front=p;

rear=p;

}

else

{

rear->next=p;

rear=p;

}

}

void linked\_list::display()

{

rear->next=front;

node \*temp=front;

if(front==NULL)

{

cout<<"Underflow";

cout<<"\nQueue is empty";

}

else

{

while(temp->next!=front)

{

cout<<temp->data<<"->";

temp=temp->next;

}

cout<<temp->data;

}

}

void linked\_list::del()

{

if(front==rear)

{

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

front=NULL;

}

else if(front==NULL)

{

cout<<"Underflow";

cout<<"\nQueue is empty";

}

else

{

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

node\* temp=front;

front=data->next;

delete temp;

}

}

int main()

{

int ch,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 insertionsort(int arr[],int n)

{

int i,j,k,temp,m;

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

{

m=arr[i];

j=i;

while(arr[j-1]>m && j>0)

{

arr[j]=arr[j-1];

j--;

}

arr[j]=m;

}

}

int main()

{

int i,arr[20],n;

cout<<"Enter no of elements in an array: ";

cin>>n;

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

cin>>arr[i];

insertionsort(arr,n);

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

{

cout<<arr[i]<<" ";

}

int item;

cout<<"\nEnter ITEM to be inserted : ";

cin>>item;

i = n-1;

while(item<arr[i] && i>=0)

{

arr[i+1] = arr[i];

i--;

}

arr[i+1] = item;

n++;

cout<<"\nAfter insertion array is :\n";

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

{

cout<<arr[i]<<" ";

}

}

**Experiment 9**

#include<iostream>

using namespace std;

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

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition (int arr[], 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);

}

void quickSort(int arr[], 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 10**

|  |
| --- |
|  |
|  | #include<iostream>  using namespace std; |
|  | int merge(int \*A, int M, int \*B, int N, int \*C) |
|  | { //Enter your code here |
|  | int p = 0; |
|  | int q = 0; |
|  | int k = 0; |
|  | int c=0; |
|  | while ( p < M && q < N) { |
|  | if (A[p] < B[q]) |
|  | { |
|  | C[k++] = A[p++]; |
|  | c++; |
|  | } |
|  | else { |
|  | C[k++] = B[q++]; |
|  | c++; } |
|  | } |
|  | while ( p < M) { |
|  | C[k++] = A[p++]; |
|  | } |
|  | while ( q < N) { |
|  | C[k++] = B[q++]; |
|  | } |
|  | return c; |
|  | } |
|  |  |
|  | int main() |
|  | { |
|  | int i,M,N; |
|  | cin >> M >> N; |
|  | int A[M],B[N],C[M+N]; |
|  | int X; |
|  | for(i=0;i<=M-1;i++) |
|  | cin >> A[i]; |
|  | for(i=0;i<=N-1;i++) |
|  | cin >> B[i]; |
|  | X = merge(A,M,B,N,C); |
|  | for(i=0;i<=M+N-1;i++) |
|  | cout << C[i] << " "; |
|  | cout << endl << X; |
|  | return 0; }  **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\* ins(node \*root,int value) | | |  | | { | | |  | | if(root==NULL) | | |  | | { | | |  | | root=createnode(value); | | |  | | } | | |  | | else if(value < root->data) | | |  | | root->left=ins(root->left,value); | | |  | | else if(value > root->data) | | |  | | root->right=ins(root->right,value); | | |  | |  | | |  | | return root; | | |  | | } | | |  | |  | | |  | | void inorder(struct node \*root) | | |  | | { | | |  | | if (root != NULL) | | |  | | { | | |  | | inorder(root->left); | | |  | | cout<<root->data<<" "; | | |  | | inorder(root->right); | | |  | | } | | |  | | } | | |  | |  | | |  | | void preorder(struct node \*root) | | |  | | { | | |  | | if (root != NULL) | | |  | | { | | |  | | cout<<root->data<<" "; | | |  | | preorder(root->left); | | |  | | preorder(root->right); | | |  | | } | | |  | | } | | |  | |  | | |  | | void postorder(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); | | |  | | } | | |  | |  | | |  | | int findheight(node \*root) | | |  | | { | | |  | | int lh,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); | | |  | | } | | |  | | } | | |  | |  | | |  | | int findsize(node \*root) | | |  | | { | | |  | | if(root==NULL) | | |  | | return 0; | | |  | | else | | |  | | return (findsize(root->left)+1+findsize(root->right)); | | |  | | } | | |  | |  | | |  | | int checkbst(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; | | |  | | int childnodes(node \*root) | | |  | | { | | |  | | if(root==NULL) | | |  | | return 0; | | |  | | if(root->left==NULL && root->right==NULL) | | |  | | count++; | | |  | | else{ | | |  | | childnodes(root->left); | | |  | | childnodes(root->right); | | |  | | } | | |  | | return count; | | |  | | } | | |  | | int internalnodes(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; | | |  | | int x,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=ins(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; | | |  | | } | | |

**Experiment 12**

|  |  |
| --- | --- |
|  | #include<iostream>  using namespace std; |
|  | void max\_heapify(int a[],int i,int n) |
|  | { |
|  | int l=2\*i+1; |
|  | int r=2\*i+2; |
|  | int largest; |
|  | if(l<n&&a[l]>a[i]) |
|  | largest=l; |
|  | else |
|  | largest=i; |
|  | if(r<n&&a[r]>a[largest]) |
|  | largest=r; |
|  | if(largest!=i) |
|  | { |
|  | swap(a[i],a[largest]); |
|  | max\_heapify(a,largest,n); |
|  | } |
|  | } |
|  | void build(int a[],int n) |
|  | { |
|  | for(int i=n/2-1;i>=0;i--) |
|  | max\_heapify(a,i,n); |
|  | } |
|  | void heapsort(int a[],int n) |
|  | { |
|  | int s=n; |
|  | build(a,n); |
|  | for(int i=n-1;i>=0;i--) |
|  | { |
|  | swap(a[0],a[i]); |
|  | s=s-1; |
|  | max\_heapify(a,0,s); |
|  | } |
|  | } |
|  | int main() |
|  | { |
|  | int n,i; |
|  | int a[20]; |
|  | cin>>n; |
|  | for(i=0;i<n;i++) |
|  | cin>>a[i]; |
|  | heapsort(a,n); |
|  | for(i=0;i<n;i++) |
|  | cout<<a[i]<<" "; |
|  |  |
|  | } |