**Q. Write a C++ program to implement double ended queue operations using linked list.**

#include <iostream>

#include <cstdlib>

using namespace std;

struct node

{

int info;

node \*next;

node \*prev;

}\*head, \*tail;

class dqueue

{

public:

int top1, top2;

void insert();

void del();

void display();

dqueue()

{

top1 = 0;

top2 = 0;

head = NULL;

tail = NULL;

}

};

int main()

{

int choice;

dqueue dl;

while (1)

{

cout<<"1.Insert Element into the Deque"<<endl;

cout<<"2.Delete Element from the Deque"<<endl;

cout<<"3.Traverse the Deque"<<endl;

cout<<"4.Quit"<<endl;

cout<<"Enter your Choice: ";

cin>>choice;

cout<<endl;

switch(choice)

{

case 1:

dl.insert();

break;

case 2:

dl.del();

break;

case 3:

dl.display();

break;

case 4:

exit(1);

break;

default:

cout<<"Wrong Choice"<<endl;

}

}

return 0;

}

void dqueue::insert()

{

struct node \*temp;

int ch, value;

if (top1 + top2 >= 50)

{

cout<<"Dequeue Overflow"<<endl;

return;

}

if (top1 + top2 == 0)

{

cout<<"Enter the value to be inserted: ";

cin>>value;

head = new (struct node);

head->info = value;

head->next = NULL;

head->prev = NULL;

tail = head;

top1++;

cout<<"Element Inserted into empty deque"<<endl;

}

else

{

while (1)

{

cout<<endl;

cout<<"1.Insert Element at first"<<endl;

cout<<"2.Insert Element at last"<<endl;

cout<<"3.Exit"<<endl;

cout<<endl;

cout<<"Enter Your Choice: ";

cin>>ch;

cout<<endl;

switch(ch)

{

case 1:

cout<<"Enter the value to be inserted: ";

cin>>value;

temp = new (struct node);

temp->info = value;

temp->next = head;

temp->prev = NULL;

head->prev = temp;

head = temp;

top1++;

break;

case 2:

cout<<"Enter the value to be inserted: ";

cin>>value;

temp = new (struct node);

temp->info = value;

temp->next = NULL;

temp->prev = tail;

tail->next = temp;

tail = temp;

top2++;

break;

case 3:

return;

break;

default:

cout<<"Wrong Choice"<<endl;

}

}

}

}

void dqueue::del()

{

if (top1 + top2 <= 0)

{

cout<<"Deque Underflow"<<endl;

return;

}

int ch;

while (1)

{

cout<<endl;

cout<<"1.Delete Element at first"<<endl;

cout<<"2.Delete Element at last"<<endl;

cout<<"3.Exit"<<endl;

cout<<endl;

cout<<"Enter Your Choice: ";

cin>>ch;

cout<<endl;

switch(ch)

{

case 1:

head = head->next;

head->prev = NULL;

top1--;

break;

case 2:

tail = tail->prev;

tail->next = NULL;

top2--;

break;

case 3:

return;

break;

default:

cout<<"Wrong Choice"<<endl;

}

}

}

void dqueue::display()

{

struct node \*temp;

int ch;

if (top1 + top2 <= 0)

{

cout<<"Deque Underflow"<<endl;

return;

}

while (1)

{

cout<<endl;

cout<<"1.Display Deque from Beginning"<<endl;

cout<<"2.Display Deque from End"<<endl;

cout<<"3.Exit"<<endl;

cout<<endl;

cout<<"Enter Your Choice: ";

cin>>ch;

cout<<endl;

switch (ch)

{

case 1:

temp = head;

cout<<"Deque from Beginning:"<<endl;

while (temp != NULL)

{

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

temp = temp->next;

}

cout<<endl;

break;

case 2:

cout<<"Deque from End:"<<endl;

temp = tail;

while (temp != NULL)

{

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

temp = temp->prev;

}

temp = tail;

cout<<endl;

break;

case 3:

return;

break;

default:

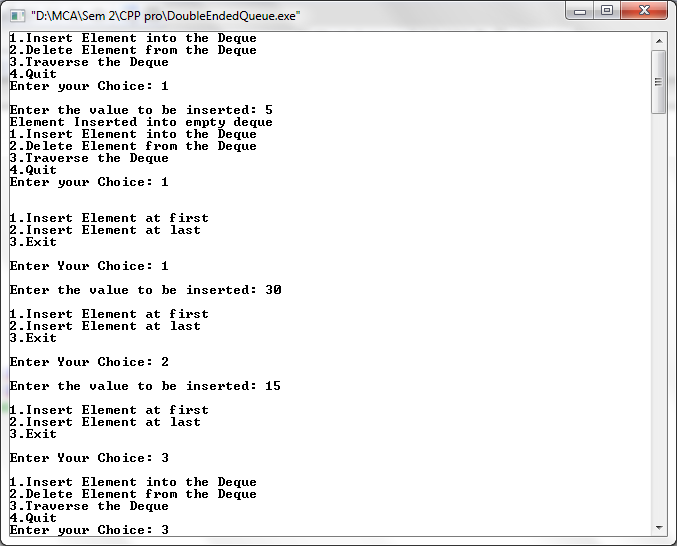
cout<<"Wrong Choice"<<endl;

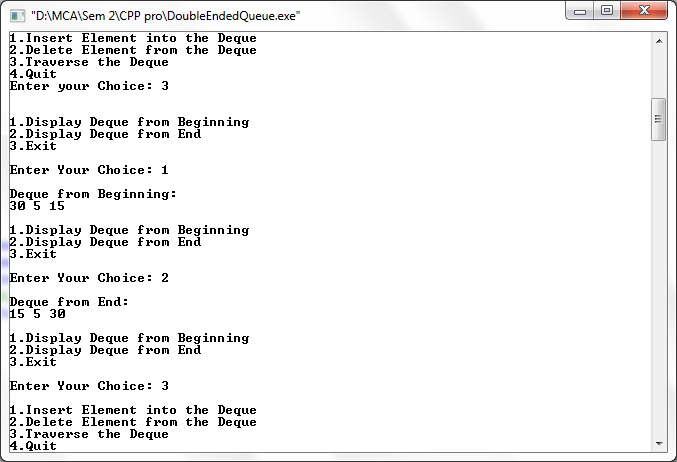
}

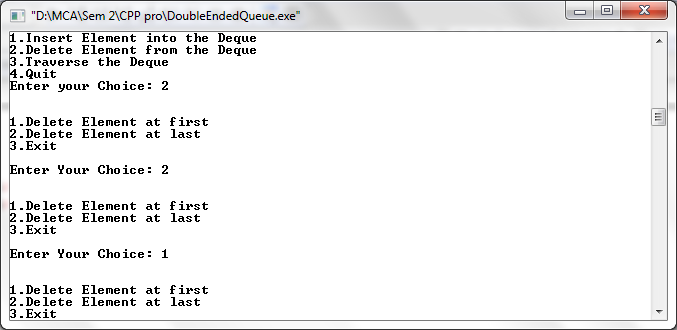
}

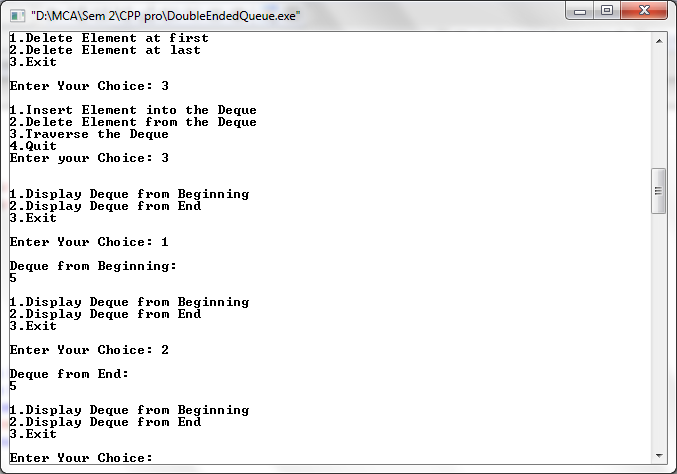
}

**Output –**



****

****

****

**Q. Write a C++ program to implement minimum spanning tree using Kruskal’s Algorithm.**

#include<iostream>

#include<stdlib.h>

#define max 30

using namespace std;

struct edge

{

int weight;

int u;

int v;

struct edge \*link;

};

struct edge \*frnt=NULL;

struct edge \*tmp;

int i,j,wt;

int father[max];

struct edge tree[max];

int wt\_tree;

int cnt=0;

void make\_tree();

void insert\_tree(int i, int j, int wt);

void insert\_pque(int i, int j, int wt);

struct edge \*del\_pque();

void create\_graph()

{

int i, n, max\_edges, origin, destin;

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

cin>>n;

max\_edges=n\*(n-1)/2;

for(i=1;i<max\_edges;i++)

{

cout<<"Enter edges (0 0 to quit) weight : ";

cin>>origin;

cin>>destin;

if((origin==0)&&(destin==0))

break;

cout<<"Enter weight for this edge : ";

cin>>wt;

if(origin>n || destin>n || origin<=0 || destin<=0)

{

cout<<"Invalid edge \n";

i--;

}

else

insert\_pque(origin,destin,wt);

}

if(i<n-1)

{

cout<<"Spanning tree is not possible \n";

exit(1);

}

}

void insert\_pque(int i,int j,int wt)

{

struct edge \*tmp, \*q;

tmp = (struct edge \*) malloc(sizeof(struct edge));

tmp->u=i;

tmp->v=j;

tmp->weight=wt;

if(frnt==NULL || tmp->weight<frnt->weight)

{

tmp->link=frnt;

frnt=tmp;

}

else

{

q=frnt;

while(q->link!=NULL&&q->link->weight<=tmp->weight)

q=q->link;

tmp->link=q->link;

q->link=tmp;

if(q->link==NULL)

tmp->link=NULL;

}

}

void make\_tree()

{

edge \*tmp;

int node1, node2, root\_n1, root\_n2, wt\_root=0,n,cnt=0;

while(cnt<n-1)

{

tmp=del\_pque();

node1=tmp->u;

node2=tmp->v;

cout<<"N1 ="<<node1;

cout<<"N2 ="<<node2;

while(node1>0)

{

root\_n1=node1;

node1=father[node1];

}

while(node2>0)

{

root\_n2=node2;

node2=father[node2];

}

cout<<"root N1= "<<root\_n1;

cout<<"root N2= "<<root\_n2;

if(root\_n1!=root\_n2)

{

insert\_tree(tmp->u,tmp->v,tmp->weight);

wt\_tree=wt\_tree+tmp->weight;

father[root\_n2]=root\_n1;

}

}

}

void insert\_tree(int i, int j, int wt)

{

cout<<"This edge inserted in the spanning tree \n";

cnt++;

tree[cnt].u=i;

tree[cnt].v=j;

tree[cnt].weight=wt;

}

struct edge \*del\_pque()

{

struct edge \* tmp;

tmp=frnt;

cout<<"Edge processed \n"<<tmp->u;

cout<<"Edge processed \n"<<tmp->v;

cout<<"Edge processed \n"<<tmp->weight;

frnt=frnt->link;

return tmp;

}

int main()

{

int i,j,wt\_tree,cnt=0;

struct edge tree[max];

create\_graph();

make\_tree();

cout<<"Edges to be included in spanning tree \n";

for(i=1;i<=cnt;i++)

{

cout<<tree[i].u;

cout<<tree[j].v;

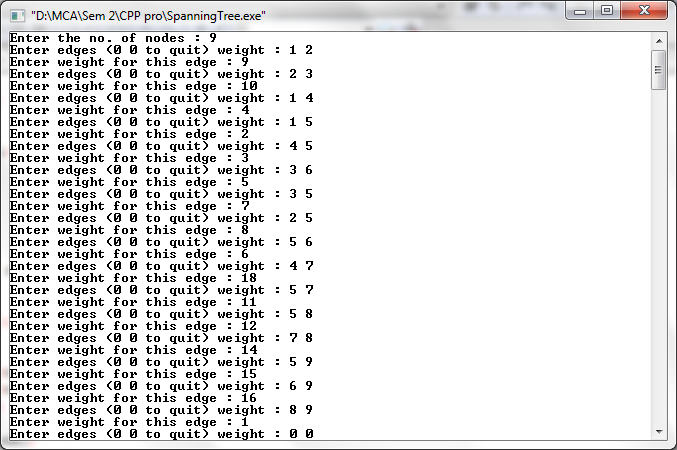
}

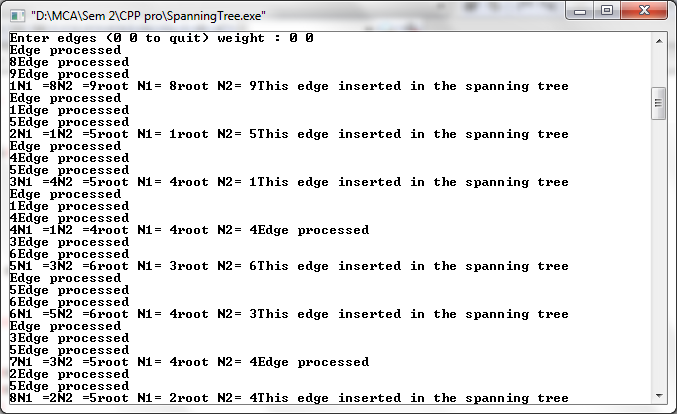
cout<<"Weight of this spanning tree is :"<<wt\_tree;

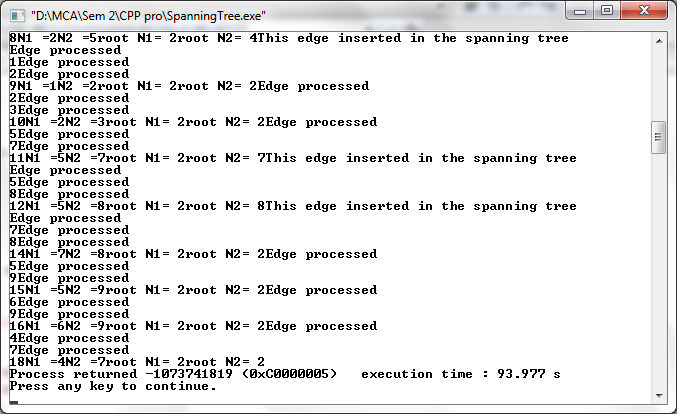
return 0;

}

**Output –**

****

****

****

**Q . Write a C++ program to perform Max Heap.**

#include <iostream>

#include <conio.h>

using namespace std;

void max\_heapify(int \*a, int i, int n)

{

int j, temp;

temp = a[i];

j = 2 \* i;

while (j <= n)

{

if (j < n && a[j+1] > a[j])

j = j + 1;

if (temp > a[j])

break;

else if (temp <= a[j])

{

a[j / 2] = a[j];

j = 2 \* j;

}

}

a[j/2] = temp;

return;

}

void build\_maxheap(int \*a,int n)

{

int i;

for(i = n/2; i >= 1; i--)

{

max\_heapify(a,i,n);

}

}

int main()

{

int n, i, x;

cout<<"enter no of elements of array\n";

cin>>n;

int a[20];

for (i = 1; i <= n; i++)

{

cout<<"enter element"<<(i)<<endl;

cin>>a[i];

}

build\_maxheap(a,n);

cout<<"Max Heap\n";

for (i = 1; i <= n; i++)

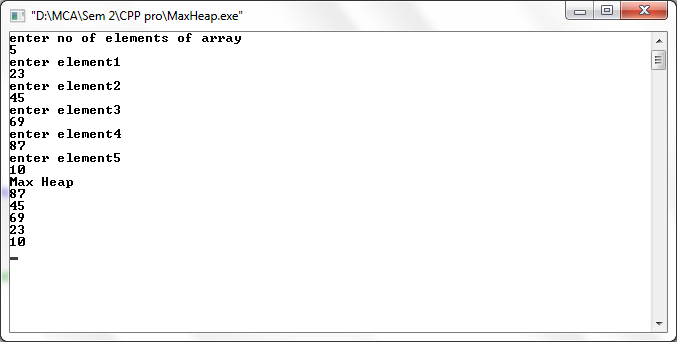
{

cout<<a[i]<<endl;

}

}

**Output –**

****

**Q . Write a C++ program to perform Min Heap.**

#include <iostream>

#include <conio.h>

using namespace std;

void min\_heapify(int \*a,int i,int n)

{

int j, temp;

temp = a[i];

j = 2 \* i;

while (j <= n)

{

if (j < n && a[j+1] < a[j])

j = j + 1;

if (temp < a[j])

break;

else if (temp >= a[j])

{

a[j/2] = a[j];

j = 2 \* j;

}

}

a[j/2] = temp;

return;

}

void build\_minheap(int \*a, int n)

{

int i;

for(i = n/2; i >= 1; i--)

{

min\_heapify(a,i,n);

}

}

int main()

{

int n, i, x;

cout<<"enter no of elements of array\n";

cin>>n;

int a[20];

for (i = 1; i <= n; i++)

{

cout<<"enter element"<<(i)<<endl;

cin>>a[i];

}

build\_minheap(a, n);

cout<<"Min Heap\n";

for (i = 1; i <= n; i++)

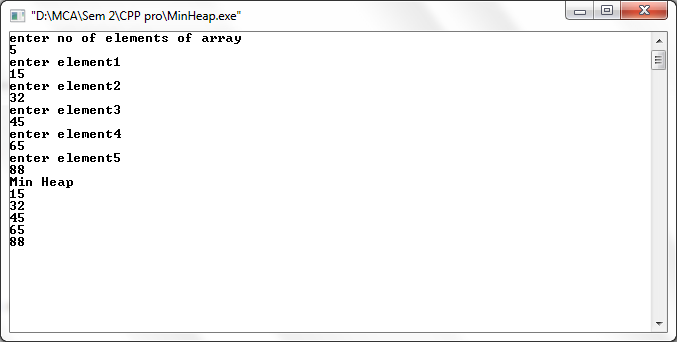
{

cout<<a[i]<<endl;

}

}

**Output –**

****

**Q. Write a C++ program to implement graph representation using Adjacency Matrix.**

#include<iostream>

using namespace std;

class adjMatrix

{

int \*\*adj;

bool \*visited;

int n,i,j;

public:

adjMatrix(int n)

{

this->n=n;

visited=new bool[n];

adj=new int \*[n];

for(i=1;i<=n;i++)

{

adj[i]=new int [n];

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

{

adj[i][j]=0;

}

}

}

int add\_edge(int origin, int dest)

{

if(origin>n||dest>n||origin<0||dest<0)

{

cout<<"Wrong nodes";

}

else

{

adj[origin][dest]=1;

}

}

int display()

{

for(i=1;i<=n;i++)

{

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

{

cout<<adj[i][j]<<"\t";

}

cout<<"\n";

}

}

};

int main()

{

int nodes, Max\_edges,i, origin, dest;

cout<<"Enter Maximum node: ";

cin>>nodes;

adjMatrix am(nodes);

Max\_edges=nodes\*(nodes-1);

cout<<"Enter -1 -1 to exit";

for(i=0;i<Max\_edges;i++)

{

cout<<"\nEnter edges: ";

cin>>origin>>dest;

if((origin==-1)&&(dest==-1))

break;

else

am.add\_edge(origin,dest);

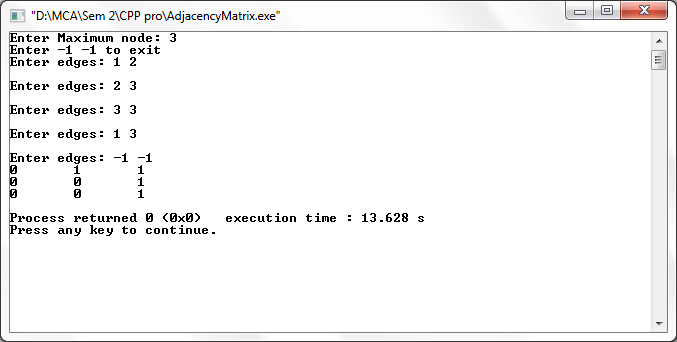
}

am.display();

return 0;

}

**Output –**

****

**Q. Write a C++ program to implement Binary Search Tree.**

# include <iostream>

# include <cstdlib>

using namespace std;

/\*Node Declaration\*/

struct node

{

int data;

struct node \*left;

struct node \*right;

}\*root;

/\*Class Declaration\*/

class BST

{

public:

void find1(int, node \*\*, node \*\*);

void insert1(node \*, node \*);

void del(int);

void case\_a(node \*,node \*);

void case\_b(node \*,node \*);

void case\_c(node \*,node \*);

void preorder(node \*);

void inorder(node \*);

void postorder(node \*);

void display(node \*, int);

BST()

{

root = NULL;

}

};

/\*Main Contains Menu\*/

int main()

{

int choice, num;

BST bst;

node \*temp;

while (1)

{

cout<<"\n1.Insert Element \n";

cout<<"2.Delete Element \n";

cout<<"3.Inorder Traversal \n";

cout<<"4.Preorder Traversal \n";

cout<<"5.Postorder Traversal \n";

cout<<"6.Display \n";

cout<<"7.Quit \n";

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

temp = new node;

cout<<"Enter the number to be inserted : ";

cin>>temp->data;

bst.insert1(root, temp);

break;

case 2:

if (root == NULL)

{

cout<<"Tree is empty, nothing to delete \n";

continue;

}

cout<<"Enter the number to be deleted : ";

cin>>num;

bst.del(num);

break;

case 3:

cout<<"Inorder Traversal:";

bst.inorder(root);

break;

case 4:

cout<<"Preorder Traversal:";

bst.preorder(root);

break;

case 5:

cout<<"Postorder Traversal:";

bst.postorder(root);

break;

case 6:

cout<<"Display BST:";

bst.display(root,1);

break;

case 7:

exit(1);

default:

cout<<"Wrong choice \n";

}

}

}

/\*Find Element in the Tree\*/

void BST::find1(int x, node \*\*par, node \*\*loc)

{

node \*ptr, \*ptrsave;

if (root == NULL)

{

\*loc = NULL;

\*par = NULL;

return;

}

if (x == root->data)

{

\*loc = root;

\*par = NULL;

return;

}

if (x < root->data)

ptr = root->left;

else

ptr = root->right;

ptrsave = root;

while (ptr != NULL)

{

if (x == ptr->data)

{

\*loc = ptr;

\*par = ptrsave;

return;

}

ptrsave = ptr;

if (x < ptr->data)

ptr = ptr->left;

else

ptr = ptr->right;

}

\*loc = NULL;

\*par = ptrsave;

}

/\*Inserting Element into the Tree\*/

void BST::insert1(node \*tree, node \*newnode)

{

if (root == NULL)

{

root = new node;

root->data = newnode->data;

root->left = NULL;

root->right = NULL;

cout<<" \t\t Root Node is Added";

return;

}

if (tree->data == newnode->data)

{

cout<<"Element already in the tree \n";

return;

}

if (tree->data > newnode->data)

{

if (tree->left != NULL)

{

insert1(tree->left, newnode);

}

else

{

tree->left = newnode;

(tree->left)->left = NULL;

(tree->left)->right = NULL;

cout<<" \t\t Node Added To Left";

return;

}

}

else

{

if (tree->right != NULL)

{

insert1(tree->right, newnode);

}

else

{

tree->right = newnode;

(tree->right)->left = NULL;

(tree->right)->right = NULL;

cout<<"\t\t Node Added To Right";

return;

}

}

}

/\*Delete Element from the tree\*/

void BST::del(int x)

{

node \*parent, \*location;

if (root == NULL)

{

cout<<"Tree empty";

return;

}

find1(x, &parent, &location);

if (location == NULL)

{

cout<<"x not present in tree";

return;

}

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

case\_a(parent, location);

if (location->left != NULL && location->right == NULL)

case\_b(parent, location);

if (location->left == NULL && location->right != NULL)

case\_b(parent, location);

if (location->left != NULL && location->right != NULL)

case\_c(parent, location);

free(location);

}

/\*Case A\*/

void BST::case\_a(node \*par, node \*loc )

{

if (par == NULL)

{

root = NULL;

}

else

{

if (loc == par->left)

par->left = NULL;

else

par->right = NULL;

}

}

/\*Case B\*/

void BST::case\_b(node \*par, node \*loc)

{

node \*child;

if (loc->left != NULL)

child = loc->left;

else

child = loc->right;

if (par == NULL)

{

root = child;

}

else

{

if (loc == par->left)

par->left = child;

else

par->right = child;

}

}

/\*Case C\*/

void BST::case\_c(node \*par, node \*loc)

{

node \*ptr, \*ptrsave, \*leaf1, \*parleaf1;

ptrsave = loc;

ptr = loc->right;

while (ptr->left != NULL)

{

ptrsave = ptr;

ptr = ptr->left;

}

leaf1 = ptr;

parleaf1 = ptrsave;

if (leaf1->left == NULL && leaf1->right == NULL)

case\_a(parleaf1, leaf1);

else

case\_b(parleaf1, leaf1);

if (par == NULL)

{

root = leaf1;

}

else

{

if (loc == par->left)

par->left = leaf1;

else

par->right = leaf1;

}

leaf1->left = loc->left;

leaf1->right = loc->right;

}

/\*Pre Order Traversal\*/

void BST::preorder(node \*ptr)

{

if (root == NULL)

{

cout<<"Tree is empty";

return;

}

if (ptr != NULL)

{

cout<<ptr->data<<" ";

preorder(ptr->left);

preorder(ptr->right);

}

}

/\*In Order Traversal\*/

void BST::inorder(node \*ptr)

{

if (root == NULL)

{

cout<<"Tree is empty";

return;

}

if (ptr != NULL)

{

inorder(ptr->left);

cout<<ptr->data<<" ";

inorder(ptr->right);

}

}

/\*Postorder Traversal\*/

void BST::postorder(node \*ptr)

{

if (root == NULL)

{

cout<<"Tree is empty";

return;

}

if (ptr != NULL)

{

postorder(ptr->left);

postorder(ptr->right);

cout<<ptr->data<<" ";

}

}

void BST::display(node \*ptr, int level)

{

int i;

if (ptr != NULL)

{

display(ptr->right, level+1);

cout<<endl;

if (ptr == root)

cout<<"Root->: ";

else

{

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

cout<<" ";

}

cout<<ptr->data;

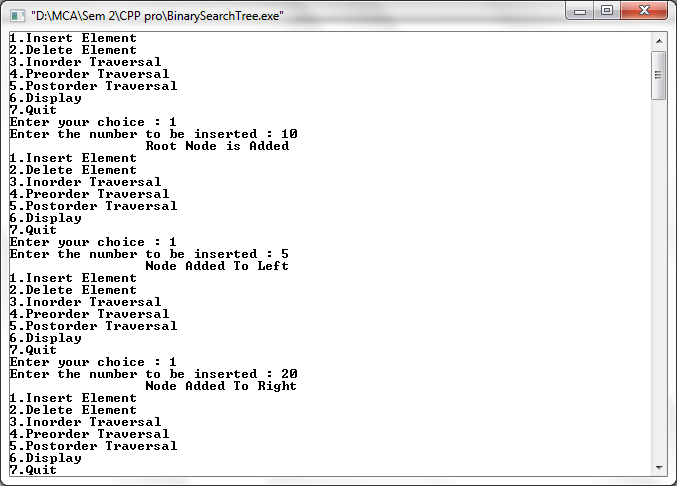
display(ptr->left, level+1);

}

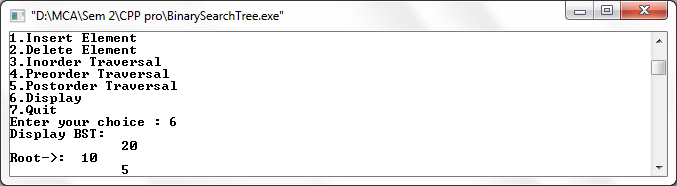
}

**Output –**

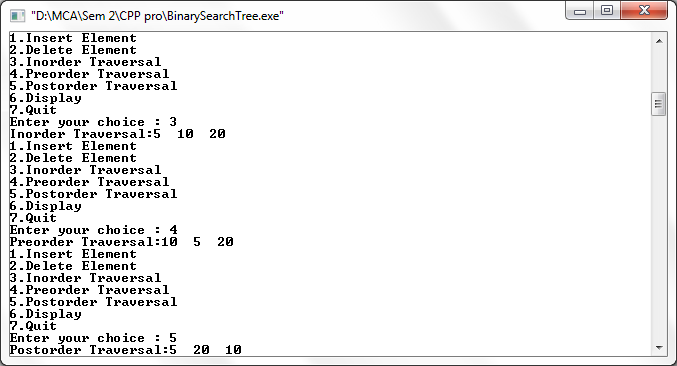
Insert



Display



Inorder Traversal, Preorder traversal & Postorder Traversal:



**Q. Write a C++ program to implement select kth element from heap.**

#include <iostream>

#include<cstdlib>

using namespace std;

int array1[100], n;

void display()

{

int i;

if (n == 0)

{

cout<<"Heap is empty \n";

return;

}

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

cout<<array1[i]<<"\t";

cout<<"\n";

}/\*End of display()\*/

void insert1(int num, int location)

{

int parentnode;

while (location > 0)

{

parentnode =(location - 1)/2;

if (num <= array1[parentnode])

{

array1[location] = num;

return;

}

array1[location] = array1[parentnode];

location = parentnode;

}/\*End of while\*/

array1[0] = num; /\*assign number to the root node \*/

}/\*End of insert()\*/

void delete1(int num)

{

int left, right, i, temp, parentnode;

for (i = 0; i < num; i++) {

if (num == array1[i])

break;

}

if (num != array1[i])

{

cout<<" not found in heap list\n"<< num;

return;

}

array1[i] = array1[n - 1];

n = n - 1;

parentnode =(i - 1) / 2; /\*find parentnode of node i \*/

if (array1[i] > array1[parentnode])

{

insert1(array1[i], i);

return;

}

left = 2 \* i + 1; /\*left child of i\*/

right = 2 \* i + 2; /\* right child of i\*/

while (right < n)

{

if (array1[i] >= array1[left] && array1[i] >= array1[right])

return;

if (array1[right] <= array1[left])

{

temp = array1[i];

array1[i] = array1[left];

array1[left] = temp;

i = left;

}

else

{

temp = array1[i];

array1[i] = array1[right];

array1[right] = temp;

i = right;

}

left = 2 \* i + 1;

right = 2 \* i + 2;

}/\*End of while\*/

if (left == n - 1 && array1[i]) {

temp = array1[i];

array1[i] = array1[left];

array1[left] = temp;

}

}

int main()

{

//int array[100], n;

int choice, num;

n = 0;/\*Represents number of nodes in the heap\*/

while(choice!=0)

{

cout<<"1.Insert the element \n";

cout<<"2.Delete the element \n";

cout<<"3.Display all elements \n";

cout<<"4.Quit \n";

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter the element to be inserted to the list : ";

cin>>num;

insert1(num, n);

n = n + 1;

break;

case 2:

cout<<"Enter the elements to be deleted from the list: ";

cin>>num;

delete1(num);

break;

case 3:

display();

break;

case 4:

exit(0);

default:

cout<<"Invalid choice \n";

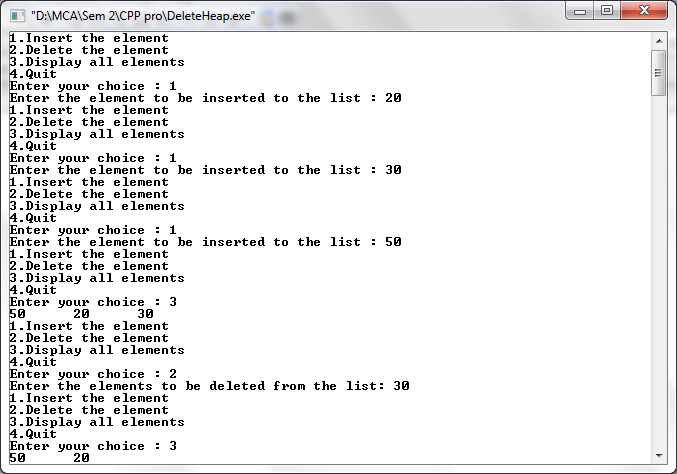
}/\*End of switch \*/

}/\*End of while \*/

return 0;

}/\*End of main()\*/

**Output –**



**Q. Write a C++ program to perform disk file searching.**

#include<iostream>

#include<conio.h>

#include<fstream>

#include<string.h>

using namespace std;

class File

{

char name[20];

public:

void getName()

{

cin>>name;

}

void writeToFile()

{

ofstream out;

out.open("123.txt",ios::out | ios::app);

out<<name<<endl;

}

void searchFromFile(char sname[20])

{

int flag=0;

ifstream in;

in.open("123.txt",ios::in);

cout<<"Searching for the Name : "<<sname<<endl;

for(int i=0;i<10;i++)

{

in>>name;

if(strcmp(sname,name)==0)

{

cout<<"The name "<<sname<<" was found successfully : "<<endl;

flag=1;

break;

}

}

if(flag==0)

cout<<"The name "<<sname<<" could not be find"<<endl;

}

};

int main()

{

File f;

for (int i=0;i<10;i++)

{

f.getName();

f.writeToFile();

}

char name[20];

cout<<"Enter a name : "<<endl;

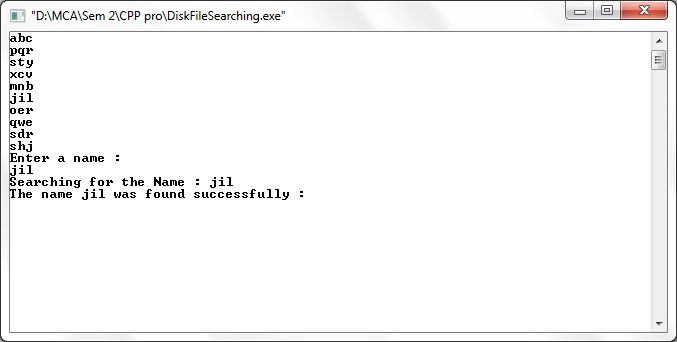
cin>>name;

f.searchFromFile(name);

getch();

return 0;}

**Output –**

****

**Q. Write a C++ program to implement shortest path using Warshall’s Algorithm.**

#include<iostream>

#include<stdlib.h>

#define max 10

#define infi 999

using namespace std;

int p[max][max];

/\*All pairs shortest path\*/

void allpairshort(int a[max][max], int n)

{

int i, j, k;

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

{

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

{

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

{

if(a[i][k]+a[k][j]<a[i][j])

{

a[i][j]=a[i][k]+a[k][j];

p[i][j]=k;

}

}

}

}

}

/\*Storing the shortest path\*/

void shortest(int i, int j)

{

int k=p[i][j];

if(k>0)

{

shortest(i,k);

cout<<" "<<k<<" ";

shortest(k,j);

}

}

/\*Display the shortest path\*/

void findpath(int a[max][max], int i, int j, int n)

{

cout<<"Path from "<<i<<" to "<<j<<":";

if(a[i][j]<infi)

{

cout<<" "<<i<<" ";

shortest(i,j);

cout<<" "<<j<<" ";

}

cout<<"\n";

}

int main()

{

int i, j;

int a[][10]={{0, 10, infi, 30, 100},

{infi, 0, 50, infi, infi},

{infi, infi, 0, infi, 10},

{infi, infi, 20, 0, 60},

(infi, infi, infi, infi, 0)};

allpairshort(a, 5);

findpath(a, 0, 4, 5);

findpath(a, 0, 3, 5);

return 0;

}

**Output –**

