1. **Write a C++ program for implementing Singly Linked list.**

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int info;

struct node \*next;

}\*start;

class single\_llist

{

public:

node\* create\_node(int);

void insert\_begin();

void insert\_last();

void insert\_pos();

void delete\_begin();

void delete\_last();

void delete\_pos();

void update\_begin();

void update\_last();

void update\_pos();

void sort();

void reverse();

void search();

void display();

single\_list()

{

start=NULL;

}

};

int main()

{

int choice;

single\_llist s1,s2;

start=NULL;

do

{

cout<<"-----------------"<<endl;

cout<<"Operattions on singly linked list"<<endl;

cout<<"-----------------"<<endl;

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

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

cout<<"3.Insert at position"<<endl;

cout<<"4.Delete at first"<<endl;

cout<<"5.Delete at Last"<<endl;

cout<<"6.Delete at position"<<endl;

cout<<"7.Update at first"<<endl;

cout<<"8.Update at last"<<endl;

cout<<"9.Update at position"<<endl;

cout<<"10.Ascending order"<<endl;

cout<<"11.Descending order"<<endl;

cout<<"12.Search"<<endl;

cout<<"13.Display"<<endl;

cout<<"14.Exit"<<endl;

cout<<"Enter your choice:";

cin>>choice;

switch(choice)

{

case 1: s1.insert\_begin();

s1.display();

break;

case 2: s1.insert\_last();

s1.display();

break;

case 3: s1.insert\_pos();

s1.display();

break;

case 4: s2.delete\_begin();

s1.display();

break;

case 5: s2.delete\_last();

s1.display();

break;

case 6: s1.delete\_pos();

s1.display();

break;

case 7: s1.update\_begin();

s1.display();

break;

case 8: s1.update\_last();

s1.display();

break;

case 9: s1.update\_pos();

s1.display();

break;

case 10: s1.sort();

s1.display();

break;

case 11: s1.reverse();

s1.display();

break;

case 12: s1.search();

s1.display();

break;

case 13:

s1.display();

break;

case 14: exit(0);

break;

default:cout<<"wrong choice...???"<<endl;

break;

}

}

while(choice!=14);

}

node \*single\_llist::create\_node(int value)

{

struct node \*temp, \*s;

temp=new(struct node);

if(temp==NULL)

{

cout<<"Memory not allocated"<<endl;

return 0;

}

else

{

temp->info=value;

temp->next=NULL;

return temp;

}

}

void single\_llist::insert\_begin()

{

int value;

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

cin>>value;

struct node \*temp, \*s;

temp = create\_node(value);

if(start==NULL)

{

start=temp;

start->next=NULL;

cout<<temp->info<<"is inserted at first in the empty list"<<endl;

}

else

{

s=start;

start=temp;

start->next=s;

cout<<temp->info<<"is inserted at first"<<endl;

}

}

void single\_llist::insert\_last()

{

int value;

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

cin>>value;

struct node \*temp, \*s;

temp = create\_node(value);

if(start==NULL)

{

start=temp;

start->next=NULL;

cout<<temp->info<<"is inserted at last in the empty list"<<endl;

}

else

{

s=start;

while(s->next!=NULL)

{

s=s->next;

}

temp->next=NULL;

s->next=temp;

cout<<temp->info<<"is inserted at last"<<endl;

}

}

void single\_llist::insert\_pos()

{

int value, pos, counter = 0, loc = 1;

struct node \*temp, \*s, \*ptr;

s = start;

while (s != NULL)

{

s = s->next;

counter++;

}

if (counter == 0){}

else

{

cout<<"Enter the postion from "<<loc<<" to "<<counter+1<<" : ";

cin>>pos;

s = start;

if(pos == 1)

{

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

cin>>value;

temp=create\_node(value);

start=temp;

start->next=s;

cout<<temp->info<<"is inserted at first"<<endl;

}

else if(pos>1 && pos<=counter)

{

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

cin>>value;

temp=create\_node(value);

for(int i=1;i<pos;i++)

{

ptr=s;

s=s->next;

}

ptr->next=temp;

temp->next=s;

cout<<temp->info<<"is inserted at position"<<pos<<endl;

}

else if(pos== counter+1)

{

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

cin>>value;

temp=create\_node(value);

while(s->next!=NULL)

{

s=s->next;

}

temp->next=NULL;

s->next=temp;

cout<<temp->info<<"is inserted at last"<<endl;

}

else

{

cout<<"Position out of range...!!!"<<endl;

}

}

}

void single\_llist::delete\_begin()

{

if(start==NULL){}

else

{

struct node \*s, \*ptr;

s=start;

start=s->next;

cout<<s->info<<"deleted from first"<<endl;

free(s);

}

}

void single\_llist::delete\_last()

{

int i, counter=0;

struct node \*s, \*ptr;

if(start==NULL){}

else

{

s=start;

while(s!=NULL)

{

s=s->next;

counter++;

}

s=start;

if(counter==1)

{

start=s->next;

cout<<s->info<<"Deleted from last"<<endl;

free(s);

}

else

{

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

{

ptr=s;

s=s->next;

}

ptr->next=s->next;

cout<<s->info<<"deleted from last"<<endl;

free(s);

}

}

}

void single\_llist::delete\_pos()

{

int pos, i, counter = 0, loc = 1;

struct node \*s, \*ptr;

s = start;

while (s != NULL)

{

s = s->next;

counter++;

}

if(counter==0){}

else

{

if(counter==1)

{

cout<<"Enter the postion [SAY"<<loc<<"]:";

cin>>pos;

s=start;

if(pos==1)

{

start=s->next;

cout<<s->info<<"Deleted from first"<<endl;

free(s);

}

else

cout<<"Position out of range....!!!"<<endl;

}

else

{

cout<<"Enter the position from"<<loc<<"to"<<counter<<":";

cin>>pos;

s=start;

if(pos==1)

{

start=s->next;

cout<<s->info<<"deleted from first"<<endl;

free(s);

}

else if(pos>1 && pos<=counter)

{

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

{

ptr=s;

s=s->next;

}

ptr->next=s->next;

if(pos==counter)

{cout<<s->info<<"deleted from last"<<endl;

free(s);}

else

{cout<<s->info<<"deleted from postion"<<pos<<endl;

free(s);}

}

else

cout<<"Position out of range...!!!"<<endl;

}

}

}

void single\_llist::update\_begin()

{

int value, pos=1,i,counter=0;

struct node \*s, \*ptr;

s=start;

while(s!=NULL)

{

s=s->next;

counter++;

}

if(counter==0){}

else if(pos==1)

{

cout<<"Enter the new node:";

cin>>value;

start->info=value;

cout<<"Node updated at first position : "<<pos<<" = "<<start->info<<endl;

}

}

void single\_llist::update\_last()

{

int value, pos, i, counter=0;

struct node \*s, \*ptr;

s=start;

while(s!=NULL)

{

s=s->next;

counter++;

}

s=start;

if(counter==0){}

else

{

cout<<"Enter the new node:";

cin>>value;

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

{

s=s->next;

}

s->info=value;

cout<<"Node updated at last position:"<<counter<<"="<<s->info<<endl;

}

}

void single\_llist::update\_pos()

{

int value, pos, i,counter = 0, loc = 1;

struct node \*s, \*ptr;

s = start;

while (s != NULL)

{

s=s->next;

counter++;

}

if(counter==0){}

else

{

if(counter==1)

{

cout<<"Enter the position[SAY"<<loc<<"];";

cin>>pos;

s = start;

if (pos == 1)

{

cout<<"Enter the new node : ";

cin>>value;

start->info = value;

cout<<"Node updated at position : "<<pos<<" = "<<start->info<<endl;

}

else

cout<<"Position out of range...!!!"<<endl;

}

else

{

cout<<"Enter the position from "<<loc<<"to"<<counter<<":";

cin>>pos;

s=start;

if(pos==1)

{

cout<<"Enter the new node:";

cin>>value;

start->info=value;

cout<<"Node updated at position:"<<pos<<"="<<start->info<<endl;

}

else if(pos>1 && pos<=counter)

{

cout<<"Enter the new node:";

cin>>value;

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

{

s = s->next;

}

s->info = value;

cout<<"Node updated at position : "<<pos<<" = "<<s->info<<endl;

}

else

cout<<"Position out of range...!!!"<<endl;

}

}

}

void single\_llist::sort()

{

struct node \*ptr, \*s;

int value;

if (start == NULL){}

else

{

ptr = start;

while (ptr != NULL)

{

for (s = ptr->next;s !=NULL;s = s->next)

{

if (ptr->info > s->info)

{

value = ptr->info;

ptr->info = s->info;

s->info = value;

}

}

ptr = ptr->next;

}

}

}

void single\_llist::reverse()

{

struct node \*ptr, \*s;

int value;

if (start == NULL){}

else

{

ptr = start;

while (ptr != NULL)

{

for (s = ptr->next;s !=NULL;s = s->next)

{

if (ptr->info < s->info)

{

value = ptr->info;

ptr->info = s->info;

s->info = value;

}

}

ptr = ptr->next;

}

}

}

void single\_llist::search()

{

int value, loc = 0, pos = 0, counter = 0;

struct node \*s;

s = start;

while (s != NULL)

{

s = s->next;

counter++;

}

if (start == NULL){}

else

{

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

cin>>value;

struct node \*s;

s = start;

while (s != NULL)

{

pos++;

if (s->info == value)

{

loc++;

if(loc == 1)

cout<<"Element "<<value<<" is found at position "<<pos;

else if(loc <= counter)

cout<<" , "<<pos;

}

s = s->next;

}

cout<<endl;

if (loc == 0)

cout<<"Element "<<value<<" not found in the list"<<endl;

}

}

void single\_llist::display()

{

struct node \*temp;

if (start == NULL)

cout<<"Linked list is empty....!!!"<<endl;

else

{

cout<<"Linked Lsit conatains:";

temp = start;

while (temp != NULL)

{

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

temp= temp->next;

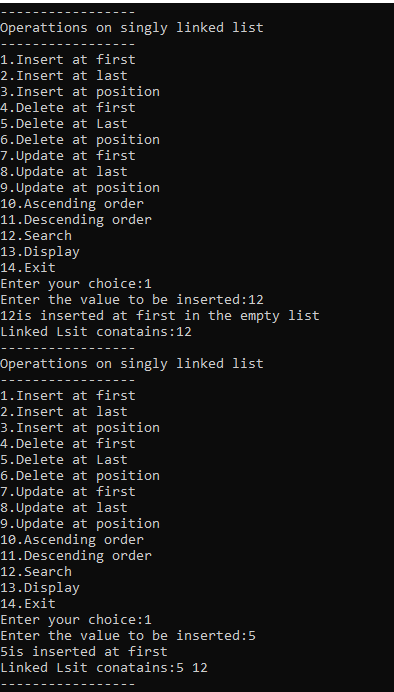
}

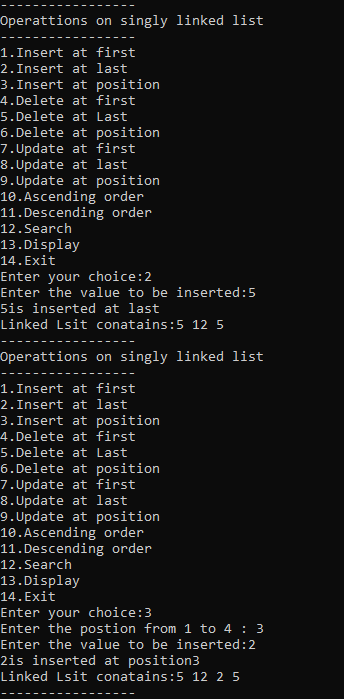
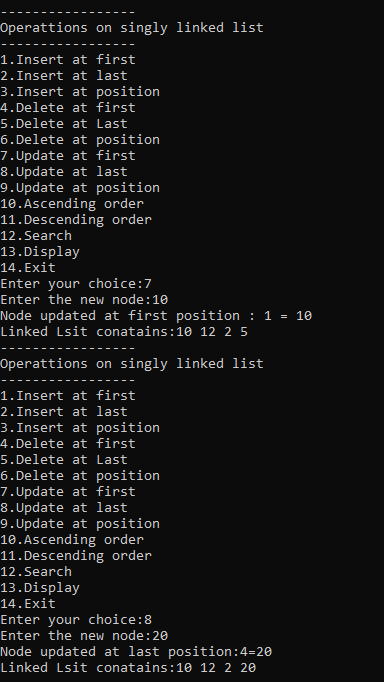
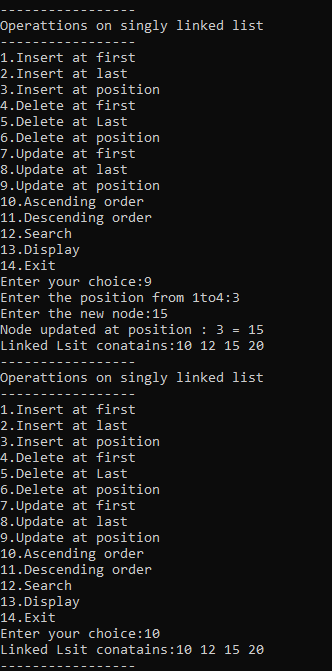
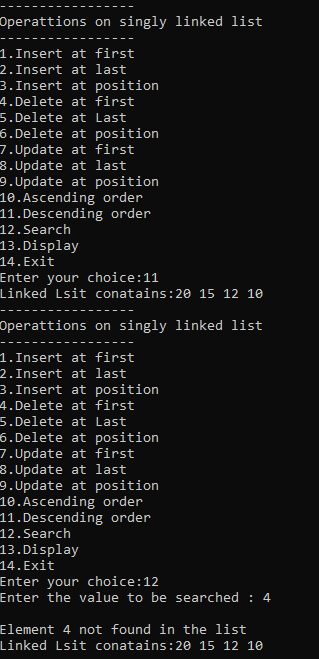
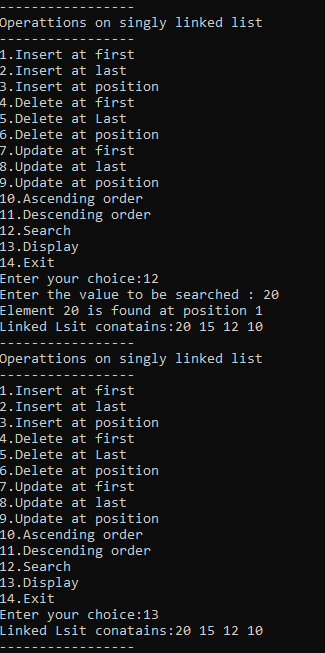
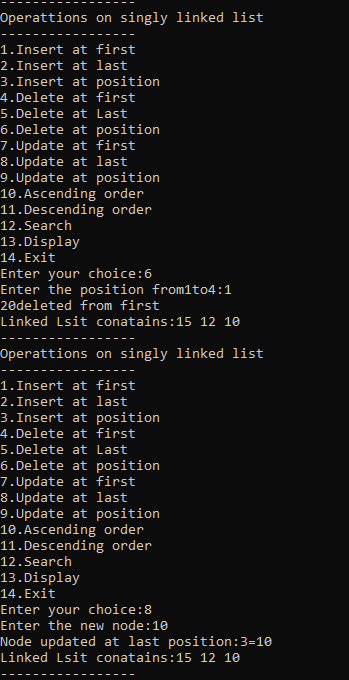
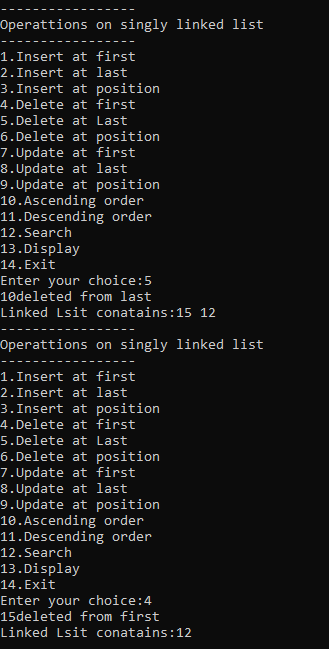
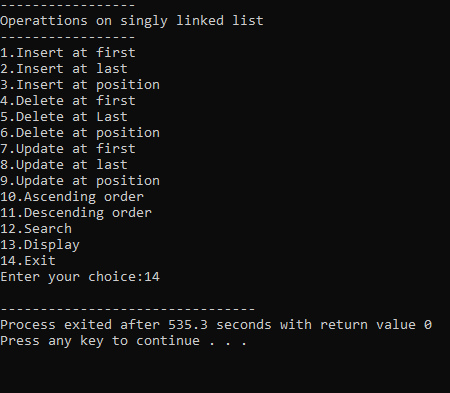
cout<<endl;

}

}

**OUTPUT:**

****

**2. Write a C++ program to split the linked list into two halves such that the element ‘e’ should be the first element of second list.**

#include<iostream>

using namespace std;

struct Node{

int value;

struct Node\*next;

};

struct Node\*head=NULL;

struct Node\*sHead=NULL;

struct Node\*temp=NULL;

void insert(int new\_data){

struct Node \*new\_node=new Node();

new\_node->value=new\_data;

new\_node->next=head;

head=new\_node;

}

int n;

int ele;

int splitIndex;

int main(){

int i;

cout<<"Enter number of elements you want in the list:\t";

cin>>n;

cout<<"Enter elements:"<<endl;

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

cin>>ele;

insert(ele);

}

cout<<"\n list of element:"<<endl;

Node \*t;

t=head;

while(t!=NULL){

cout<<t->value<<"\t";

t=t->next;

}

cout<<"\n\n Enter the position you want the list to split:";

cin>>splitIndex;

while(splitIndex<0 || splitIndex>n-1){

cout<<"Invalid position.Try again."<<endl;

cin>>splitIndex;

}

temp=head;

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

if(i==splitIndex-1){

Node \*tN;

tN=temp->next;

sHead=tN;

temp->next=NULL;

break;

}

temp=temp->next;

}

temp=head;

if(temp==NULL){

cout<<"\n FIrst list is empty"<<endl;

}else{

cout<<"\n \n First list element"<<endl;

while(temp!=NULL){

cout<<temp->value<<"\t";

temp=temp->next;

}

}

temp=sHead;

if(temp==NULL){

cout<<"\nSecond list is empty"<<endl;

}else{

cout<<"\n\nSecond list elements "<<endl;

while(temp != NULL){

cout<<temp->value<<"\t";

temp = temp->next;

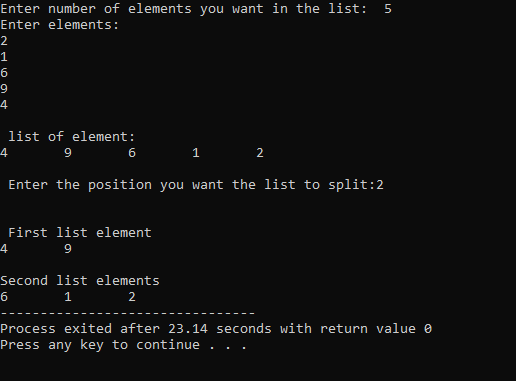
}

}

return 0;

}

**OUTPUT:**

****

**3.** [**Write a C++ program to check if a binary tree is BST or not**](https://www.geeksforgeeks.org/a-program-to-check-if-a-binary-tree-is-bst-or-not/)**.**

#include<bits/stdc++.h>

using namespace std;

struct Node

{

int data;

struct Node \*left, \*right;

Node(int data)

{

this->data=data;

left=right=NULL;

}

};

bool isBSTUtil(struct Node\*root, Node\*&prev)

{

if(root)

{

if(!isBSTUtil(root->left, prev))

return false;

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

return false;

prev=root;

return isBSTUtil(root->right, prev);

}

return true;

}

bool isBST(Node \*root)

{

Node \*prev=NULL;

return isBSTUtil(root,prev);

}

int main()

{

// struct Node \*root=new Node(200);

// root->left=new Node(6);

// root->left->right=new Node(80);

// root->left->right->left=new Node(9);

// root->left->right->right=new Node(100);

// root->left->right->right->right=new Node(150);

// root->left->right->left->left=new Node(7);

// root->left->right->left->right=new Node(30);

// root->left->right->left->right->left=new Node(17);

// root->left->right->left->right->right=new Node(65);

// root->left->right->left->right->right->left=new Node(58);

struct Node \*root = new Node(7);

root->left = new Node(5);

root->right = new Node(8);

root->left->left = new Node(3);

root->left->right = new Node(6);

if(isBST(root))

cout<<"Is BST";

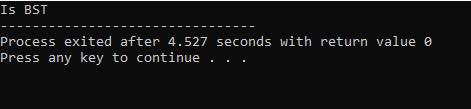
else

cout<<"Not a BST";

return 0;

}

**OUTPUT:**



**4. Construct a binary search tree (BST) to support the following operations.**

**Given a key, perform a search in the BST. If the key is found then display “key found”.**

* **Insert an element into a binary search tree.**
* **Delete an element from a binary search tree.**

**Display the tree using inorder, preorder and post order traversal methods(a).**

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int info;

struct node\*left;

struct node\*right;

}\*root;

class BST

{

public:

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

void insert(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;

}

};

int main()

{

int choice,num;

BST bst;

node \*temp;

while(1)

{

cout<<"------"<<endl;

cout<<"Operations on BST"<<endl;

cout<<"------"<<endl;

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

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

cout<<"3.Inorder Traversal"<<endl;

cout<<"4.Preorder Traversal"<<endl;

cout<<"5.Postorder Traversal"<<endl;

cout<<"6.Display"<<endl;

cout<<"7.Quit"<<endl;

cout<<"Enter your choice:";

cin>>choice;

switch(choice)

{

case 1:

temp=new node;

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

cin>>temp->info;

bst.insert(root,temp);

break;

case 2:

if(root==NULL)

{

cout<<"Tree is empty,nothing to delete"<<endl;

continue;

}

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

cin>>num;

bst.del(num);

break;

case 3:

cout<<"Inorder Traversal of BST:"<<endl;

bst.inorder(root);

cout<<endl;

break;

case 4:

cout<<"Preorder Traversal of BST:"<<endl;

bst.preorder(root);

cout<<endl;

break;

case 5:

cout<<"Postorder Traversal of BST:"<<endl;

bst.postorder(root);

cout<<endl;

break;

case 6:

cout<<"Display BST:"<<endl;

bst.display(root,1);

cout<<endl;

break;

case 7:

exit(1);

default:

cout<<"Wrong choice"<<endl;

}

}

}

void BST::find(int item, node \*\*par,node \*\*loc)

{

node \*ptr, \*ptrsave;

if(root==NULL)

{

\*loc=NULL;

\*par=NULL;

return;

}

if(item==root->info)

{

\*loc=root;

\*par=NULL;

return;

}

if(item<root->info)

ptr=root->left;

else

ptr=root->right;

ptrsave=root;

while(ptr!=NULL)

{

if(item==ptr->info)

{

\*loc=ptr;

\*par=ptrsave;

return;

}

ptrsave = ptr;

if (item < ptr->info)

ptr = ptr->left;

else

ptr = ptr->right;

}

\*loc=NULL;

\*par=ptrsave;

}

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

{

if(root==NULL)

{

root=new node;

root->info=newnode->info;

root->left=NULL;

root->right=NULL;

cout<<"Root Node is Added"<<endl;

return;

}

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

{

cout<<"Element already in the tree"<<endl;

return;

}

if(tree->info>newnode->info)

{

if(tree->left!=NULL)

{

insert(tree->left,newnode);

}

else

{

tree->left=newnode;

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

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

cout<<"Node Added To Left"<<endl;

return;

}

}

else

{

if(tree->right!=NULL)

{

insert(tree->right,newnode);

}

else

{

tree->right = newnode;

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

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

cout<<"Node Added To Right"<<endl;

return;

}

}

}

void BST::del(int item)

{

node \*parent,\*location;

if(root==NULL)

{

cout<<"Tree empty"<<endl;

return;

}

find(item, &parent,&location);

if(location==NULL)

{

cout<<"Item not present in tree"<<endl;

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);

}

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;

}

}

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;

}

}

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

{

node \*ptr, \*ptrsave, \*suc, \*parsuc;

ptrsave = loc;

ptr = loc->right;

while (ptr->left != NULL)

{

ptrsave = ptr;

ptr = ptr->left;

}

suc = ptr;

parsuc = ptrsave;

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

case\_a(parsuc, suc);

else

case\_b(parsuc, suc);

if(par==NULL)

{

root=suc;

}

else

{

if(loc==par->left)

par->left=suc;

else

par->right=suc;

}

suc->left=loc->left;

suc->right=loc->right;

}

void BST::preorder(node \*ptr)

{

if (root == NULL)

{

cout<<"Tree is empty"<<endl;

return;

}

if (ptr != NULL)

{

cout<<ptr->info<<" ";

preorder(ptr->left);

preorder(ptr->right);

}

}

void BST::inorder(node \*ptr)

{

if(root==NULL)

{

cout<<"Tree is empty"<<endl;

return;

}

if(ptr!=NULL)

{

inorder(ptr->left);

cout<<ptr->info<<" ";

inorder(ptr->right);

}

}

void BST::postorder(node \*ptr)

{

if(root==NULL)

{

cout<<"Tree is empty"<<endl;

return;

}

if(ptr!=NULL)

{

postorder(ptr->left);

postorder(ptr->right);

cout<<ptr->info<<" ";

}

}

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<<" ";

}

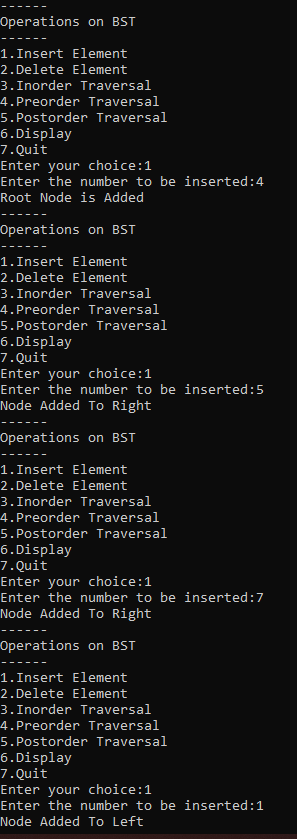
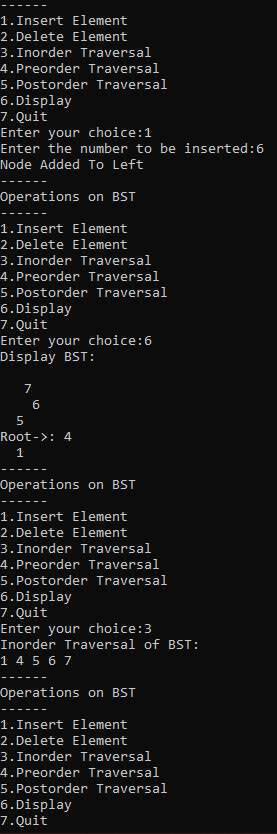
cout<<ptr->info;

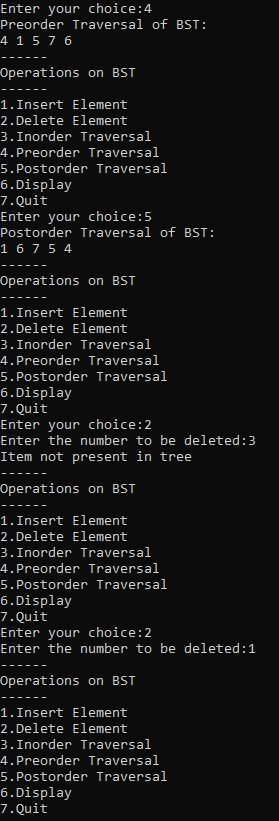
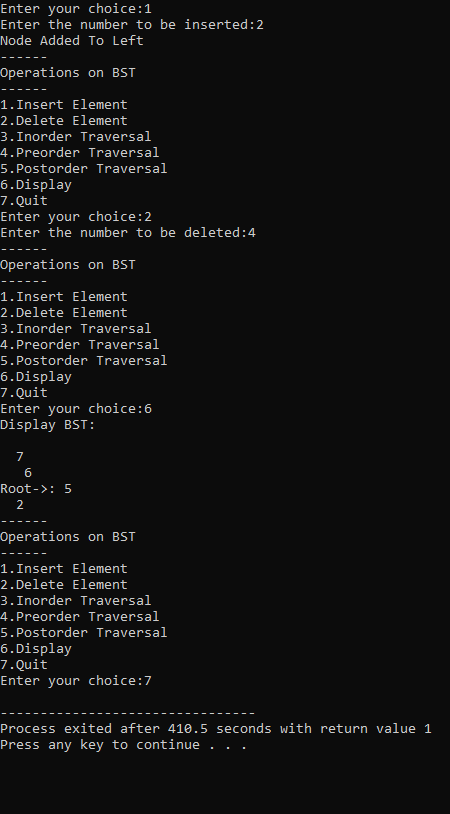
display(ptr->left, level+1);

}

}

**OUTPUT:**

** **

** **

**5. Write a program to implement Minheap.**

#include<iostream>

using namespace std;

void MinHeapify(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--)

MinHeapify(a,i,n);

}

int main()

{

int n,i,arr[100];

cout<<"\n Enter the number of data element to be sorted:";

cin>>n;

n++;

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

{

cout<<"Enter element"<<i<<":";

cin>>arr[i];

}

Build\_MinHeap(arr,n-1);

cout<<"\n Min Heap";

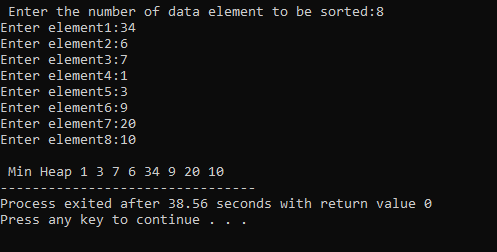
for(i=1;i<n;i++)

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

return 0;

}

**OUTPUT:**

****

**6. Write a program to implement Max heap.**

#include<iostream>

using namespace std;

void MaxHeapify(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--)

MaxHeapify(a,i,n);

}

int main()

{

int n,i,arr[100];

cout<<"\n Enter the number of data element to be sorted:";

cin>>n;

n++;

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

{

cout<<"Enter element"<<i<<":";

cin>>arr[i];

}

Build\_MaxHeap(arr,n-1);

cout<<"\n Max Heap";

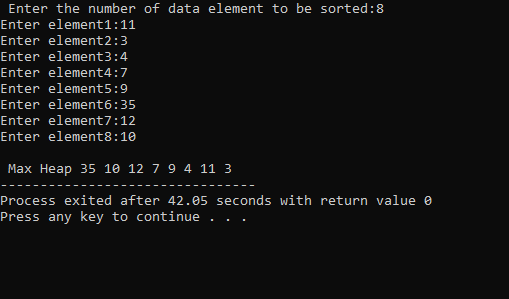
for(i=1;i<n;i++)

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

return 0;

}

**OUTPUT:**

****

**7. Write a C++ program for implementing Heap sort technique.**

#include<iostream>

using namespace std;

void MaxHeapify(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 HeapSort(int a[],int n)

{

int i,temp;

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

{

temp=a[i];

a[i]=a[1];

a[1]=temp;

MaxHeapify(a,1,i-1);

}

}

void Build\_MaxHeap(int a[],int n)

{

int i;

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

MaxHeapify(a,i,n);

}

int main()

{

int n,i,arr[100];

cout<<"\n Enter the number of data element to be sorted:";

cin>>n;

n++;

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

{

cout<<"Enter element"<<i<<":";

cin>>arr[i];

}

Build\_MaxHeap(arr,n-1);

HeapSort(arr,n-1);

cout<<"\n Sorted Data";

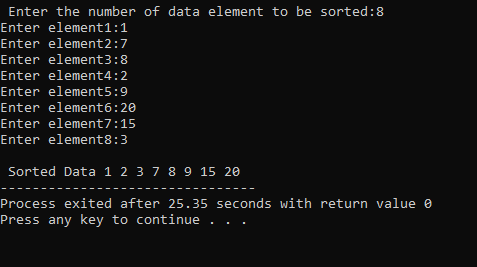
for(i=1;i<n;i++)

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

return 0;

}

**OUTPUT:**

****

**8. C++ program to implement sum of subsets using backtracking.**

#include <iostream>

#include<limits.h>

using namespace std;

class Subset

{

public:

void printSum(int result[], int front, int tail)

{

cout << "{";

for (int i = front; i < tail; ++i)

{

if (result[i] != INT\_MAX)

{

cout << " " << result[i] << " ";

}

}

cout << "}\n";

}

void subsetSum(int arr[], int result[], int sum, int size, int current\_sum, int location)

{

if (location == -1)

{

return;

}

this->subsetSum(arr, result, sum, size, current\_sum, location - 1);

result[location] = arr[location];

if (current\_sum + arr[location] == sum)

{

this->printSum(result, location, size);

}

this->subsetSum(arr, result, sum, size, current\_sum + arr[location], location - 1);

result[location] = INT\_MAX;

}

void findSubset(int arr[], int size, int sum)

{

if (size <= 0)

{

return;

}

int result[size];

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

{

result[i] = INT\_MAX;

}

cout << "Subset Sum of " << sum << " is \n";

this->subsetSum(arr, result, sum, size, 0, size - 1);

}

};

int main()

{

Subset task = Subset();

int n;

cout<<"Enter the size of the array\n";

cin>>n;

int arr[n]={};

cout<<"Enter the array element:"<<endl;

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

{

cin>>arr[i];

}

int size = sizeof(arr) / sizeof(arr[0]);

int sum;

cout<<"Enter the sum element: "<<endl;

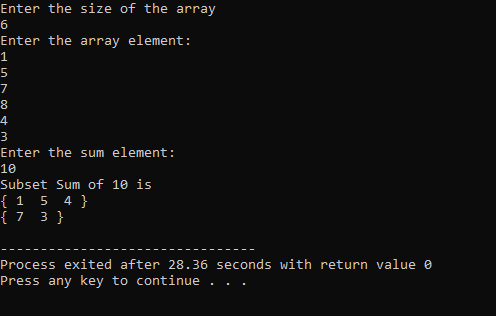
cin>>sum;

task.findSubset(arr, size, sum);

return 0;

}

**OUTPUT:**

****

**9. Write a C++ program to implement merge sort technique using divide and conquer method.**

#include<iostream>

using namespace std;

void Merge(int \*a,int low, int high,int mid)

{

int i,j,k,temp[high-low+1];

i=low;

k=0;

j=mid+1;

while(i<=mid && j<=high)

{

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

{

temp[k]=a[i];

k++;

i++;

}

else

{

temp[k]=a[j];

k++;

j++;

}

}

while(i<=mid)

{

temp[k]=a[i];

k++;

i++;

}

while(j<=high)

{

temp[k]=a[j];

k++;

j++;

}

for(i=low;i<=high;i++)

{

a[i]=temp[i-low];

}

}

void MergeSort(int \*a,int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

MergeSort(a,low,mid);

MergeSort(a,mid+1,high);

Merge(a,low,high,mid);

}

}

int main()

{

int n,i;

cout<<"\n Enter the number of data element to be sorted:";

cin>>n;

int arr[n];

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

{

cout<<"enter element"<<i+1<<":";

cin>>arr[i];

}

MergeSort(arr,0,n-1);

cout<<"\n Sorted Data";

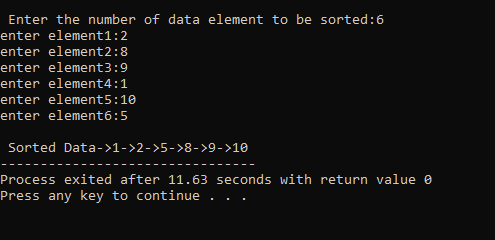
for(i=0;i<n;i++)

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

return(0);

}

**OUTPUT:**

****