**NAME: ALI HASSAN CLASS: BS.IT 3A ENROLL.NO: 03-135211-005**

**TASK 1:**

#include<iostream>

using namespace std;

char tree[15];

int root(char key) {

if (tree[0] != '\0')

cout << "Tree already had root";

else

tree[0] = key;

return 0;

}

int set\_left(char key, int parent) {

if (tree[parent] == '\0')

cout << "\nCan't set child at "<< (parent \* 2) + 1<< " , no parent found";

else

tree[(parent \* 2) + 1] = key;

return 0;

}

int set\_right(char key, int parent) {

if (tree[parent] == '\0')

cout << "\nCan't set child at "<< (parent \* 2) + 2<< " , no parent found";

else

tree[(parent \* 2) + 2] = key;

return 0;

}

int print\_tree() {

cout << "\n";

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

if (tree[i] != '\0')

cout << tree[i];

else

cout << "-";

}

return 0;

}

int main() {

root('A');

set\_right('W', 0);

set\_left('X', 2);

set\_right('Y', 1);

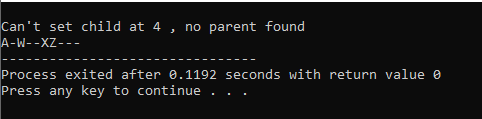
set\_right('Z', 2);

print\_tree();

return 0;

}

**OUTPUT:**



**TASK 2:**

#include <iostream>

#include <string>

#include <queue>

using namespace std;

struct ListNode

{

int data;

ListNode\* next;

};

struct BinaryTreeNode

{

int data;

BinaryTreeNode \*left, \*right;

};

void push(struct ListNode\*\* head\_ref, int new\_data)

{

struct ListNode\* new\_node = new ListNode;

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

BinaryTreeNode\* newBinaryTreeNode(int data)

{

BinaryTreeNode \*temp = new BinaryTreeNode;

temp->data = data;

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

return temp;

}

void convertList2Binary(ListNode \*head, BinaryTreeNode\* &root)

{

queue<BinaryTreeNode \*> q;

if (head == NULL)

{

root = NULL;

return;

}

root = newBinaryTreeNode(head->data);

q.push(root);

head = head->next;

while (head)

{

BinaryTreeNode\* parent = q.front();

q.pop();

BinaryTreeNode \*leftChild = NULL, \*rightChild = NULL;

leftChild = newBinaryTreeNode(head->data);

q.push(leftChild);

head = head->next;

if (head)

{

rightChild = newBinaryTreeNode(head->data);

q.push(rightChild);

head = head->next;

}

parent->left = leftChild;

parent->right = rightChild;

}

}

void inorderTraversal(BinaryTreeNode\* root)

{

if (root)

{

inorderTraversal( root->left );

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

inorderTraversal( root->right );

}

}

int main()

{

struct ListNode\* head = NULL;

push(&head, 36);

push(&head, 30);

push(&head, 25);

push(&head, 15);

push(&head, 12);

push(&head, 10);

BinaryTreeNode \*root;

convertList2Binary(head, root);

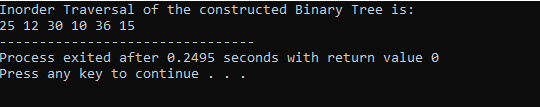
cout << "Inorder Traversal of the constructed Binary Tree is: \n";

inorderTraversal(root);

return 0;

}

**OUTPUT:**



**TASK 3:**

#include<iostream>

using namespace std;

class node{

public:

node \*left\_child=NULL;

node \*right\_child=NULL;

int parent;

};

node \*start;

node \*create\_node(int data)

{ node \*n=new node();

if(start==NULL){

start=n;

start->parent=data;

}

return n;

}

void tree(node \*t,int data){

if(t->left\_child==NULL and t->parent>data)

{ node \*n=new node();

t->left\_child=n;

t->left\_child->parent=data;

}

if(t->right\_child==NULL and t->parent<data)

{ node \*n=new node();

t->right\_child=n;

t->right\_child->parent=data;

}

if(t->parent>data){

tree(t->left\_child,data);

}

if(t->parent<data){

tree(t->right\_child,data);

}

}

void pre\_traverse(node \*pass )

{

if(pass!=NULL)

{

cout<<" "<<pass->parent;

pre\_traverse(pass->left\_child);

pre\_traverse(pass->right\_child);

}

}

void post\_traverse(node \*pass )

{

if(pass!=NULL)

{

post\_traverse(pass->left\_child);

post\_traverse(pass->right\_child);

cout<<" "<<pass->parent;

}

}

void inoder\_traverse(node \*pass )

{

if(pass!=NULL)

{

inoder\_traverse(pass->left\_child);

cout<<" "<<pass->parent;

inoder\_traverse(pass->right\_child);

}

}

node search\_tree(node \*y,int x)

{ if(y->left\_child==NULL and y->right\_child==NULL)

{

cout<<"data not found";

}

if(y->parent==x)

{

cout<<"data found";

}

if(y->parent<x)

{

search\_tree(y->right\_child,x);

}

if(y->parent>x)

{

search\_tree(y->left\_child,x);

}

}

int main()

{

int num;

int data;

int find\_value;

cout<<"Enter number of element you want to store\n";

cin>>num;

cout<<"Enter values\n";

cin>>data;

create\_node(data);

for(int i=1;i<num;i++){

cin>>data;

tree(start,data);

}

cout<<"\npreorder travesal\n";

pre\_traverse(start);

cout<<"\npostorder travesal\n";

post\_traverse(start);

cout<<"\n inorder traversal\n"<<endl;

inoder\_traverse(start);

search\_tree(start,find\_value);

return 0;

}

**OUTPUT:**

