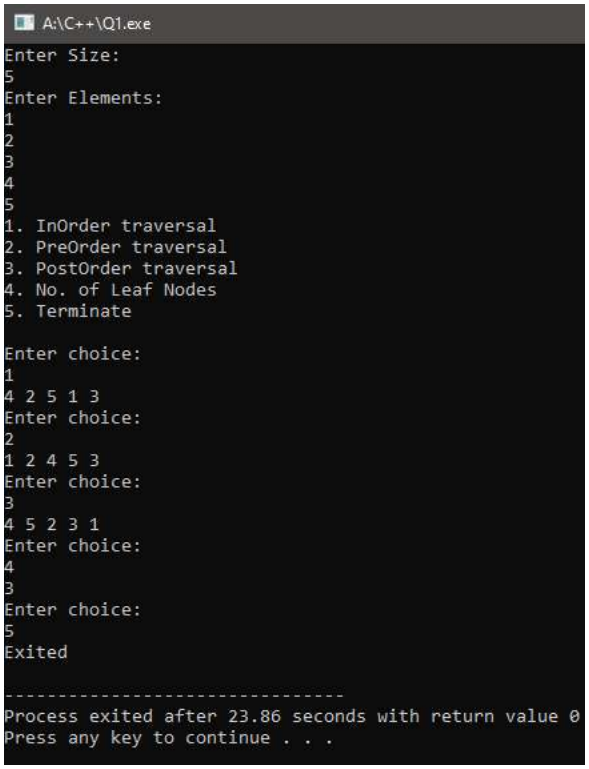
Week6 Batch3

Thursday, December 30, 2021 2:51 PM

1. Write user defined functions to perform the following operations on binary trees:
a) Iteratively create a binary tree
b) In order traversal (Iterative)
c) Post order traversal (Iterative)
d) Preorder traversal(Iterative)
e) Count the number of leaf nodes in a binary tree



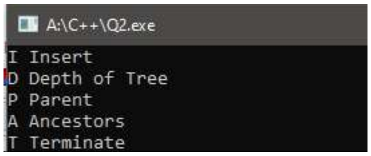
```
Enter Size:
5
Enter Elements:
1
2
3
4
5
1. InOrder traversal
2. PreOrder traversal
3. PostOrder traversal
4. No. of Leaf Nodes
5. Terminate

Enter choice:
1
4 2 5 1 3
Enter choice:
2
1 2 4 5 3
Enter choice:
3
4 5 2 3 1
Enter choice:
4
3
Enter choice:
5
Exited

--------------------------------
Process exited after 23.86 seconds with return value 0
Press any key to continue . . .
```

```
#include<iostream>
using namespace std;
struct Node {
  int Data;
  int Rcount;
  int Lcount;
  struct Node* left;
  struct Node* right;
};

bool isPBT(int count)
{
  count = count + 1;
  while (count % 2 == 0)
    count = count / 2;

  if (count == 1)
    return true;
  else
    return false;
}

int LeafCount(struct Node* node)
{
  if(node == NULL)
    return 0;
  if(node->left == NULL && node->right == NULL)
    return 1;
  else
    return LeafCount(node->left)+
      LeafCount(node->right);
}

struct Node* newNode(int Data)
{
  struct Node* temp = new Node();
  temp->Data = Data;
  temp->right = NULL;
  temp->left = NULL;
  temp->Rcount = 0;
  temp->Lcount = 0;
}

struct Node* insert(struct Node* root,
          int Data)
```

```
{

    if (root == NULL) {
        struct Node* n = newNode(Data);
        return n;
    }

    if (root->Rcount == root->Lcount) {
        root->left = insert(root->left, Data);
        root->Lcount += 1;
    }

    else if (root->Rcount < root->Lcount) {

        if (isPBT(root->Lcount)) {
            root->right = insert(root->right, Data);
            root->Rcount += 1;
        }

        else {
            root->left = insert(root->left, Data);
            root->Lcount += 1;
        }
    }
    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 << " ";
    }
}

int main()
{
    int size;
    struct Node* root = NULL;
    cout << "Enter Size:" << endl;
    cin >> size;
    int arr[size];
    cout << "Enter Elements:" << endl;
    for(int i = 0; i<size; i++)
      cin >> arr[i];

    for(int i = 0; i < size; i++)
      root = insert(root, arr[i]);
    cout<<"1. InOrder traversal" << endl;
    cout<<"2. PreOrder traversal" <<endl;
    cout<<"3. PostOrder traversal"<<endl;
    cout<<"4. No. of Leaf Nodes " <<endl;
    cout<<"5. Terminate"<<endl;
    int c;
    while(1)
    {
      cout << "\n" << "Enter choice:" << endl;
      cin >> c;
      switch(c)
      {
        case 1:
          InOrder(root);
          break;

        case 2:
          PreOrder(root);
          break;

        case 3:
          PostOrder(root);
          break;

        case 4:
          cout << LeafCount(root);
          break;

        case 5:
        cout << "Exited" <<endl;
          exit(0);

        default:
          cout << "Error!! \n Invalid Choice" << endl;
      }
    }
    return 0;
  }
```

2. Write a program to perform the following:
a) Print the parent of the given element
b) Print the depth of a tree
c) Print the ancestors of a given node

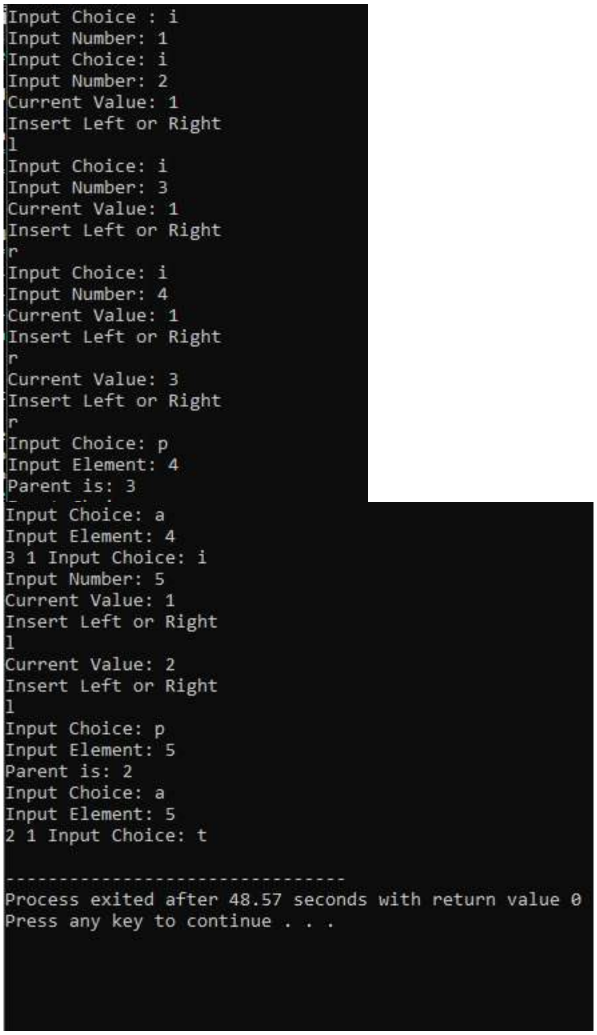
```
#include<iostream>
using namespace std;

struct node
```



A:\C++\Q2.exe

```
I Insert
D Depth of Tree
P Parent
A Ancestors
T Terminate
```

```
{
int val;
node *left;
node *right;
};

node* getNewNode(int x){
node *temp = new node;
temp->val = x;
temp->left = NULL;
temp->right = NULL;
return temp;
}

node* rootNode = NULL;

node* insert(node* root, int x){
if(root == NULL){
node *temp = getNewNode(x);
return temp;
}
cout << "Current Value: " << root->val << endl;
cout << "Insert Left or Right" << endl;
char ch;
cin >> ch;
if(ch == 'L' || ch =='l')
root->left = insert(root->left , x);
else if(ch == 'R' || ch =='r')
root->right = insert(root->right , x);
else
cout << "INVALID!" << endl;
}



void printParent(node* root , int find , int parent){
if(root == NULL){
return ;
}

if(root->val == find){
cout << "Parent is: " << parent << endl;
return;
}
printParent(root->left , find , root->val);
printParent(root->right , find , root->val);
}
bool printAncestors(node* root , int find  ){
if(root == rootNode && find == root->val){
cout << "Root Node has No Ancestors " << endl;
}
if(root->val == find){
return true;
}
if(root->left == NULL && root->right == NULL){
return false;
}
if(root->left != NULL){
bool flag = printAncestors(root->left , find );
```



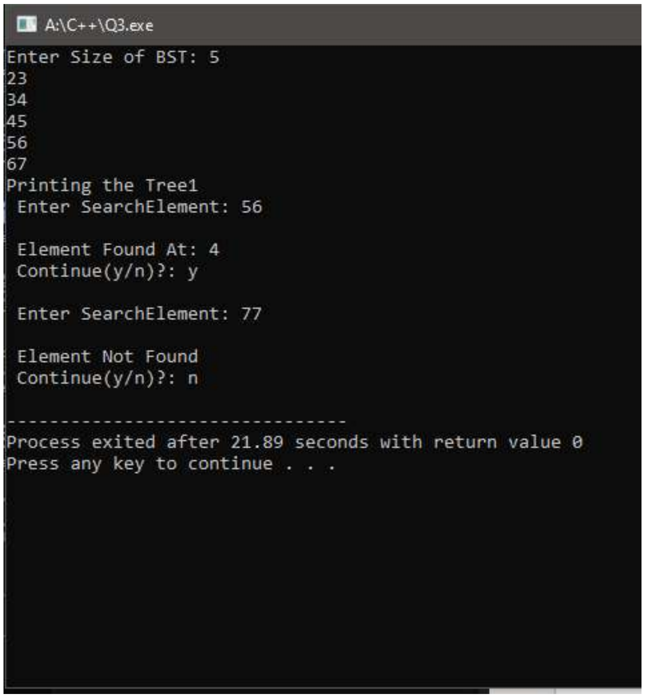
```
Input Choice : i
Input Number: 1
Input Choice: i
Input Number: 2
Current Value: 1
Insert Left or Right
l
Input Choice: i
Input Number: 3
Current Value: 1
Insert Left or Right
r
Input Choice: i
Input Number: 4
Current Value: 1
Insert Left or Right
r
Current Value: 3
Insert Left or Right
r
Input Choice: p
Input Element: 4
Parent is: 3
Input Choice: a
Input Element: 4
3 1 Input Choice: i
Input Number: 5
Current Value: 1
Insert Left or Right
l
Current Value: 2
Insert Left or Right
l
Input Choice: p
Input Element: 5
Parent is: 2
Input Choice: a
Input Element: 5
2 1 Input Choice: t

--------------------------------
Process exited after 48.57 seconds with return value 0
Press any key to continue . . .
```

```
if(flag){
cout << root->val << " ";
return true;
}
}
if(root->right != NULL){
bool flag = printAncestors(root->right , find);
if(flag){
cout << root->val << " ";
return true;
}
}
}
int getDepth(node* root){

if(root->left == NULL && root->right == NULL){
return 1;
}
if(root->left == NULL){
return 1 + getDepth(root->right);
}
if(root->right == NULL){
return 1 + getDepth(root->left);
}
return  1 + max(getDepth(root->left),getDepth(root->right));
}
int depthOfTree(node* root){
return getDepth(root) - 1;
}
int main(){
cout <<"I Insert" << endl;
cout <<"D Depth of Tree" << endl;
cout <<"P Parent" << endl;
cout <<"A Ancestors" << endl;
cout <<"T Terminate" << endl;
cout << "Input Choice : ";
char ch;
cin >> ch;
while(ch != 'T' || ch != 't'){
if(ch == 'I'|| ch =='i'){
cout << "Input Number: ";
int x;
cin >> x;
if(rootNode == NULL){
rootNode = insert(rootNode, x);
}
else
insert(rootNode, x);
}
else if(ch == 'D'|| ch=='d'){
depthOfTree(rootNode);
cout<<endl;
}
else if(ch == 'P'|| ch=='p'){
cout << "Input Element: ";
int x ; cin >> x;
printParent(rootNode, x , -9999);
}
else if(ch == 'A'|| ch=='a'){
```

```
cout << "Input Element: ";
int x ; cin >> x;
printAncestors(rootNode, x);
}
else{
return 0;
}
cout << "Input Choice: ";
cin >> ch;
}
}
```

3. Write a program to search for a given element in a binary search tree.

```
#include<iostream>
using namespace std;
struct node {
  int d;
  node *left;
  node *right;
};
node* CreateNode(int d) {
  node *newnode = new node;
  newnode->d = d;
  newnode->left = NULL;
  newnode->right = NULL;
  return newnode;
}
node* InsertIntoTree(node* root, int d) {
  node *temp = CreateNode(d);
  node *t = new node;
  t = root;
  if(root == NULL)
    root = temp;
  else {
    while(t != NULL) {
      if(t->d < d) {
        if(t->right == NULL) {
          t->right = temp;
          break;
        }
        t = t->right;
      } else if(t->d > d) {
        if(t->left == NULL) {
          t->left = temp;
          break;
        }
        t = t->left;
      }
    }
  }
  return root;
}
void Search(node *root, int d) {
  int depth = 0;
```



```
Enter Size of BST: 5
23
34
45
56
67
Printing the Tree1
 Enter SearchElement: 56

 Element Found At: 4
 Continue(y/n)?: y

 Enter SearchElement: 77

 Element Not Found
 Continue(y/n)?: n

--------------------------------
Process exited after 21.89 seconds with return value 0
Press any key to continue . . .
```

```
  node *temp = new node;
  temp = root;
  while(temp != NULL) {
    depth++;
    if(temp->d == d) {
      cout<<"\n Element Found At: "<<depth;
      return;
    } else if(temp->d > d)
      temp = temp->left;
      else
        temp = temp->right;
  }
  cout<<"\n Element Not Found";
  return;
}
void print2DUtil(node *root, int space)
{
int COUNT =10;
   if (root == NULL)
      return;
   space += COUNT;
   print2DUtil(root->right, space);
   cout<<endl;
   for (int i = COUNT; i < space; i++)
      cout<<" ";
   cout<<root->d<<"\n";
   print2DUtil(root->left, space);
}
void print2D(node *root)
{
   print2DUtil(root, 0);
}
int main() {
  char ch='y';
  int n, i, a[10];
  int counter,counter_a;
  cout<<"Enter Size of BST: ";
  cin>>counter;
  for(int i=0;i<counter;i++){
  cin>>counter_a;
  a[i]=counter_a;
  }
  node *root = new node;
  root = NULL;
  for (i = 0; i < counter; i++)
    root = InsertIntoTree(root, a[i]);
   cout<<"Printing the Tree"<<print2DUtil;
   //print2DUtil(root,3);
  do{
  cout<<"\n Enter SearchElement: ";
  cin>>n;
  Search(root, n);
  cout<<"\n Continue(y/n)?: ";
  cin>>ch;
  }while(ch == 'y' || ch == 'Y');
  return 0;
}
```