**QUESTION NO: 01**

**CODE:**

#include <iostream>

using namespace std;

class BSTree

{ int data;

BSTree \*left, \*right;

public:

BSTree();

BSTree(int);

BSTree\* Insert(BST\*, int);

void Inorder(BSTree\*);

int getHeight(BSTree \*);

bool isEmpty(BSTree\*);

};

BSTree ::BSTree() : data(0) , left(NULL) , right(NULL)

{ }

BSTree ::BSTree(int value)

{ data = value;

left = right = NULL;

}

bool BSTree :: isEmpty(BSTree \*root)

{ if(root == NULL)

return true;

return false;

}

int BSTree :: getHeight(BSTree \*node)

{

if (node == NULL)

return 0;

else

{

int lDepth = getHeight(node->left);

int rDepth = getHeight(node->right);

if (lDepth > rDepth)

return(lDepth + 1);

else return(rDepth + 1);

}

}

BSTree\* BSTree ::Insert(BSTree\* root, int value)

{ if (!root)

{

return new BSTree(value);

}

if (value > root->data)

{

root->right = Insert(root->right, value);

}

else

{

root->left = Insert(root->left, value);

}

return root;

}

void BSTree ::Inorder(BSTree\* root)

{

if (!root)

{

return;

}

Inorder(root->left);

cout << root->data << endl;

Inorder(root->right);

}

int main()

{

BSTree b, \*root = NULL;

root = b.Insert(root, 50);

b.Insert(root, 30);

b.Insert(root, 20);

b.Insert(root, 40);

b.Insert(root, 70);

b.Insert(root, 60);

b.Insert(root, 80);

b.Inorder(root);

return 0;

}

1. A database is a collection of related pieces of information that is organized for easy retrieval. The following set of accounts records, for instance, form an accounts database. Construct BST for accounts database.

**CODE:**

#include <iostream>

#include <fstream>

using namespace std;

struct PersonAccount

{

int RecNum;

int AccountID;

string FirstName;

string LastName;

double Balance;

}

personAccount[15];

struct IndexEntry

{

int acctID;

long recNum;

};

struct node

{

struct IndexEntry \*index;

struct node \*left;

struct node \*right;

}\*root;

class BST

{

public:

BST()

{

root = NULL;

}

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

void search(node\*,int);

};

int main()

{

BST bst;

ifstream infile("accounts.dat");

if(!infile)

{

cout<<endl<<"Create the input file accounts.dat";

exit(0);

}

int n=0;

while(!infile.eof())

{

infile>>personAccount[n].RecNum>>personAccount[n].AccountID>>personAccount[n].FirstName>>personAccount[n].LastName>>personAccount[n].Balance;

struct IndexEntry \*index=new IndexEntry;

index->acctID=personAccount[n].AccountID;

index->recNum=personAccount[n].RecNum;

node\* temp=new node;

temp->index=index;

temp->left=NULL;

temp->right=NULL;

if(root==NULL)

root=temp;

else

bst.insert(root,temp);

n++;

}

int accountID;

cout<<endl<<"Enter account ID: ";

cin>>accountID;

bst.search(root,accountID);

}

void BST::insert(node \*r, node \*temp)

{

if (r == NULL)

{

r = new node;

r->index=temp->index;

r->left = NULL;

r->right = NULL;

}

if (temp->index->acctID < r->index->acctID )

{

if (r->left != NULL)

insert(r->left, temp);

else

{

r->left = new node;

r->left->index=temp->index;

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

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

return;

}

}

else

{

if (r->right != NULL)

{

insert(r->right, temp);

}

else

{

r->right = temp;

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

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

return;

}

}

}

void BST::search(node \*root, int accountID )

{

if(root==NULL) return;

if (root->index->acctID > accountID)

{

if(root->left!=NULL)

search(root->left, accountID);

}

else if( root->index->acctID < accountID)

{

if(root->right!=NULL)

search(root->right, accountID);

}

else if( root->index->acctID == accountID)

{

cout<<endl<<"matched record number is "<<root->index->recNum;

cout<<endl<<"Corresponding account record from the database file: ";

cout<<endl<<"Record # Account ID First Name Last Name Balance";

cout<<endl<<personAccount[root->index->recNum].RecNum<<"\t "<<personAccount[root->index->recNum].AccountID<<"\t "<<personAccount[root->index->recNum].FirstName<<"\t"<<personAccount[root->index->recNum].LastName<<"\t"<<personAccount[root->index->recNum].Balance;

return;

}

}

**OUTPUT:**

Text

Description automatically generated

Text

Description automatically generated

Diagram

Description automatically generated

1. Implement binary search tree. Then create a function which receive a node and should return its level. e.g. Node 14 is at level 1.

A picture containing clock

Description automatically generated

**CODE:**

#include <iostream>

using namespace std;

struct Node

{

int data;

struct Node\* lchild;

struct Node\* rchild;

};

bool find(int data, struct Node\* root)

{

if (root\_ == NULL)

{

return false;

}

else if (root\_->data == data)

{

return true;

}

else if (data > root\_->data)

{

return find(data, root\_->rchild);

}

else

{

return find(data, root\_->lchild);

}

}

bool insert(int data, struct Node\*\* root)

{

if (\*root\_ == NULL)

{

\*root\_ = (struct Node\*)malloc(sizeof(struct Node));

(\*root\_)->data = data;

(\*root\_)->lchild = NULL;

(\*root\_)->rchild = NULL;

return true;

}

else if (data > (\*root\_)->data)

{

return insert(data, &(\*root\_)->rchild);

}

else if (data < (\*root\_)->data)

{

return insert(data, &(\*root\_)->data);

return insert(data, &(\*root\_)->lchild);

}

else

{

return false;

}

}

void postorder(struct Node\* root\_)

{

if (root\_ != NULL)

{

postorder(root\_->lchild);

postorder(root\_->rchild);

std::cout << root\_->data << " ";

}

}