National University of Computer and Emerging Sciences



Name: Muhammad Suleman

Roll #: 22F-3350

Section: BCS-4B

Lab # 09

# Problem 01

#include <iostream>

#include <string>

using namespace std;

struct TreeNode

{

int productID;

string name;

char category;

int price;

float bill;

string personName;

int CNIC;

TreeNode \*left;

TreeNode \*right;

TreeNode(int productID, string name, char category, int price, float bill, string personName, int CNIC)

: productID(productID), name(name), category(category), price(price), bill(bill), personName(personName), CNIC(CNIC), left(nullptr), right(nullptr) {}

};

class BinarySearchTree

{

private:

TreeNode \*root;

void destroyTree(TreeNode \*node)

{

if (node != nullptr)

{

destroyTree(node->left);

destroyTree(node->right);

delete node;

}

}

TreeNode \*insert(TreeNode \*root, int productID, string name, char category, int price, float bill, string personName, int CNIC)

{

if (root == nullptr)

{

TreeNode \*newNode = new TreeNode(productID, name, category, price, bill, personName, CNIC);

return newNode;

}

if (price < root->price)

{

root->left = insert(root->left, productID, name, category, price, bill, personName, CNIC);

}

else if (price > root->price)

{

root->right = insert(root->right, productID, name, category, price, bill, personName, CNIC);

}

return root;

}

TreeNode \*find(TreeNode \*root, int key)

{

if (root == nullptr || root->productID == key)

{

return root;

}

if (key < root->productID)

{

return find(root->left, key);

}

else

{

return find(root->right, key);

}

}

void calculateBill(TreeNode \*root, float &totalBill)

{

if (root != nullptr)

{

calculateBill(root->left, totalBill);

totalBill += root->bill;

calculateBill(root->right, totalBill);

}

}

void print(TreeNode \*root)

{

if (root != nullptr)

{

print(root->left);

cout << "Product ID: " << root->productID << ", Name: " << root->name << ", Category: " << root->category << ", Price: " << root->price << endl;

print(root->right);

}

}

public:

BinarySearchTree()

: root(nullptr) {}

~BinarySearchTree()

{

destroyTree(root);

}

void insert(int productID, string name, char category, int price, float bill, string personName, int CNIC)

{

root = insert(root, productID, name, category, price, bill, personName, CNIC);

}

TreeNode \*find(int key)

{

return find(root, key);

}

float calculateBill()

{

float totalBill = 0.0;

calculateBill(root, totalBill);

return totalBill;

}

void printItemsPurchased()

{

if (root == nullptr)

{

cout << "No items purchased yet" << endl;

}

else

{

cout << "Purchased Items:" << endl;

print(root);

}

}

};

int main()

{

BinarySearchTree eTree;

BinarySearchTree vTree;

eTree.insert(1234, "Laptop", 'E', 1500, 1500, "Suleman", 123456);

eTree.insert(5678, "Smartphone", 'E', 1000, 1000, "Adnam", 987654);

eTree.insert(9012, "Headphones", 'E', 500, 500, "Waleed", 246810);

vTree.insert(3456, "Car", 'V', 50000, 50000, "Abdullah", 135792);

vTree.insert(7890, "Bike", 'V', 2000, 2000, "Usama", 468135);

float totalBill = eTree.calculateBill() + vTree.calculateBill();

cout << "Total Bill: " << totalBill << endl;

eTree.printItemsPurchased();

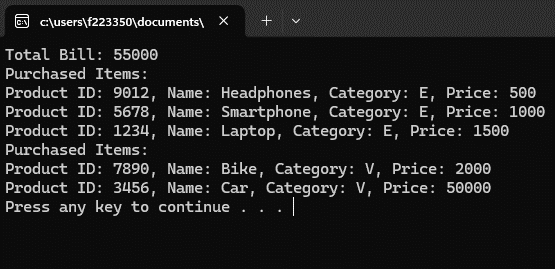
vTree.printItemsPurchased();

system("pause");

return 0;

}

## Output



# Problem 02

#include <iostream>

using namespace std;

template<typename T>

struct Node

{

T data;

Node\* left;

Node\* right;

Node(T value)

: data(value), left(nullptr), right(nullptr) {}

};

template<typename T>

class BinarySearchTree

{

private:

Node<T>\* root;

Node<T>\* copyTree(const Node<T>\* srcNode)

{

if (srcNode == nullptr)

{

return nullptr;

}

Node<T>\* newNode = new Node<T>(srcNode->data);

newNode->left = copyTree(srcNode->left);

newNode->right = copyTree(srcNode->right);

return newNode;

}

void destroyTree(Node<T>\* node)

{

if (node != nullptr)

{

destroyTree(node->left);

destroyTree(node->right);

delete node;

}

}

Node<T>\* insert(Node<T>\* root, T data)

{

if (root == nullptr)

{

Node<T>\* newNode = new Node<T>(data);

return newNode;

}

if (data < root->data)

{

root->left = insert(root->left, data);

}

else if (data > root->data)

{

root->right = insert(root->right, data);

}

return root;

}

Node<T>\* find(Node<T>\* root, T key)

{

if (root == nullptr || root->data == key)

{

return root;

}

if (key < root->data)

{

return find(root->left, key);

}

else

{

return find(root->right, key);

}

}

void InorderPrint(Node<T>\* root)

{

if (root == nullptr)

{

return;

}

InorderPrint(root->left);

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

InorderPrint(root->right);

}

void preorderorderPrint(Node<T>\* root)

{

if (root == nullptr)

{

return;

}

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

preorderorderPrint(root->left);

preorderorderPrint(root->right);

}

void postorderPrint(Node<T>\* root)

{

if (root == nullptr)

{

return;

}

postorderPrint(root->left);

postorderPrint(root->right);

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

}

int length(Node<T>\* root)

{

if (root == nullptr)

{

return 0;

}

return 1 + length(root->left) + length(root->right);

}

int leafCount(Node<T>\* root)

{

if (root == nullptr)

{

return 0;

}

if (root->left == nullptr && root->right == nullptr)

{

return 1;

}

return leafCount(root->left) + leafCount(root->right);

}

public:

BinarySearchTree()

: root(nullptr) {}

BinarySearchTree(const BinarySearchTree<T>& other)

{

root = copyTree(other.root);

}

~BinarySearchTree()

{

destroyTree(root);

}

void insert(T data)

{

root = insert(root, data);

}

Node<T>\* find(T key)

{

return find(root, key);

}

void InorderPrint()

{

InorderPrint(root);

}

void preorderorderPrint()

{

preorderorderPrint(root);

}

void postorderPrint()

{

postorderPrint(root);

}

int length()

{

return length(root);

}

int leafCount()

{

return leafCount(root);

}

};

int main()

{

BinarySearchTree<int> tree1;

tree1.insert(500);

tree1.insert(1000);

tree1.insert(1);

tree1.insert(600);

tree1.insert(700);

tree1.insert(10);

tree1.insert(30);

tree1.insert(9000);

tree1.insert(50000);

tree1.insert(20);

cout << "Printing data using recursive inorder traversal: ";

tree1.InorderPrint();

cout << endl;

cout << "Using copy constructor to copy tree1 to tree2" << endl;

BinarySearchTree<int> tree2(tree1);

cout << "Preorder Traversal: ";

tree2.preorderorderPrint();

cout << endl;

cout << "Inorder Traversal: ";

tree2.InorderPrint();

cout << endl;

cout << "Postorder Traversal: ";

tree2.postorderPrint();

cout << endl;

cout << "SEARCH:" << endl;

int arr[] = { 1,30,50 };

for (int i = 0; i < 3; i++)

{

if (tree2.find(arr[i]))

{

cout << arr[i] << " exists in tree" << endl;

}

else

{

cout << arr[i] << " doesn't exists in tree" << endl;

}

}

cout << "Tree Length: " << tree1.length() << endl;

cout << "Tree Leaf Nodes: " << tree1.leafCount() << endl;

system("pause");

return 0;

}

## Output

