/\* -----------------------------------------------

Name: J-Zach Loke

Course: CMPS-385

Semester: Spring 2020

Project: No. 10 Part 1

Purpose: Insert data from an array into a BST

----------------------------------------------- \*/

#include <iostream>

#include <string>

#include <algorithm>

template <class T>

class BST

{

private:

struct node

{

T info;

node \*left, \*right;

};

node \*root = NULL;

void place(node \*insert, node \*curr)

{

if (insert->info < curr->info)

{

if (curr->left == NULL) { curr->left = insert; }

else { place(insert, curr->left); }

}

else if (insert->info > curr->info)

{

if (curr->right == NULL) { curr->right = insert;}

else { place(insert, curr->right); }

}

}

public:

node\* getRoot() { return root; }

void insert(T x)

{

node \*insert = new node;

insert->info = x;

insert->left = NULL;

insert->right = NULL;

if (root == NULL) { root = insert; }

else { place(insert, root); }

}

void displayPre(node \*curr)

{

if (curr != NULL)

{

std::cout << curr->info << '\t';

displayPre(curr->left);

displayPre(curr->right);

}

}

void displayIn(node \*curr)

{

if (curr != NULL)

{

displayIn(curr->left);

std::cout << curr->info << '\t';

displayIn(curr->right);

}

}

void displayPost(node \*curr)

{

if (curr != NULL)

{

displayPost(curr->left);

displayPost(curr->right);

std::cout << curr->info << '\t';

}

}

void displayLeaves(node \*curr)

{

if (curr != NULL)

{

if (curr->left == NULL && curr->right == NULL) { std::cout << curr->info << '\t'; }

displayLeaves(curr->left);

displayLeaves(curr->right);

}

}

void displayOnlyChild(node \*curr)

{

if (curr != NULL)

{

if ((curr->left == NULL || curr->right == NULL) && (curr->left != NULL || curr->right != NULL)) { std::cout << curr->info << '\t'; }

displayOnlyChild(curr->left);

displayOnlyChild(curr->right);

}

}

int getHeight(node \*curr)

{

if (curr == NULL) { return 0; }

else { return 1 + std::max(getHeight(curr->left), getHeight(curr->right)); }

}

bool search(T x, node \*curr)

{

if (curr != NULL)

{

if (curr->info == x) { return true; }

if (search(x, curr->left)) { return true; }

if (search(x, curr->right)) { return true; }

return false;

}

return false;

}

int getNodeCount(node \*curr)

{

if (curr == NULL) { return 0; }

else { return 1 + getNodeCount(curr->left) + getNodeCount(curr->right); }

}

};

int main()

{

/\* name: main

input: N/A

output: N/A

purpose: main function to drive the program \*/

// given

std::string Days[7] = {"MON", "TUE", "WED", "THU", "FRI", "SAT", "SUN"};

BST<std::string> tree;

for (int i = 0; i < 7; ++i) { tree.insert(Days[i]); }

// part a: display all nodes using inorder traversal

std::cout << "Inorder:\t";

tree.displayIn(tree.getRoot());

std::cout << std::endl;

// part b: display all nodes using postorder traversal

std::cout << "Postorder:\t";

tree.displayPost(tree.getRoot());

std::cout << std::endl;

// part c: display all nodes using preorder traversal

std::cout << "Preorder:\t";

tree.displayPre(tree.getRoot());

std::cout << std::endl << std::endl;

// part d: display only the leaves of the tree

std::cout << "Leaves:\t";

tree.displayLeaves(tree.getRoot());

std::cout << std::endl;

// part e: display nodes with only one child

std::cout << "Nodes with only one child : ";

tree.displayOnlyChild(tree.getRoot());

std::cout << std::endl;

// part f: return the height of the tree

std::cout << "Height: " << tree.getHeight(tree.getRoot()) << std::endl;

// part g: search for "MON" and "THR" to determine whether they are in the tree or not

if (tree.search("MON", tree.getRoot())) { std::cout << "MON is in the tree." << std::endl; }

else { std::cout << "MON is not in the tree." << std::endl; }

if (tree.search("THR", tree.getRoot())) { std::cout << "THR is in the tree." << std::endl; }

else { std::cout << "THR is not in the tree." << std::endl; }

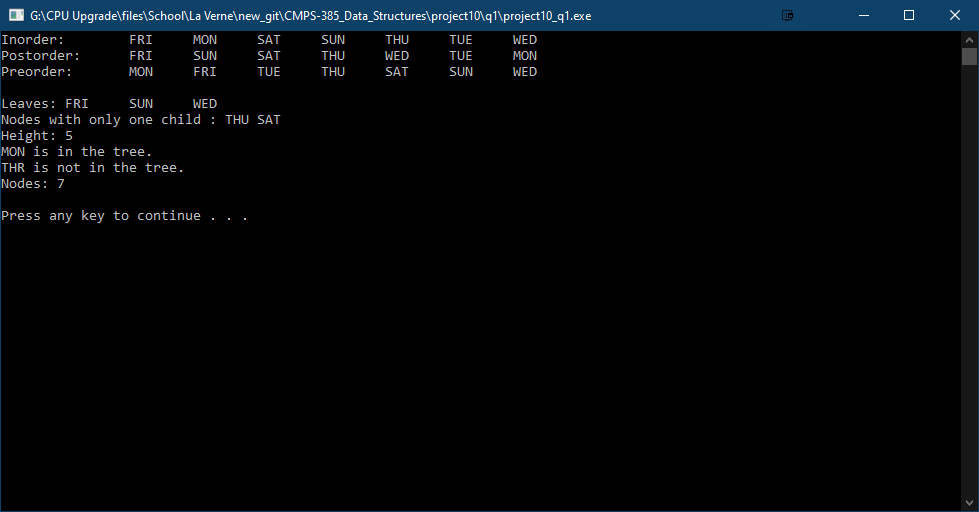
// part h: create a recursive function to return the number of nodes in the tree

std::cout << "Nodes: " << tree.getNodeCount(tree.getRoot()) << std::endl << std::endl;

system("pause");

return 0;

}



/\* -----------------------------------------------

Name: J-Zach Loke

Course: CMPS-385

Semester: Spring 2020

Project: No. 10 Part 2

Purpose: a. generate 12 random numbers < 100 and store them all in array a[12]

b. display array a

c. copy data in array a[] into a BST

d. display BST using inorder traversal

e. add a member to class BST to return the max data in the tree

f. add a member to class BST to return the min data in the tree

----------------------------------------------- \*/

#include <iostream>

#include <algorithm>

#include <time.h>

template <class T>

class BST

{

private:

struct node

{

T info;

node \*left, \*right;

};

node \*root = NULL;

void place(node \*insert, node \*curr)

{

if (insert->info < curr->info)

{

if (curr->left == NULL) { curr->left = insert; }

else { place(insert, curr->left); }

}

else if (insert->info > curr->info)

{

if (curr->right == NULL) { curr->right = insert;}

else { place(insert, curr->right); }

}

}

public:

node\* getRoot() { return root; }

void insert(T x)

{

node \*insert = new node;

insert->info = x;

insert->left = NULL;

insert->right = NULL;

if (root == NULL) { root = insert; }

else { place(insert, root); }

}

void displayPre(node \*curr)

{

if (curr != NULL)

{

std::cout << curr->info << '\t';

displayPre(curr->left);

displayPre(curr->right);

}

}

void displayIn(node \*curr)

{

if (curr != NULL)

{

displayIn(curr->left);

std::cout << curr->info << '\t';

displayIn(curr->right);

}

}

void displayPost(node \*curr)

{

if (curr != NULL)

{

displayPost(curr->left);

displayPost(curr->right);

std::cout << curr->info << '\t';

}

}

void displayLeaves(node \*curr)

{

if (curr != NULL)

{

if (curr->left == NULL && curr->right == NULL) { std::cout << curr->info << '\t'; }

displayLeaves(curr->left);

displayLeaves(curr->right);

}

}

void displayOnlyChild(node \*curr)

{

if (curr != NULL)

{

if ((curr->left == NULL || curr->right == NULL) && (curr->left != NULL || curr->right != NULL)) { std::cout << curr->info << '\t'; }

displayOnlyChild(curr->left);

displayOnlyChild(curr->right);

}

}

int getHeight(node \*curr)

{

if (curr == NULL) { return 0; }

else { return 1 + std::max(getHeight(curr->left), getHeight(curr->right)); }

}

bool search(T x, node \*curr)

{

if (curr != NULL)

{

if (curr->info == x) { return true; }

if (search(x, curr->left)) { return true; }

if (search(x, curr->right)) { return true; }

return false;

}

return false;

}

int getNodeCount(node \*curr)

{

if (curr == NULL) { return 0; }

else { return 1 + getNodeCount(curr->left) + getNodeCount(curr->right); }

}

int getMax()

{

node \*curr = root;

while (curr->right != NULL) { curr = curr->right; }

return curr->info;

}

int getMin()

{

node \*curr = root;

while (curr->left != NULL) { curr = curr->left; }

return curr->info;

}

};

int main()

{

/\* name: main

input: N/A

output: N/A

purpose: main function to drive the program \*/

// a. generate 12 random numbers < 100 and store them all in array a[12]

int a[12];

srand(time(NULL));

for (int i = 0; i < 12; ++i) { a[i] = rand() % 100; }

// b. display array a

std::cout << "Array a:\t";

for (int i : a) { std::cout << i << '\t'; }

std::cout << std::endl;

// c. copy data in array a[] into a BST

BST<int> tree;

for (int i : a) { tree.insert(i); }

// d. display BST using inorder traversal

std::cout << "Inorder:\t";

tree.displayIn(tree.getRoot());

std::cout << std::endl;

// e. add a member to class BST to return the max data in the tree

std::cout << "Max: " << tree.getMax() << std::endl;

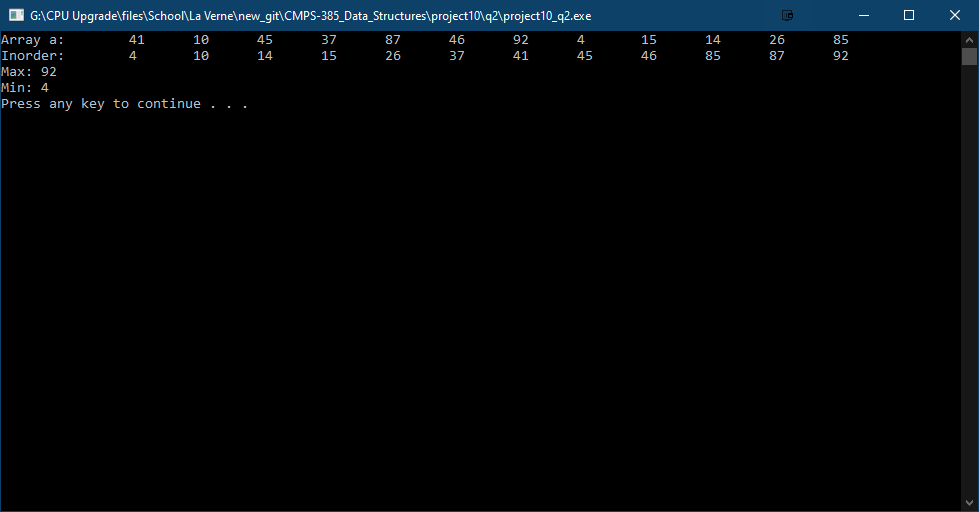
// f. add a member to class BST to return the min data in the tree

std::cout << "Min: " << tree.getMin() << std::endl;

system("pause");

return 0;

}



1. Given the following BT. Trace each algorithm and show their output

22

1. 13 Inorder traversal…**5 33 11 6 22 13 9**..

5 11 9 Preorder traversal…**22 33 5 11 6 13 9**…

\ 6 Postorder traversal…**5 6 11 33 9 13 22**…

1. Trace the following functions using the tree in problem 3

|  |  |
| --- | --- |
| int f( node \*r)  {  If( r == NULL) return 0;  else return r->info + f(r->left) + f(r->right);  }  Calling statement:  cout<<f( root); ……………**99**……………… | void g(node \*r)  { If( r != NULL)  { g(r->right);  cout<<r->info<<” “;  g( r->left);  }  Calling statement:  g(root); **9 13 22 6 11 33 5** |