**Bahria University, Lahore Campus**

Department of Computer Science

Lab Journal 09

**(Spring 2023)**

|  |  |  |
| --- | --- | --- |
| Course: | **Data Structures and Algorithm - Lab** | Date: 18-5-2023\_ |
| Course Code: | CSL-221 | Max Marks: 10 |
| Faculty’s Name: | Fatima Zulfiqar |  |

Name: \_AFFAN AHMAD \_ Enroll No: \_03-134221-003\_\_ Class: \_BS(sc)\_\_\_\_\_\_\_\_\_\_\_\_\_

Objective(s):

Upon completion of this lab session, learners will be able to:

* Implement binary tree data structure
* Implement tree traversal i.e. in-order, pre-order and post-order
* Search for a particular element in a binary tree.

## Lab Tasks:

**Task 1**

Write a program to implement a given binary tree data structure



#include <iostream>

using namespace std;

struct node {

int data;

node \*left;

node \*right;

};

node \*create(int val)

{

node \* newnode = new node;

newnode->data = val;

newnode->right = NULL;

newnode->left = NULL;

return newnode;

}

void preorder(node\* root)

{

if (root != NULL)

{

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

preorder(root->left);

preorder(root->right);

}

}

void inorder(node\* root)

{

if (root != NULL)

{

inorder(root->left);

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

inorder(root->right);

}

}

void postorder(node\* root)

{

if (root != NULL)

{

postorder(root->left);

postorder(root->right);

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

}

}

void searching( node \*root, int val )

{

if (root != NULL)

{

if (root->data == val)

{

cout << val << " is found " << endl;

}

searching(root->left, val);

searching(root->right, val);

}

else

{

cout << "elsement not found " << endl;

}

}

int main()

{

int v,val;

node \*newnode = create (1);

//root left

newnode->left = create(2);

newnode->left->right = create(5);

newnode->left->right->left = create(10);

newnode->left->right->right = create(11);

newnode->left->left = create(4);

newnode->left->left->right = create(9);

newnode->left->left->left = create(8);

// root right

newnode->right = create(3);

newnode->right->left = create(6);

newnode->right->right = create(7);

newnode->right->left->left = create(13);

newnode->right->right->right=create(14);

cout << "pree 1 for pre order " << endl;

cout << "pree 2 for in order " << endl;

cout << "pree 3 for post order " << endl;

cout << "pree 5 for searching using preorder " << endl;

cout << "pree 4 for exit " << endl;

do

{

cout << "enter your choise :"; cin >> v;

if (v == 1)

{

cout << "pre order traversal is :" << endl;

preorder(newnode);

cout << endl;

}

if (v == 2)

{

cout << "in order traversal is :" << endl;

inorder(newnode);

cout << endl;

}

if (v == 3)

{

cout << "post order traversal is :" << endl;

postorder(newnode);

cout << endl;

}

if (v == 5)

{

cout << "enter your value "; cin >> val;

searching(newnode, val);

}

if (v == 4)

{

cout << "EXIT " << endl;

break;

}

} while (v != 4);

system("pause");

return 0;

}

**Task 2**

Using tree obtained in **Task 1,** implement a function to perform following tree traversal.

* In-order traversal
* Pre-order traversal
* Post-order traversal
* #include <iostream>
* using namespace std;
* struct node {
* int data;
* node \*left;
* node \*right;
* };
* node \*create(int val)
* {
* node \* newnode = new node;
* newnode->data = val;
* newnode->right = NULL;
* newnode->left = NULL;
* return newnode;
* }
* void preorder(node\* root)
* {
* if (root != NULL)
* {
* cout << root->data<< " ";
* preorder(root->left);
* preorder(root->right);
* }
* }
* void inorder(node\* root)
* {
* if (root != NULL)
* {
* inorder(root->left);
* cout << root->data<<" ";
* inorder(root->right);
* }
* }
* void postorder(node\* root)
* {
* if (root != NULL)
* {
* postorder(root->left);
* postorder(root->right);
* cout << root->data << " ";
* }
* }
* void searching( node \*root, int val )
* {
* if (root != NULL)
* {
* if (root->data == val)
* {
* cout << val << " is found " << endl;
* }
* searching(root->left, val);
* searching(root->right, val);
* }
* else
* {
* cout << "elsement not found " << endl;
* }
* }
* int main()
* {
* int v,val;
* node \*newnode = create (1);
* //root left
* newnode->left = create(2);
* newnode->left->right = create(5);
* newnode->left->right->left = create(10);
* newnode->left->right->right = create(11);
* newnode->left->left = create(4);
* newnode->left->left->right = create(9);
* newnode->left->left->left = create(8);
* // root right
* newnode->right = create(3);
* newnode->right->left = create(6);
* newnode->right->right = create(7);
* newnode->right->left->left = create(13);
* newnode->right->right->right=create(14);
* cout << "pree 1 for pre order " << endl;
* cout << "pree 2 for in order " << endl;
* cout << "pree 3 for post order " << endl;
* cout << "pree 5 for searching using preorder " << endl;
* cout << "pree 4 for exit " << endl;
* do
* {
* cout << "enter your choise :"; cin >> v;
* if (v == 1)
* {
* cout << "pre order traversal is :" << endl;
* preorder(newnode);
* cout << endl;
* }
* if (v == 2)
* {
* cout << "in order traversal is :" << endl;
* inorder(newnode);
* cout << endl;
* }
* if (v == 3)
* {
* cout << "post order traversal is :" << endl;
* postorder(newnode);
* cout << endl;
* }
* if (v == 5)
* {
* cout << "enter your value "; cin >> val;
* searching(newnode, val);
* }
* if (v == 4)
* {
* cout << "EXIT " << endl;
* break;
* }
* } while (v != 4);
* system("pause");
* return 0;
* }

**Task 3**

Write a function to search for a specific element from the binary tree generated in **Task 1.** If an element is present in a tree simply display a message **“*an element is present in a binary tree*”** otherwise display **“*an element is not present in a binary tree*”.**

#include <iostream>

using namespace std;

struct node {

int data;

node \*left;

node \*right;

};

node \*create(int val)

{

node \* newnode = new node;

newnode->data = val;

newnode->right = NULL;

newnode->left = NULL;

return newnode;

}

void preorder(node\* root)

{

if (root != NULL)

{

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

preorder(root->left);

preorder(root->right);

}

}

void inorder(node\* root)

{

if (root != NULL)

{

inorder(root->left);

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

inorder(root->right);

}

}

void postorder(node\* root)

{

if (root != NULL)

{

postorder(root->left);

postorder(root->right);

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

}

}

void searching( node \*root, int val )

{

if (root != NULL)

{

if (root->data == val)

{

cout << val << " is found " << endl;

}

searching(root->left, val);

searching(root->right, val);

}

else

{

cout << "elsement not found " << endl;

}

}

int main()

{

int v,val;

node \*newnode = create (1);

//root left

newnode->left = create(2);

newnode->left->right = create(5);

newnode->left->right->left = create(10);

newnode->left->right->right = create(11);

newnode->left->left = create(4);

newnode->left->left->right = create(9);

newnode->left->left->left = create(8);

// root right

newnode->right = create(3);

newnode->right->left = create(6);

newnode->right->right = create(7);

newnode->right->left->left = create(13);

newnode->right->right->right=create(14);

cout << "pree 1 for pre order " << endl;

cout << "pree 2 for in order " << endl;

cout << "pree 3 for post order " << endl;

cout << "pree 5 for searching using preorder " << endl;

cout << "pree 4 for exit " << endl;

do

{

cout << "enter your choise :"; cin >> v;

if (v == 1)

{

cout << "pre order traversal is :" << endl;

preorder(newnode);

cout << endl;

}

if (v == 2)

{

cout << "in order traversal is :" << endl;

inorder(newnode);

cout << endl;

}

if (v == 3)

{

cout << "post order traversal is :" << endl;

postorder(newnode);

cout << endl;

}

if (v == 5)

{

cout << "enter your value "; cin >> val;

searching(newnode, val);

}

if (v == 4)

{

cout << "EXIT " << endl;

break;

}

} while (v != 4);

system("pause");

return 0;

}

**Lab Grading Sheet :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Max Marks** | **Obtained Marks** | **Comments(*if any*)** |
| 1. | 2 |  |  |
| 2. | 6 |  |  |
| 3 | 2 |  |  |
| **Total** | **10** |  | **Signature** |

**Note : Attempt all tasks and get them checked by your Lab Instructor. Also for each task, attach a screenshot of the output.**