**Data Structures and Algorithms**

**Lab Journal - Lab 7**

Name: Saad Ahmad

Enrollment #: 01-134222-130

Class/Section: BS-CS-3A

**Objective**

This lab session is aimed at introducing students to the ‘Binary Tree’ data structure.

**Task 1 :**

Give answers to the following.

|  |  |
| --- | --- |
| 1. | For a complete binary tree of depth ‘*d*’, find the following.   1. Number of nodes at level *k* of the tree: 2k 2. Number of leaves in the tree: 2d 3. Total number of nodes in the tree: 2d+1-1 |
| 2. | A complete binary tree has a total of 15 nodes. What is the depth of the binary tree? What would be the depth of a complete binary tree with ‘*K*’ nodes?  Depth : 3  Depth for k : log2(K+1)-1 |
| 3. | For the given binary tree, state the following.   1. Number of leaf nodes: 4 2. Number of descendants of node containing 11:4 3. Depth of the tree: 3 4. Parent node of the node containing 30: 11 5. Type of Binary Tree: Simple Binary tree 6. Level/Depth of node containing 10:2 7. Children of the root: 2 11 8. Name the ancestors of node containing 7: 10 11 14 |
| 4. | Traverse the binary tree given above in pre, post and inorder.   1. Preorder Traversal: 14,2,1,3,11,10,7,30,40 2. Post Traversal: 1,3,2,7,10,40,30,11,14 3. In-order Traversal: 1,2,3,14,7,10,11,40,30 |
| 5. | Draw the expression tree of the given algebraic expression and traverse the tree in pre, post and inorder.  **(a+b\*c)+((d\*e+f)\*g)**  Pre : ++a\*bc\*+\*defg  Post: abc\*+de\*f+g\*+  Inorder: (a+(b\*c))+(((d\*e)+f)\*g) |

**Task 2 :**

Implement the following exercises.

**Exercise 1**

|  |
| --- |
| Complete the given class to implement a binary tree.  #include <iostream>  using namespace std;  class Node  {  public:  Node\* left;  Node\* right;  int data;  };  class bt  {  public:  Node\* root;  bt() {  root = NULL;  }  bool isempty() {  if (root == NULL) {  return true;  }  else {  return false;  }  }  Node\* insert(int item) {  Node\* temp = new Node();  temp->data = item;  temp->left = NULL;  temp->right = NULL;  return temp;  }  void traversal(Node\* temp) {  if (temp != NULL)  {  cout << temp->data << " ";  traversal(temp->left);  traversal(temp->right);  }  else {  return;  }  }  void Preorder(Node\* ptr)  {  if (ptr != NULL)  {  cout << ptr->data << " ";  Preorder(ptr->left);  Preorder(ptr->right);  }  }  void Postorder(Node\* ptr)  {  if (ptr != NULL)  {  Postorder(ptr->left);  Postorder(ptr->right);  cout << ptr->data << " ";  }  }  void Inorder(Node\* ptr)  {  if (ptr != NULL)  {  Inorder(ptr->left);  cout << ptr->data << " ";  Inorder(ptr->right);  }  }  };  int main() {  bt b;  b.root = b.insert(0);  b.root->left = b.insert(1);  b.root->left->right = b.insert(2);  b.root->left->left = b.insert(3);  b.root->right = b.insert(4);  b.root->right->left = b.insert(5);  b.root->right->right = b.insert(6);  b.traversal(b.root);  cout << endl;  b.Postorder(b.root);  cout << endl;  b.Preorder(b.root);  cout << endl;  b.Inorder(b.root);  } |

**Exercise 2**

|  |
| --- |
| #include <iostream>  using namespace std;  class Node  {  public:  Node\* left;  Node\* right;  int data;  };  class bt  {  public:  Node\* root;  bt() {  root = NULL;  }  bool isempty() {  if (root == NULL) {  return true;  }  else {  return false;  }  }  Node\* insert(int item) {  Node\* temp = new Node();  temp->data = item;  temp->left = NULL;  temp->right = NULL;  return temp;  }  void traversal(Node\* temp) {  if (temp != NULL)  {  cout << temp->data << " ";  traversal(temp->left);  traversal(temp->right);  }  else {  return;  }  }  void Preorder(Node\* ptr)  {  if (ptr != NULL)  {  cout << ptr->data << " ";  Preorder(ptr->left);  Preorder(ptr->right);  }  }  void Postorder(Node\* ptr)  {  if (ptr != NULL)  {  Postorder(ptr->left);  Postorder(ptr->right);  cout << ptr->data << " ";  }  }  void Inorder(Node\* ptr)  {  if (ptr != NULL)  {  Inorder(ptr->left);  cout << ptr->data << " ";  Inorder(ptr->right);  }  }  };  int main() {  bt b;  b.root = b.insert(0);  b.root->left = b.insert(1);  b.root->left->right = b.insert(2);  b.root->left->left = b.insert(3);  b.root->right = b.insert(4);  b.root->right->left = b.insert(5);  b.root->right->right = b.insert(6);  b.traversal(b.root);  cout << endl;  b.Postorder(b.root);  cout << endl;  b.Preorder(b.root);  cout << endl;  b.Inorder(b.root);  } |

**Implement the given exercises and get them checked by your instructor. If you are unable to complete the tasks in the lab session, deposit this journal alongwith your programs (printed or handwritten) before the start of the next lab session.**

|  |  |  |
| --- | --- | --- |
| **S No.** | **Exercise** | **Checked By:** |
| 1. | Exercise 1 |  |
| 2. | Exercise 2 |  |

+++++++++++++++++++++++++