|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Data Structures Laboratory (using C/C++)** | **Semester:** | **III** |
| **Date of Performance:** | **10 / 10 / 2023** | **Batch No:** | **A-3** |
| **Faculty Name:** | **Om Goswami** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No: 5**

**Title: Binary Search Tree Traversal**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Write a program for binary search tree (BST) traversal. |

|  |
| --- |
| **COs to be achieved:** |
| **CO1:** Understand and implement the different data structures used in problem solving  **CO2:** Apply linear and non-linear data structure in application development |

|  |
| --- |
| **Books/Journals/Websites referred:** |
|  |

|  |
| --- |
| **Tools required:** |
| **DEV C/C++ compiler/ Code blocks C compiler** |

|  |
| --- |
| **Theory:** |
| Binary search tree (BST) is a special kind of binary tree where each node contains -   1. Only larger values in its right subtree. 2. Only smaller values in its left subtree.   Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.  Example Tree   1. Inorder (Left, Root, Right): 4 2 5 1 3 2. Preorder (Root, Left, Right): 1 2 4 5 3 3. Postorder (Left, Right, Root): 4 5 2 3 1 |

|  |
| --- |
| **Stepwise-Procedure:** |
| 1. Inorder(tree)    1. Traverse the left subtree, i.e., call Inorder(left-subtree)    2. Visit the root.    3. Traverse the right subtree, i.e., call Inorder(right-subtree) 2. Preorder(tree)    1. Visit the root.    2. Traverse the left subtree, i.e., call Preorder(left-subtree)    3. Traverse the right subtree, i.e., call Preorder(right-subtree) 3. Postorder(tree)    1. Traverse the left subtree, i.e., call Postorder(left-subtree)    2. Traverse the right subtree, i.e., call Postorder(right-subtree)    3. Visit the root. |

|  |
| --- |
| **Implementation details:** |
| 1. **Enlist all the Steps followed and various options explored.**   The code provided is a C program that implements a binary search tree (BST) and provides options for various tree operations. The steps followed in the program are:   * 1. Creation of a new node   2. Insertion of a node   3. Inorder, Preorder, and Postorder traversals of the tree   4. A menu-driven program in the main function to interact with these operations  1. **Explain your program logic and methods used.**   The program uses a struct to define the node of a BST, which contains an integer data and pointers to the left and right children.   * createNode(int data): This function creates a new node with the given data, sets its left and right children to NULL, and returns the node. * \*insert(struct Node root, int data)\*\*: This function inserts a new node with the given data into the BST. If the tree is empty, it creates a new node. If the data is less than the root, it inserts the node in the left subtree; if the data is more than the root, it inserts the node in the right subtree. * \*inorderTraversal(struct Node root)\*\*: This function prints the elements of the BST in Inorder traversal. It first visits the left subtree, then the root, and finally the right subtree. * \*preorderTraversal(struct Node root)\*\*: This function prints the elements of the BST in Preorder traversal. It first visits the root, then the left subtree, and finally the right subtree. * \*postorderTraversal(struct Node root)\*\*: This function prints the elements of the BST in Postorder traversal. It first visits the left subtree, then the right subtree, and finally the root.  1. **Explain the Importance of the approach followed by you.**   The approach used in this program is fundamental to understanding and implementing binary search trees. BST is a crucial data structure used in many areas of computer science, including database and file systems. The program demonstrates the basic operations performed on a BST, such as insertion and different types of traversals.  The traversal methods (Inorder, Preorder, Postorder) are significant as they are used in many applications. Inorder traversal on a BST gives nodes in non-decreasing order, Preorder traversal can be used to create a copy of the tree, and Postorder traversal is used to delete the tree. |

|  |
| --- |
| **C/C++ Code implemented:** |
| #include <stdio.h>  #include <stdlib.h>  struct Node {      int data;      struct Node\* left;      struct Node\* right;  };  struct Node\* createNode(int data) {      struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));      newNode->data = data;      newNode->left = NULL;      newNode->right = NULL;      return newNode;  }  struct Node\* insert(struct Node\* root, int data) {      if (root == NULL)          return createNode(data);      if (data < root->data)          root->left = insert(root->left, data);      else if (data > root->data)          root->right = insert(root->right, data);      return root;  }  void inorderTraversal(struct Node\* root) {      if (root != NULL) {          inorderTraversal(root->left);          printf("%d ", root->data);          inorderTraversal(root->right);      }  }  void preorderTraversal(struct Node\* root) {      if (root != NULL) {          printf("%d ", root->data);          preorderTraversal(root->left);          preorderTraversal(root->right);      }  }  void postorderTraversal(struct Node\* root) {      if (root != NULL) {          postorderTraversal(root->left);          postorderTraversal(root->right);          printf("%d ", root->data);      }  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050");      struct Node\* root = NULL;      int choice, data;      while (1) {          printf("\n\nBinary Search Tree Operations:\n");          printf("1. Insert Node\n");          printf("2. Inorder Traversal\n");          printf("3. Preorder Traversal\n");          printf("4. Postorder Traversal\n");          printf("5. Exit\n");          printf("Enter your choice: ");          scanf("%d", &choice);          switch (choice) {              case 1:                  printf("Enter data to insert: ");                  scanf("%d", &data);                  root = insert(root, data);                  break;              case 2:                  printf("Inorder Traversal: ");                  inorderTraversal(root);                  printf("\n");                  break;              case 3:                  printf("Preorder Traversal: ");                  preorderTraversal(root);                  printf("\n");                  break;              case 4:                  printf("Postorder Traversal: ");                  postorderTraversal(root);                  printf("\n");                  break;              case 5:                  printf("Exiting the program.\n");                  exit(0);              default:                  printf("Invalid choice! Please enter a valid option.\n");          }      }      return 0;  } |

|  |
| --- |
| **Output:** |
|  |

|  |
| --- |
| **Conclusion:** |
| We have successfully understood the applications and working of binary search tree. |

|  |
| --- |
| **Post lab:** |
| For the binary search tree created above, find the smallest and largest node in BST.  #include <stdio.h>  #include <stdlib.h>  struct Node {      int data;      struct Node\* left;      struct Node\* right;  };  struct Node\* createNode(int data) {      struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));      newNode->data = data;      newNode->left = NULL;      newNode->right = NULL;      return newNode;  }  struct Node\* insert(struct Node\* root, int data) {      if (root == NULL)          return createNode(data);      if (data < root->data)          root->left = insert(root->left, data);      else if (data > root->data)          root->right = insert(root->right, data);      return root;  }  // Function to find the smallest node in the BST  struct Node\* findSmallest(struct Node\* root) {      if (root == NULL)          return NULL;      while (root->left != NULL)          root = root->left;      return root;  }  // Function to find the largest node in the BST  struct Node\* findLargest(struct Node\* root) {      if (root == NULL)          return NULL;      while (root->right != NULL)          root = root->right;      return root;  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050");      struct Node\* root = NULL;      // Insert nodes into the BST      root = insert(root, 12);      root = insert(root, 24);      root = insert(root, 35);      root = insert(root, 87);      // Find the smallest and largest nodes      struct Node\* smallest = findSmallest(root);      struct Node\* largest = findLargest(root);      if (smallest != NULL)          printf("\nSmallest Node: %d\n", smallest->data);      else          printf("The tree is empty.\n");      if (largest != NULL)          printf("Largest Node: %d\n", largest->data);      else          printf("The tree is empty.\n");      return 0;  } |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |