|  |  |
| --- | --- |
| **Name** | **Hatim Yusuf Sawai** |
| **UID no.** | **2021300108** |
| **Experiment No.** | **5** |

|  |  |
| --- | --- |
| **AIM:** | **Program on BST (Binary Search Tree)** |
| **Program 1** | |
| **PROBLEM STATEMENT:** | Write A Menu-Driven program to show insertion, traversal & deletion operations on a Binary Search Tree. |
| **THEORY:** |  |
| **ALGORITHM:** |  |
| **PROBLEM SOLVING:** |  |
| **PROGRAM:** | **BTCheck.java:**  import *java*.*util*.*Scanner*;  import *bintreeds*.*BinTree*;  *public* *class* BTCheck {  *public* *static* void main(String[] args) {          Scanner sc = new Scanner(System.*in*);          BinTree bt = new BinTree();          int choice,flag;          int n,d;          while(true) {              System.*out*.println("Select 1 operation:\n1. Insert\t2. Delete\n3. PreOrder\t4. InOrder\t5. PostOrder");              choice = sc.nextInt();              switch(choice) {                  case 1:                      System.*out*.print("Enter no. of elements to insert: ");                      n = sc.nextInt();                      System.*out*.print("Enter the elements: ");                      for (int i = 0; i < n; i++) {                          bt.insert(sc.nextInt());                      }                      System.*out*.print("InOrder: ");                      bt.InOrder(bt.*root*);                      System.*out*.println();                      break;                  case 2:                      System.*out*.print("Enter the element to delete: ");                      d = sc.nextInt();                      bt.delete(bt.*root*,d);                      System.*out*.print("InOrder: ");                      bt.InOrder(bt.*root*);                      System.*out*.println();                      break;                  case 3:                      System.*out*.print("PreOrder: ");                      bt.PreOrder(bt.*root*);                      System.*out*.println();                      break;                  case 4:                      System.*out*.print("InOrder: ");                      bt.InOrder(bt.*root*);                      System.*out*.println();                      break;                  case 5:                      System.*out*.print("PostOrder: ");                      bt.PostOrder(bt.*root*);                      System.*out*.println();                      break;                  case 6:                      System.*out*.println("Size of the tree is " + bt.size(bt.*root*));                      break;                  default:                      System.*out*.println("Invalid choice!");              }              System.*out*.println("Do you want to continue?\n1. Yes\t2. No");              flag = sc.nextInt();              if (flag == 2) {                  break;              }          }          sc.close();      }  }  **BinTree.java:**  package *bintreeds*;  *public* *class* BinTree {  *class* Node {          int data;          Node left, right;          Node(int data) {              this.*data* = data;              left = right = null;          }      }  *public* Node root=null;  *public* void insert(int data) {          root = inserter(root, data);      }  *private* Node inserter(Node root, int data) {          if (root == null) {              root = new Node(data);              return root;          }          if (data < root.*data*) {              root.*left* = inserter(root.*left*, data);          } else if (data > root.*data*) {              root.*right* = inserter(root.*right*, data);          }          return root;      }  *public* void delete(Node root,int data) {          root = deleter(root, data);      }  *private* Node deleter(Node root, int data) {          if (root == null) {              return root;          }          if (data < root.*data*) {              root.*left* = deleter(root.*left*, data);          } else if (data > root.*data*) {              root.*right* = deleter(root.*right*, data);          } else {              if (root.*left* == null) {                  return root.*right*;              } else if (root.*right* == null) {                  return root.*left*;              }              root.*data* = minValue(root.*right*);              root.*right* = deleter(root.*right*, root.*data*);          }          return root;      }  *private* int minValue(Node root) {          int minv = root.*data*;          while (root.*left* != null) {              minv = root.*left*.*data*;              root = root.*left*;          }          return minv;      }  *public* int size(Node root) {          if (root == null)              return 0;          else              return (size(root.*left*) + 1 + size(root.*right*));      }  *public* void PreOrder(Node root) {          if (root!=null) {              System.*out*.print(root.*data* + " ");              PreOrder(root.*left*);              PreOrder(root.*right*);          }      }  *public* void InOrder(Node root) {          if (root!=null) {              InOrder(root.*left*);              System.*out*.print(root.*data* + " ");              InOrder(root.*right*);          }      }  *public* void PostOrder(Node root) {          if (root!=null) {              PostOrder(root.*left*);              PostOrder(root.*right*);              System.*out*.print(root.*data* + " ");          }      }  } |
| **OUTPUT:**  **Insertion & Traversal:**  **Deletion (All 3 cases in order):** | |
| **CONCLUSION:** | In this experiment, we learned how to perform insertion & deletion operations on a binary search tree. We also learned how to perform 3 types of traversals (preorder, inorder & postorder) on a BST. |