# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BESE 13****A**

# Lab 9: Implementation of Binary Search Tree

**Date: 24th November, 2023**

**Time: 10 am - 1 pm**

**Lab Engineer: Anum Asif**

# 

# Lab 9: Implementation of Binary Search Tree

**Tasks**

Implement the following operations of Binary Search Tree ADT. A skeleton file is already created for you

1. Write a main method which contains a menu to take user’s inputs. The menu will allow the user to do the following operations:
   1. Insert new data
   2. In-Order Traversal
   3. Pre-Order Traversal
   4. Post-Order Traversal
   5. Delete an item
2. For insertion operation, the user will give a value which will be added in the binary search tree at its correct position.
3. For in-order, pre-order, and post-order traversals, the user should see the traversed tree as output in correct order.
4. On removal of an item, the nodes should connect correctly so that it does not break the data hierarchy within the tree.
5. Test your program with different inputs from smaller array e.g. of size 10 to larger arrays of size 100.

**Solution**

|  |
| --- |
| Solution |
| Task 1 Screen Shots (execution: Insertion, 3 types of traversals, deletion):    Task 1 Code:  #include <iostream>  #include <cstdlib>  using namespace std;  class BinarySearchTree  {  private:      struct tree\_node      {          tree\_node \*left;          tree\_node \*right;          int data;      };  public:      tree\_node \*root;      BinarySearchTree()      {          root = nullptr;      }      bool isEmpty() const { return root == NULL; }      void print\_inorder();      void inorder(tree\_node \*);      void print\_preorder();      void preorder(tree\_node \*);      void print\_postorder();      void postorder(tree\_node \*);      void insert(int);      tree\_node \*remove(int d, tree\_node \*node)      {          if (node == nullptr)              return node;          if (d < node->data)              node->left = remove(d, node->left);          else if (d > node->data)              node->right = remove(d, node->right);          else          {              // Node to be deleted is found              // Node with only one child or no child              if (node->left == nullptr)              {                  tree\_node \*temp = node->right;                  delete node;                  return temp;              }              else if (node->right == nullptr)              {                  tree\_node \*temp = node->left;                  delete node;                  return temp;              }              // Node with two children: Get the inorder successor (smallest in the right subtree)              tree\_node \*temp = minValueNode(node->right);              // Copy the inorder successor's content to this node              node->data = temp->data;              // Delete the inorder successor              node->right = remove(temp->data, node->right);          }          return node;      }      tree\_node \*minValueNode(tree\_node \*node)      {          tree\_node \*current = node;          // Find the leftmost leaf          while (current && current->left != nullptr)              current = current->left;          return current;      }  };  void BinarySearchTree::insert(int d)  {      // Create a new node and set its data      tree\_node \*newNode = new tree\_node;      newNode->data = d;      newNode->left = nullptr;      newNode->right = nullptr;      // If the tree is empty, make this node the root      if (root == nullptr)      {          root = newNode;          return;      }      // Traverse the tree to find the correct position to insert      tree\_node \*current = root;      while (true)      {          if (d < current->data)          {              if (current->left == nullptr)              {                  current->left = newNode;                  break;              }              current = current->left;          }          else          {              if (current->right == nullptr)              {                  current->right = newNode;                  break;              }              current = current->right;          }      }  }  void BinarySearchTree::print\_inorder()  {      inorder(root);      cout << endl;  }  void BinarySearchTree::inorder(tree\_node \*p)  {      if (p != nullptr)      {          inorder(p->left);          cout << p->data << "\t";          inorder(p->right);      }  }  void BinarySearchTree::print\_preorder()  {      preorder(root);      cout << endl;  }  void BinarySearchTree::preorder(tree\_node \*p)  {      if (p != nullptr)      {          cout << p->data << "\t";          preorder(p->left);          preorder(p->right);      }  }  void BinarySearchTree::print\_postorder()  {      postorder(root);      cout << endl;  }  void BinarySearchTree::postorder(tree\_node \*p)  {      if (p != nullptr)      {          postorder(p->left);          postorder(p->right);          cout << p->data << "\t";      }  }  void insertIntoTree(BinarySearchTree \*b)  {      cout << "How many elements do you want to insert?" << endl;      int n;      cin >> n;      int val;      for (int i = 0; i < n; i++)      {          cout << "Enter element # " << i << ":"               << "\t";          cin >> val;          b->insert(val);      }  }  int main()  {      BinarySearchTree b;      while (true)      {          int menuOption = 0;          while (menuOption > 6 || menuOption < 1) // This loop can exit program          {              cout << "\t-----------------------------------------" << endl;              cout << "\t\tBinary Search Tree Menu" << endl;              cout << "\t-----------------------------------------" << endl;              cout << "\t1. Insert New Element" << endl;              cout << "\t2. In-Order Traversal" << endl;              cout << "\t3. Pre-Order Traversal" << endl;              cout << "\t4. Post-Order Traversal" << endl;              cout << "\t5. Delete an Element" << endl;              cout << "\t6. Exit" << endl;              cout << "\t-----------------------------------------" << endl;              cout << "Enter your choice (1-6): ";              cin >> menuOption;              if (menuOption == 6)              {                  return 0; // Exiting              }              switch (menuOption)              {              case 1:              {                  insertIntoTree(&b);                  break;              }              case 2:              {                  b.print\_inorder();                  break;              }              case 3:              {                  b.print\_preorder();                  break;              }              case 4:              {                  b.print\_postorder();                  break;              }              case 5:              {                  cout << "Enter element to remove:" << endl;                  int elem;                  cin >> elem;                  b.remove(elem, b.root);              }              }          }      }      getchar();  } |

### Deliverables

Compile a single word document by filling in the solution part and submit this Word file on LMS. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS.