Exercise

1. Define following methods in Example 01 by using stack.

o Preorder traversal

o Postorder travsersal

SOURCE CODE

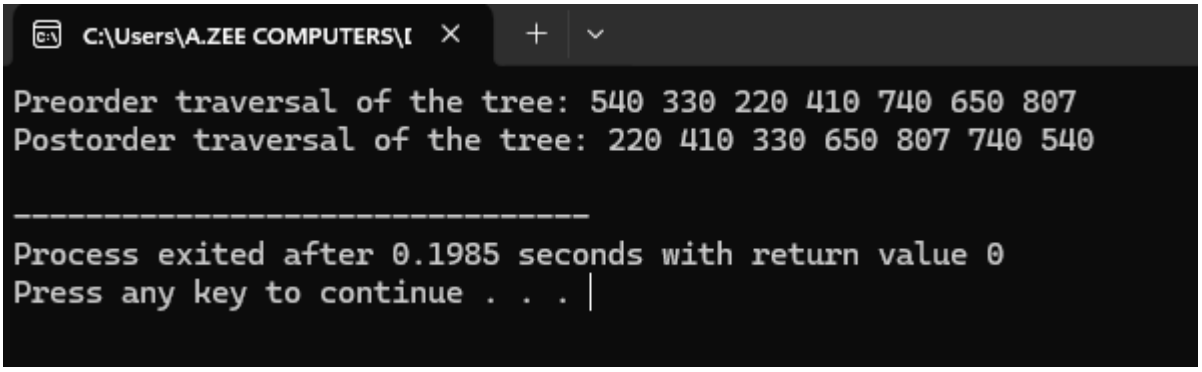
```
#include <iostream>
#include <stack>
using namespace std;
class Node {
public:
   int value;
   Node* left;
   Node* right;

   Node(int x) {
      value = x;
      left = right = nullptr;
   }
};

class Tamia_004 {
public:
   Node* root;
   Tamia_004() {
      root = nullptr;
   }

   void insert(int key) {
```

```
    root = insertRec(root, key);
  }

  void preorderTraversal() {
    if (root == nullptr) return;

    stack<Node*> s;
    s.push(root);

    while (!s.empty()) {
      Node* current = s.top();
      s.pop();
      cout << current->value << " ";

      if (current->right) s.push(current->right);
      if (current->left) s.push(current->left);
    }
    cout << endl;
  }

  void postorderTraversal() {
    if (root == nullptr) return;

    stack<Node*> s1, s2;
    s1.push(root);

    while (!s1.empty()) {
      Node* current = s1.top();
      s1.pop();
```

```
        s2.push(current);

        if (current->left) s1.push(current->left);
        if (current->right) s1.push(current->right);
      }

      while (!s2.empty()) {
        cout << s2.top()->value << " ";
        s2.pop();
      }
      cout << endl;
   }

private:
   Node* insertRec(Node* node, int key) {
      if (node == nullptr)
        return new Node(key);

      if (key < node->value)
        node->left = insertRec(node->left, key);
      else if (key > node->value)
        node->right = insertRec(node->right, key);

      return node;
   }
};

int main() {
   Tamia_004 bst;
```

```
    bst.insert(540);
    bst.insert(330);
    bst.insert(220);
    bst.insert(410);
    bst.insert(740);
    bst.insert(650);
    bst.insert(807);

    cout << "Preorder traversal of the tree: ";
    bst.preorderTraversal();
    cout << "Postorder traversal of the tree: ";
    bst.postorderTraversal();
    return 0;
}
```

OUTPUT



2. You are tasked with designing an employee management system for a small company. Each employee has a unique ID, name, and department. You are required to store and manage employee records in a way that allows quick insertion, deletion, and search operations based on the employee ID. You decide to use a Binary Search Tree (BST) to store the employee records. Each node in the BST will store: Employee ID (used as the key for the BST), Employee Name, Employee Department.

- Create a C++ class EmployeeNode representing each employee in the BST. Each node should store: int id (Employee ID), string name (Employee Name), string department (Employee Department), A

pointer to the left child (EmployeeNode* left), a pointer to the right child (EmployeeNode* right).

- Create a class EmployeeBST with the following member functions:

o insert(int id, string name, string department): Inserts a new employee into the BST.

o search(int id): Searches for an employee by ID. If found, return their name and department; otherwise, print an appropriate message.

o deleteNode(int id): Deletes an employee from the BST based on their ID.

o inOrderTraversal(): Prints all employees in ascending order of their ID, showing their ID, name, and department.

o findMin(): Returns the name and department of the employee with the smallest ID.

o findMax(): Returns the name and department of the employee with the largest ID.

- Also, Handle edge cases such as: Inserting an employee with a duplicate ID, Deleting an employee that does not exist, Deleting nodes with no children, one child, and two children, Handling an empty tree for search or delete operations.

SOURCE CODE

```
#include <iostream>
using namespace std;
class EmployeeNode {
public:
   int id;
   string name;
   string dept;
   EmployeeNode* left;
   EmployeeNode* right;

   EmployeeNode(int ID, const string& n, const string& department){
   id = ID;
   name = n;
   dept = department;
   left = right = nullptr;
   }
};

class Tamia_004 {
private:
   EmployeeNode* root;
```

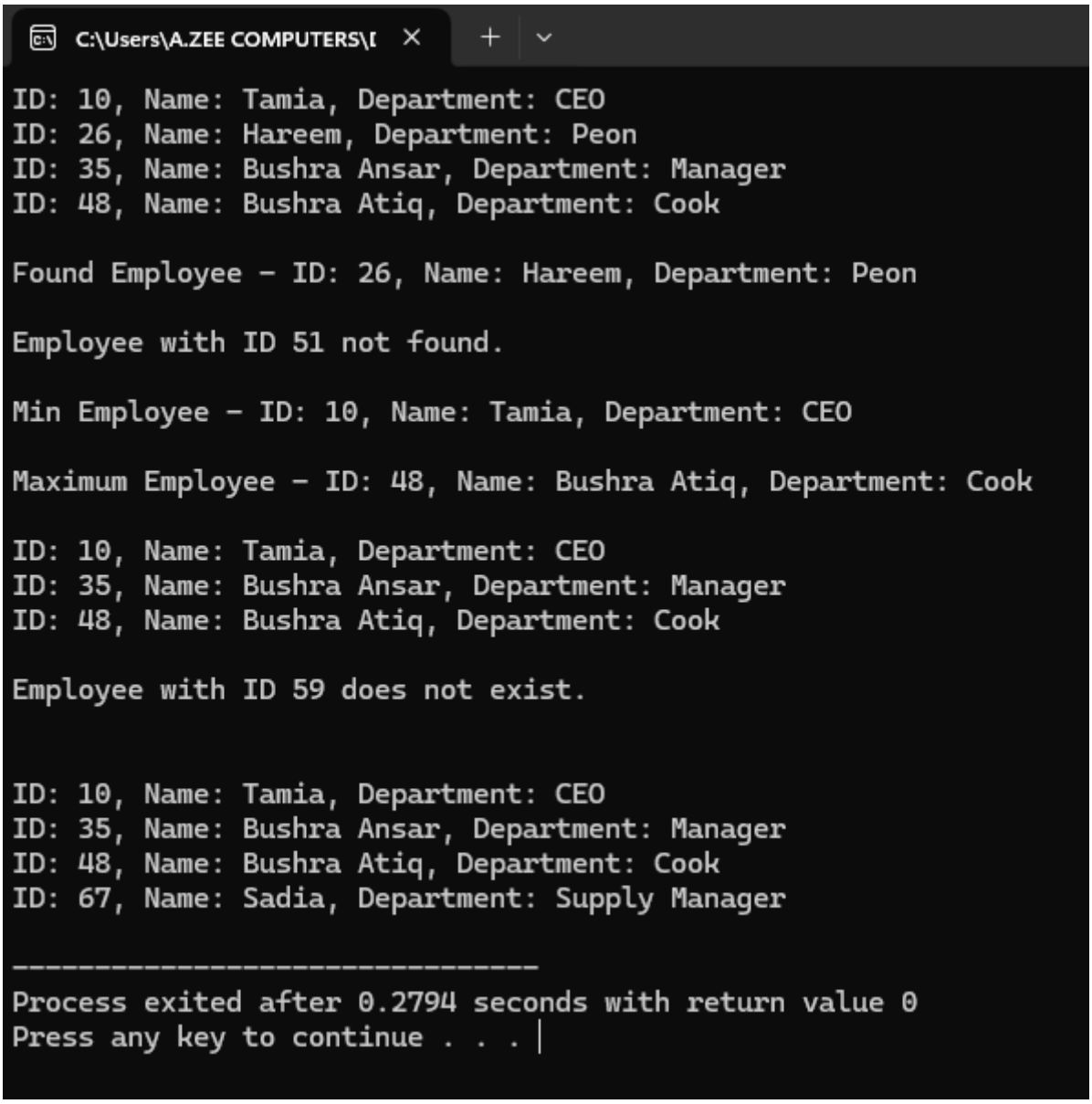
```
public:
     Tamia_004 (){
   root = nullptr;
}

   void insert(int id, const string& name, const string& dept) {
      EmployeeNode** node = &root;
      while (*node != nullptr) {
         if (id < (*node)->id) {
            node = &(*node)->left;
         } else if (id > (*node)->id) {
            node = &(*node)->right;
         } else {
            cout << "Employee with ID " << id << " already exists.";
            return;
         }
      }
      *node = new EmployeeNode(id, name, dept);
   }

   void search(int id) {
      EmployeeNode* node = root;
      while (node != nullptr) {
         if (node->id == id) {
            cout << "Found Employee - ID: " << node->id << ", Name: " << node->name << ", Department: "
<< node->dept << endl;
            return;
         } else if (id < node->id) {
            node = node->left;
         } else {
            node = node->right;
         }
      }
      cout << "Employee with ID " << id << " not found." << endl;
   }

   void deleteNode(int id) {
      EmployeeNode** node = &root;
      while (*node != nullptr && (*node)->id != id) {
         if (id < (*node)->id) {
            node = &(*node)->left;
         } else {
            node = &(*node)->right;
```

```
        }
    }

    if (*node == nullptr) {
        cout << "Employee with ID " << id << " does not exist." << endl;
        return;
    }

    EmployeeNode* temp = *node;
    if ((*node)->left == nullptr) {
        *node = (*node)->right;
    } else if ((*node)->right == nullptr) {
        *node = (*node)->left;
    } else {
        EmployeeNode** minNode = &(*node)->right;
        while ((*minNode)->left != nullptr) {
            minNode = &(*minNode)->left;
        }
        (*node)->id = (*minNode)->id;
        (*node)->name = (*minNode)->name;
        (*node)->dept = (*minNode)->dept;
        temp = *minNode;
        *minNode = (*minNode)->right;
    }
    delete temp;
}

void inOrderTraversal() {
    struct StackEntry {
        EmployeeNode* node;
        bool visited;
    };

    StackEntry stack[100];
    int stackSize = 0;
    EmployeeNode* current = root;

    while (current != nullptr || stackSize > 0) {
        while (current != nullptr) {
            stack[stackSize++] = {current, false};
            current = current->left;
        }

        StackEntry& top = stack[--stackSize];
```

```
        current = top.node;

        if (!top.visited) {
            top.visited = true;
            cout << "ID: " << current->id << ", Name: " << current->name << ", Department: " << current-
>dept << endl;
            current = current->right;
        } else {
            current = nullptr;
        }
    }
  }

  void findMin() {
    EmployeeNode* node = root;
    if (!node) {
        cout << "Tree is empty." << endl;
        return;
    }
    while (node->left != nullptr) {
        node = node->left;
    }
    cout << "Min Employee - ID: " << node->id << ", Name: " << node->name << ", Department: " <<
node->dept << endl;
  }

  void findMax() {
    EmployeeNode* node = root;
    if (!node) {
        cout << "Tree is empty.\n";
        return;
    }
    while (node->right != nullptr) {
        node = node->right;
    }
    cout << "Maximum Employee - ID: " << node->id << ", Name: " << node->name << ", Department: "
<< node->dept << endl;
  }
};

int main() {
  Tamia_004 bst;

  bst.insert(10, "Tamia", "CEO");
```

```
    bst.insert(35, "Bushra Ansar", "Manager");
    bst.insert(48, "Bushra Atiq", "Cook");
    bst.insert(26, "Hareem", "Peon");

    bst.inOrderTraversal();

    cout << endl;

    bst.search(26);

    cout << endl;

    bst.search(51);

    cout << endl;

    bst.findMin();

    cout << endl;

    bst.findMax();

    cout << endl;

    bst.deleteNode(26);

    bst.inOrderTraversal();

    cout << endl;

    bst.deleteNode(59);

    cout << endl;

    bst.insert(67, "Sadia", "Supply Manager");

    cout << endl;

    bst.inOrderTraversal();

    return 0;
}
```

OUTPUT



3. Given the two nodes of a binary search tree, return their least common ancestor.

SOURCE CODE

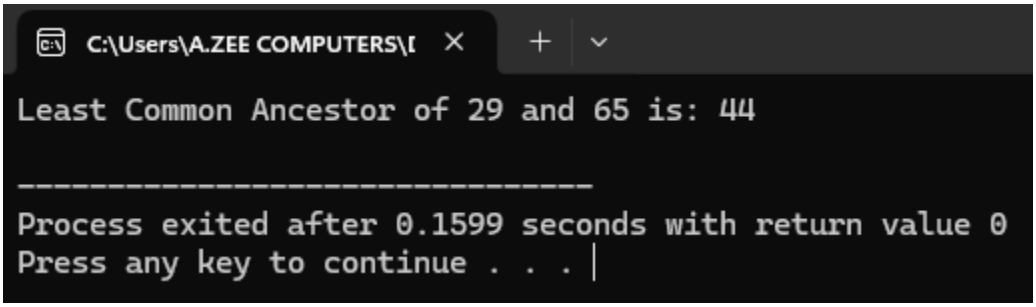
```
#include <iostream>

using namespace std;

class Node {

public:

   int value;

   Node* left;
```

```
    Node* right;

    Node(int x) {
        value = x;
        left = right = nullptr;
    }
};

class Tamia_004 {
public:
    Node* root;

    Tamia_004() {
        root = nullptr;
    }

    Node* insert(Node* root, int value) {
        if (root == nullptr) {
            return new Node(value);
        }
        if (value < root->value) {
            root->left = insert(root->left, value);
        } else if (value > root->value) {
            root->right = insert(root->right, value);
        }
        return root;
    }

    Node* lca(Node* root, Node* p, Node* q) {
```

```
        if (root == nullptr) return nullptr;


        if (p->value < root->value && q->value < root->value) {
            return lca(root->left, p, q);
        }
        if (p->value > root->value && q->value > root->value) {
            return lca(root->right, p, q);
        }
        return root;
    }
};


int main() {
    Tamia_004 tree;
    tree.root = tree.insert(tree.root, 44);
    tree.insert(tree.root, 29);
    tree.insert(tree.root, 65);
    tree.insert(tree.root, 23);
    tree.insert(tree.root, 34);


    Node* p = tree.root->left;
    Node* q = tree.root->right;


    Node* lca = tree.lca(tree.root, p, q);
    if (lca) {
        cout << "Least Common Ancestor of " << p->value << " and " << q->value << " is: " << lca->value <<
endl;
    } else {
        cout << "Least Common Ancestor not found." << endl;
```

```
    }


    return 0;
}
```

OUTPUT



4. Given the root of a binary search tree, recursively find the sum of all nodes of the tree.

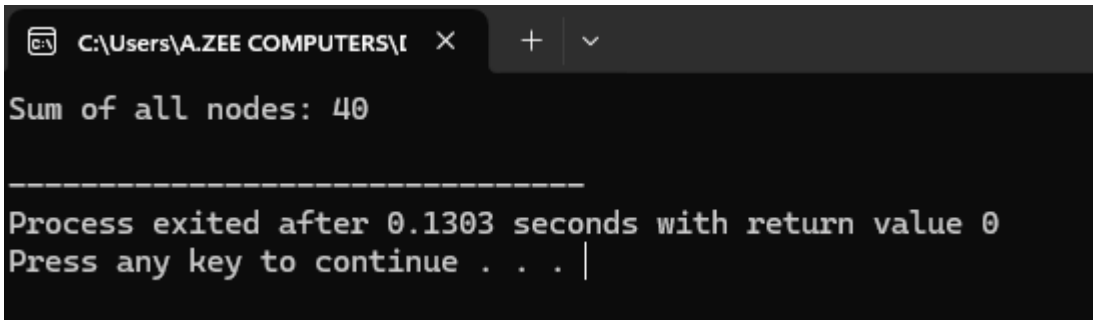
SOURCE CODE

```
#include <iostream>
using namespace std;
class Node {
public:
    int value;
    Node* left;
    Node* right;

    Node(int x) {
        value = x;
        left = right = nullptr;
    }
};

class Tamia_004 {
public:
```

```
    Node* root;

    Tamia_004() {
        root = nullptr;
    }

    Node* insert(Node* root, int value) {
        if (root == nullptr) {
            return new Node(value);
        }
        if (value < root->value) {
            root->left = insert(root->left, value);
        } else if (value > root->value) {
            root->right = insert(root->right, value);
        }
        return root;
    }

    int sumOfNodes(Node* root) {
        if (root == nullptr) return 0;

        return root->value + sumOfNodes(root->left) + sumOfNodes(root->right);
    }
};

int main() {
    Tamia_004 T;
    T.root = T.insert(T.root, 10);
    T.insert(T.root, 5);
```

```
    T.insert(T.root, 15);
    T.insert(T.root, 3);
    T.insert(T.root, 7);
    int sum = T.sumOfNodes(T.root);
    cout << "Sum of all nodes: " << sum << endl;
    return 0;
}
```

OUTPUT



5. Given the root of a Binary Search Tree (BST), return the minimum difference between the values of any two different nodes in the tree.

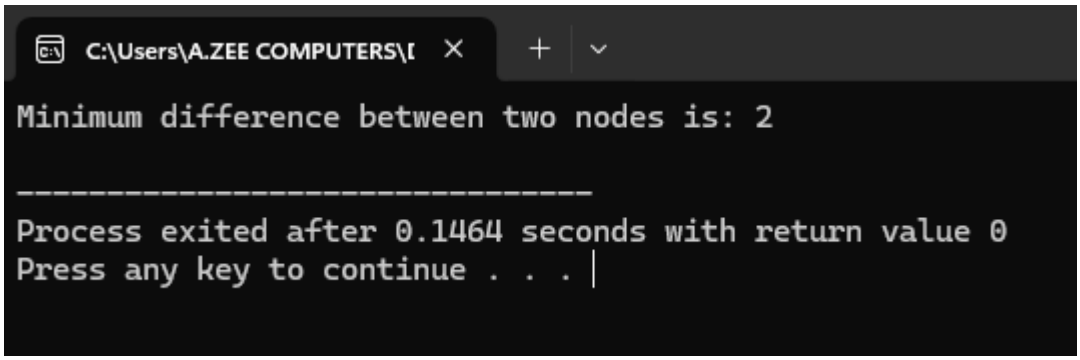
SOURCE CODE

```
#include <iostream>
using namespace std;

class Node {
public:
    int value;
    Node* left;
    Node* right;

    Node(int x) {
        value = x;
        left = right = nullptr;
```

```
    }
};

class Tamia_004 {
public:
    Node* root;
    int minDiff;
    Node* prev;

    Tamia_004() {
        root = prev = nullptr;
        minDiff = INT_MAX;
    }

    Node* insert(Node* root, int value) {
        if (root == nullptr) {
            return new Node(value);
        }
        if (value < root->value) {
            root->left = insert(root->left, value);
        } else if (value > root->value) {
            root->right = insert(root->right, value);
        }
        return root;
    }

    void inorderTraversal(Node* root) {
        if (root == nullptr) return;
```

```
        inorderTraversal(root->left);

        if (prev != nullptr) {
            minDiff = min(minDiff, root->value - prev->value);
        }
        prev = root;

        inorderTraversal(root->right);
    }

    int MinimumDifference(Node* root) {
        inorderTraversal(root);
        return minDiff;
    }
};

int main() {
    Tamia_004 tree;
    tree.root = tree.insert(tree.root, 10);
    tree.insert(tree.root, 5);
    tree.insert(tree.root, 15);
    tree.insert(tree.root, 2);
    tree.insert(tree.root, 7);

    int md = tree.MinimumDifference(tree.root);
    cout << "Minimum difference between two nodes is: " << md << endl;
    return 0;
}
```

OUTPUT


C:\Users\A.ZEE COMPUTERS\[
Minimum difference between two nodes is: 2

Process exited after 0.1464 seconds with return value 0
Press any key to continue . . .