****

**PAMANTASAN NG LUNGSOD NG MAYNILA**

College of Information Systems and Technology Management (CISTM)

**ICC 0104-1 – Data Structures and Algorithms**

*A.Y. 2023- 2024*

**Group 4**: Searching

**Submitted by**:

Abundo, Jonalene Ryza B.

Dela Peña, Daniella Mae N.

Diamzon, Momer Ailes M.

Lau, Trisha Mae R.

Mahusay, Lindsay G.

Matanga, Sophia Vien V.

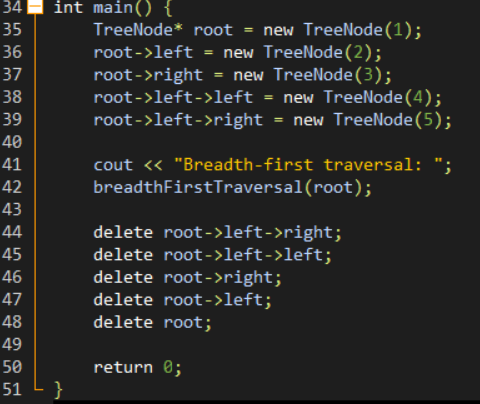
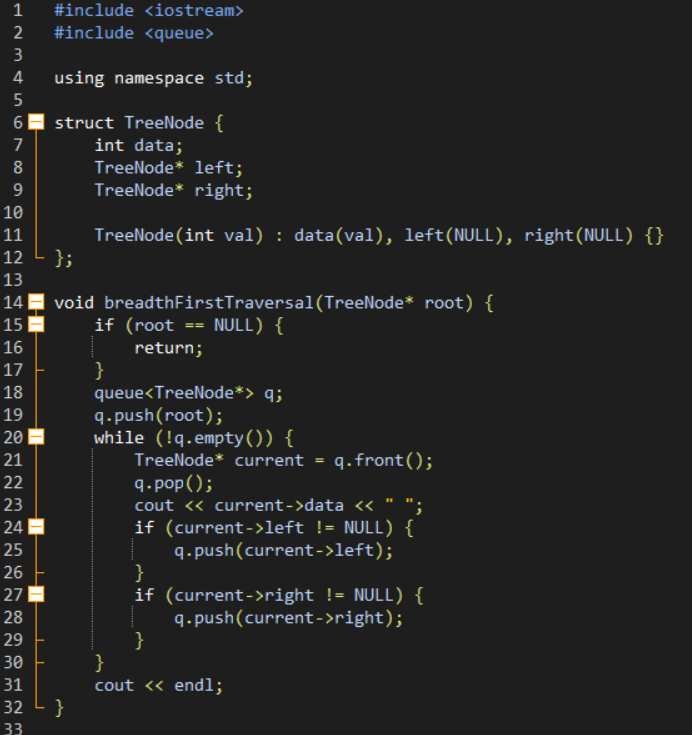
Rivera, Ramyll C.

Sibayan, Joan F.

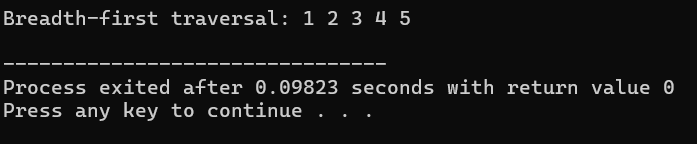
BSCS 1-1

1. **Breadth-First**

Source Code in C++



Output

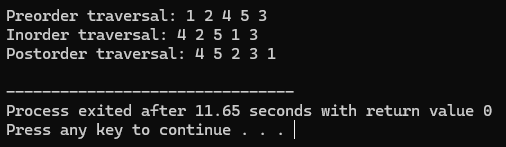


* This code shows how to create a breadth-first traversal algorithm for a binary tree in C++. The traversal order makes sure all nodes at each level are viewed before progressing to the next level, allowing for a level-by-level examination of the tree structure.

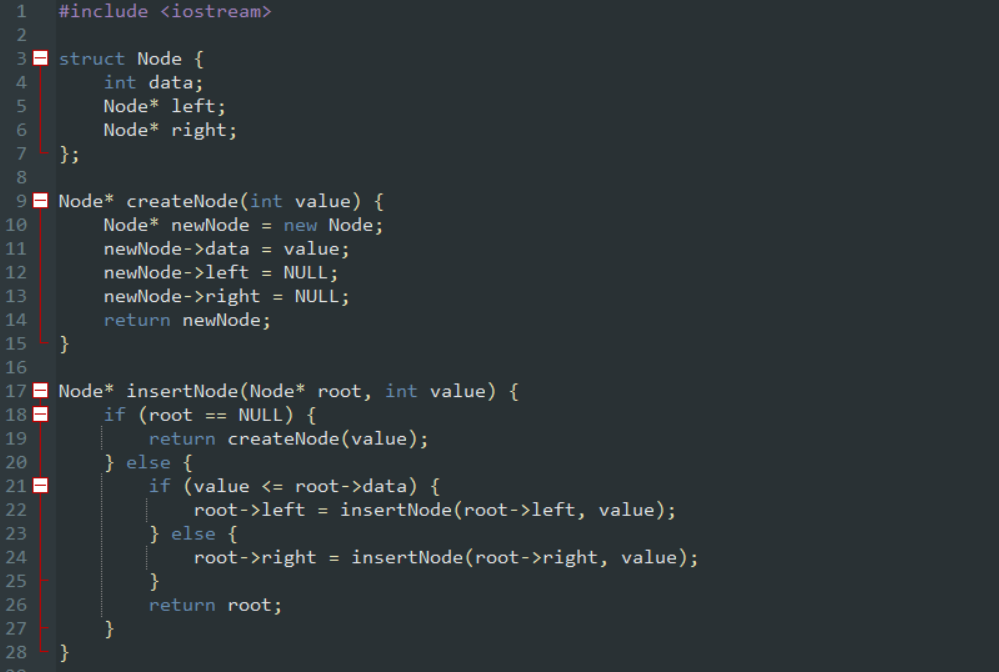
1. **Depth-First**

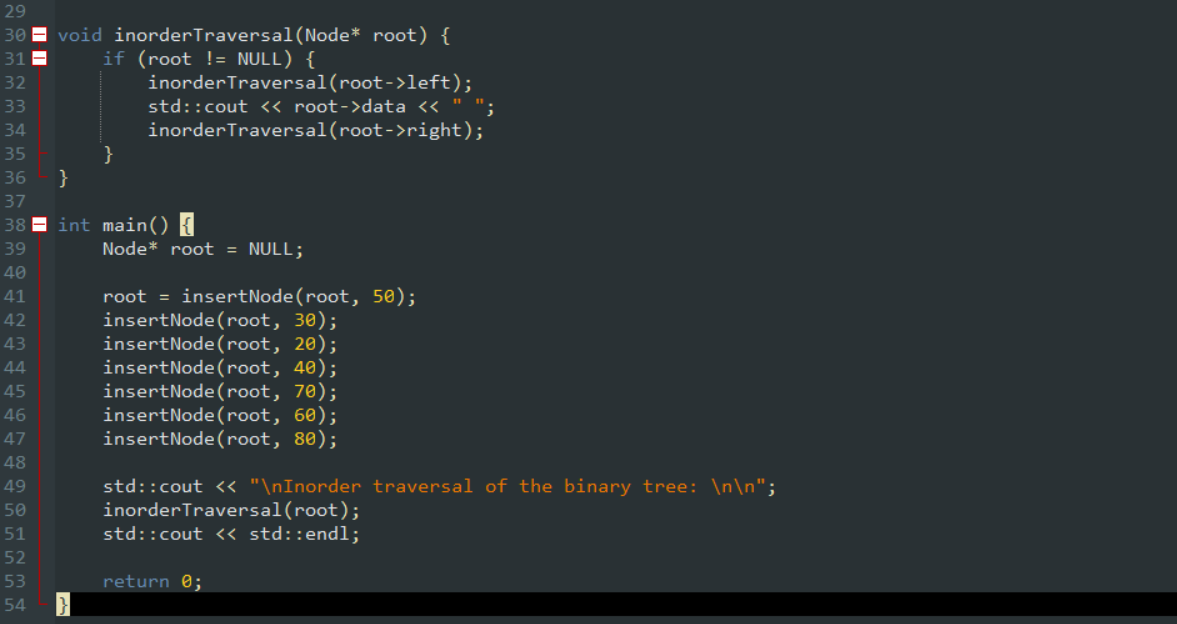
Source Code in C++

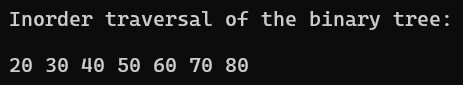
| #include <iostream>  #include <cstdlib>  struct TreeNode {  int data;  struct TreeNode\* left;  struct TreeNode\* right;  };  struct StackNode {  struct TreeNode\* data;  struct StackNode\* next;  };  struct TreeNode\* createNode(int data) {  struct TreeNode\* newNode = (struct TreeNode\*)std::malloc(sizeof(struct TreeNode));  newNode->data = data;  newNode->left = NULL;  newNode->right = NULL;  return newNode;  }  struct StackNode\* createStackNode(struct TreeNode\* data) {  struct StackNode\* newNode = (struct StackNode\*)std::malloc(sizeof(struct StackNode));  newNode->data = data;  newNode->next = NULL;  return newNode;  }  void push(struct StackNode\*\* top, struct TreeNode\* data) {  struct StackNode\* newNode = createStackNode(data);  newNode->next = \*top;  \*top = newNode;  }  struct TreeNode\* pop(struct StackNode\*\* top) {  if (\*top == NULL) return NULL;  struct StackNode\* temp = \*top;  \*top = (\*top)->next;  struct TreeNode\* popped = temp->data;  free(temp);  return popped;  }  int isEmpty(struct StackNode\* top) {  return top == NULL;  }  void preorder(struct TreeNode\* root) {  if (root == NULL) return;  struct StackNode\* stack = NULL;  struct TreeNode\* current = root;  while (current != NULL || !isEmpty(stack)) {  while (current != NULL) {  std::cout << current->data << " ";  push(&stack, current);  current = current->left;  }  current = pop(&stack);  current = current->right;  }  }  void inorder(struct TreeNode\* root) {  if (root == NULL) return;  struct StackNode\* stack = NULL;  struct TreeNode\* current = root;  while (current != NULL || !isEmpty(stack)) {  while (current != NULL) {  push(&stack, current);  current = current->left;  }  current = pop(&stack);  std::cout << current->data << " ";  current = current->right;  }  }  void postorder(struct TreeNode\* root) {  if (root == NULL) return;  struct StackNode\* stack = NULL;  struct TreeNode\* current = root;  struct TreeNode\* peek = NULL;  struct TreeNode\* rightChild = NULL;  do {  while (current != NULL) {  if (current->right) {  push(&stack, current->right);  }  push(&stack, current);  current = current->left;  }  current = pop(&stack);  peek = isEmpty(stack) ? NULL : stack->data;  if (current->right != NULL && current->right == peek) {  rightChild = pop(&stack);  push(&stack, current);  current = current->right;  } else {  std::cout << current->data << " ";  current = NULL;  }  } while (!isEmpty(stack));  }  int main() {  struct TreeNode\* root = createNode(1);  root->left = createNode(2);  root->right = createNode(3);  root->left->left = createNode(4);  root->left->right = createNode(5);  std::cout << "Preorder traversal: ";  preorder(root);  std::cout << "\n";  std::cout << "Inorder traversal: ";  inorder(root);  std::cout << "\n";  std::cout << "Postorder traversal: ";  postorder(root);  std::cout << "\n";  return 0;  } |
| --- |

Output

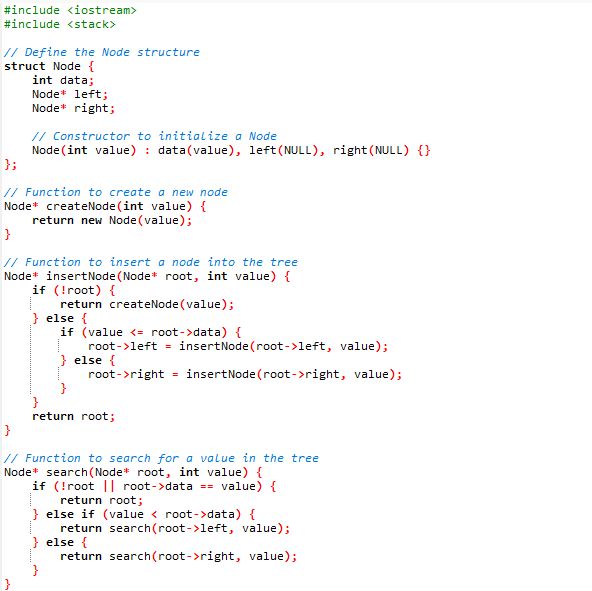
1. **Tree Insertion**

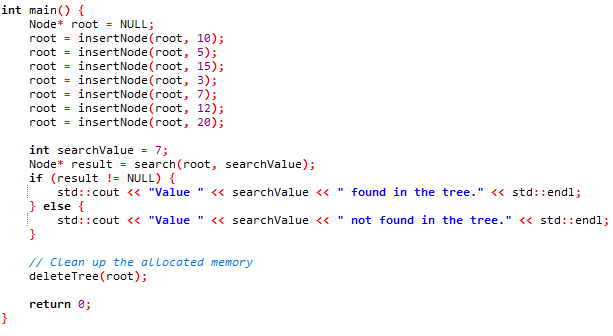
Source Code in C++



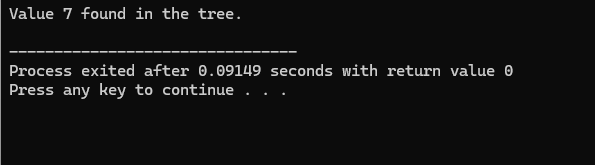
Output

1. **Tree Search**

Source Code in C++

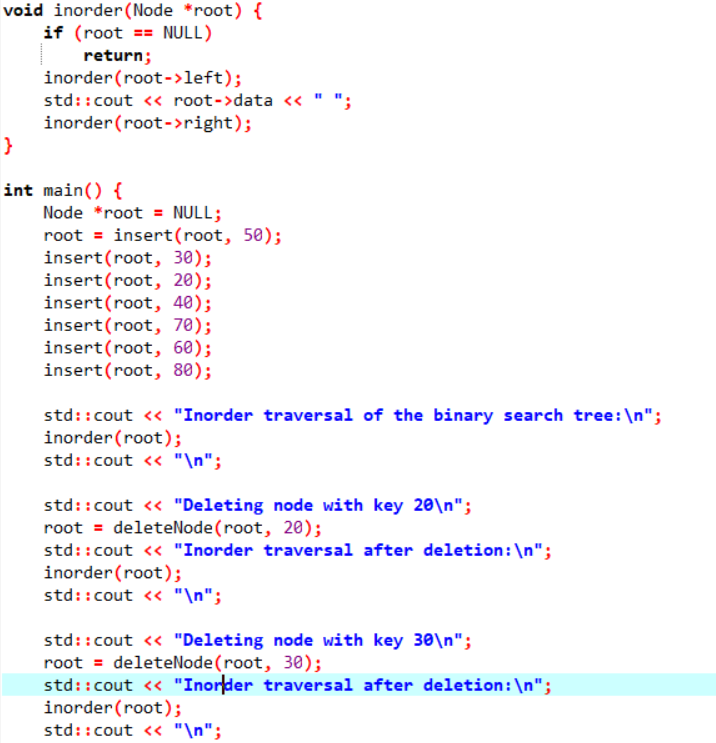
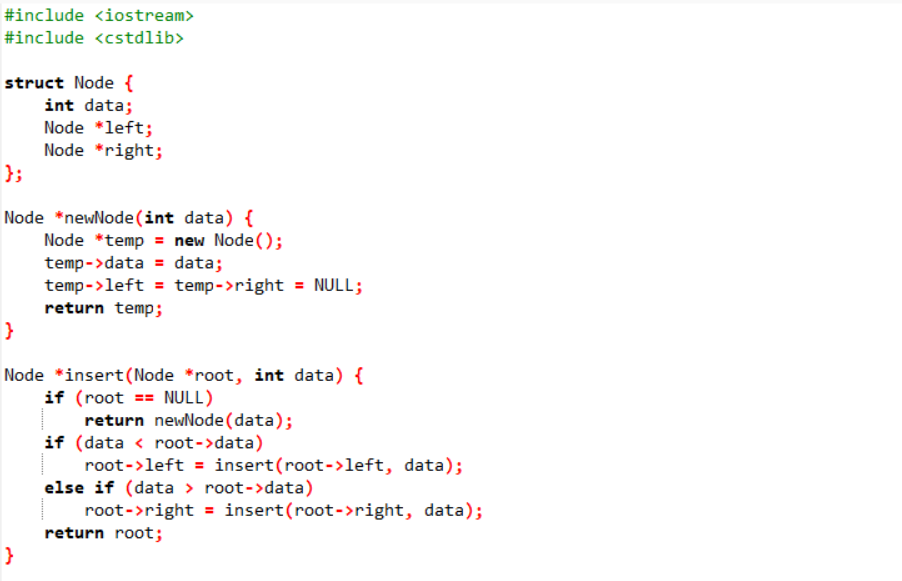


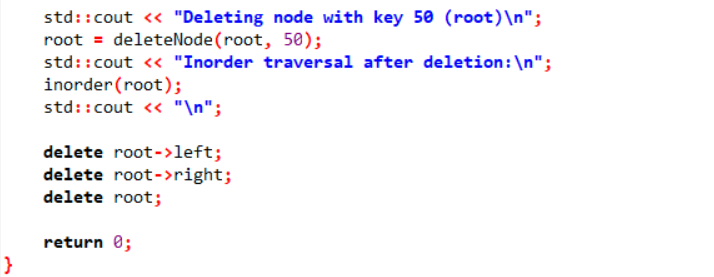
Output

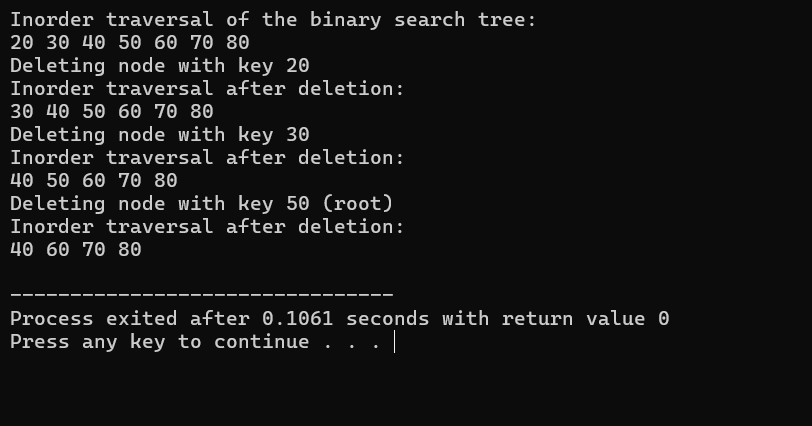


1. **Tree Deletion**

Source Code in C++



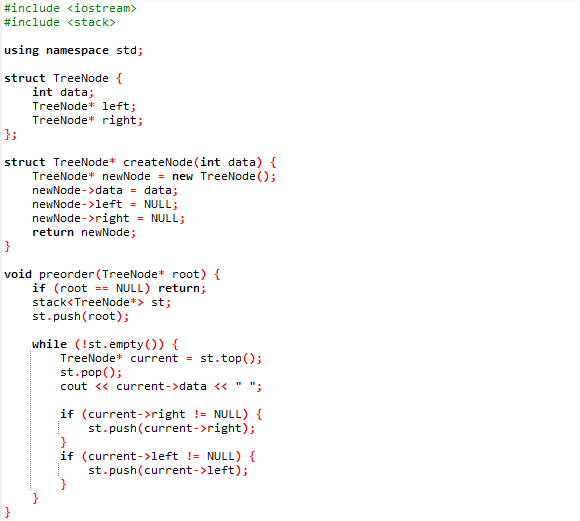


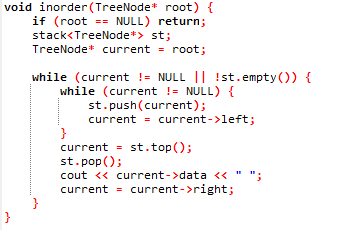
Output

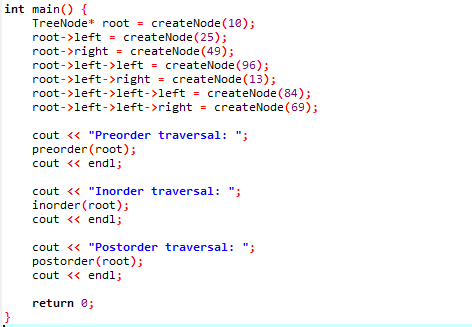
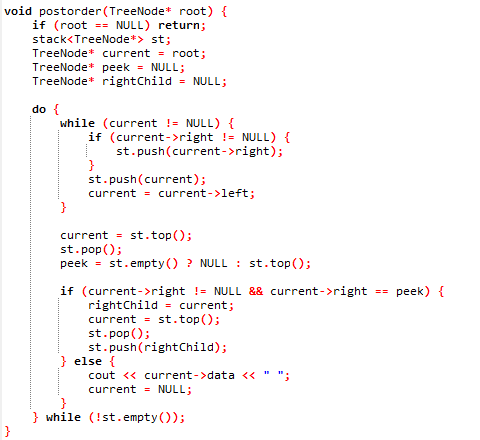
* This code presents how to remove a node from a binary tree using C++. The deletion of a node in the tree looks after the attributes of a binary tree. It ensures that different situations are effectively addressed so that the tree remains balanced and ordered.

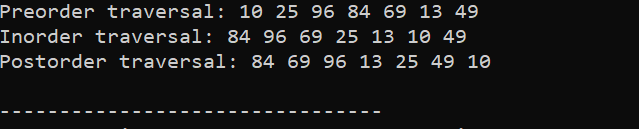
1. **treeE1**

Source Code in C++



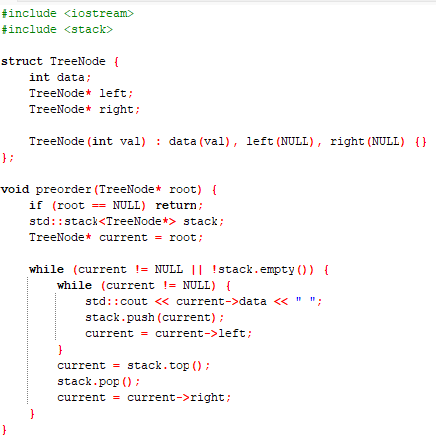


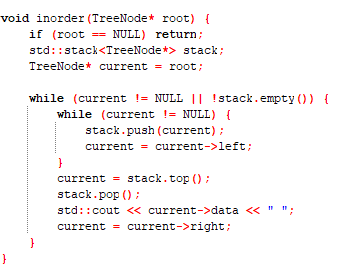


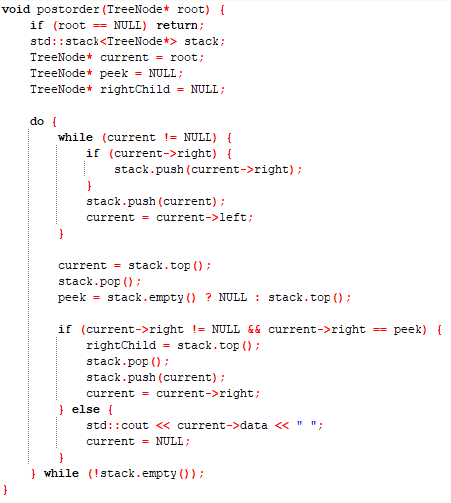
Output  


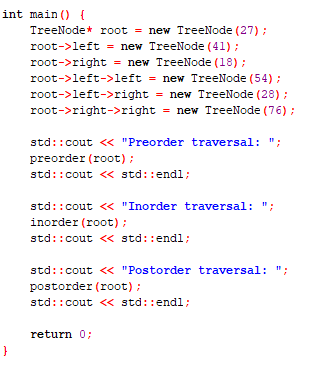
1. **treeE2**

Source Code in C++

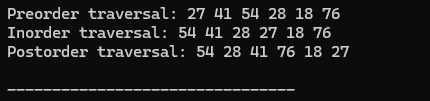






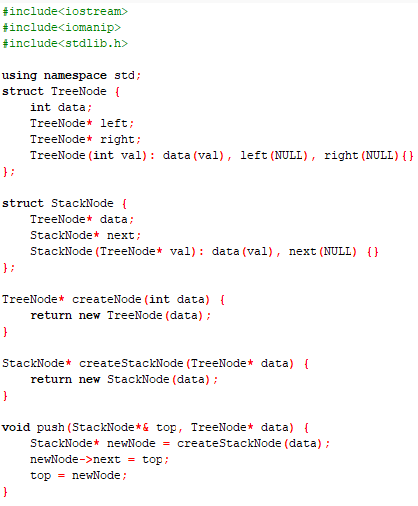


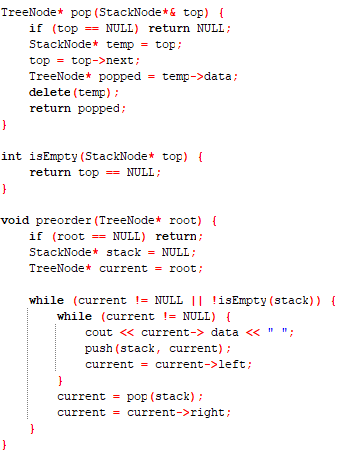
Output

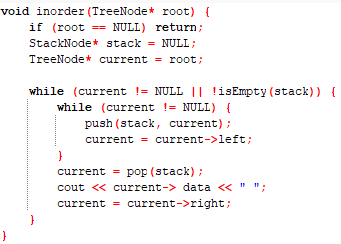


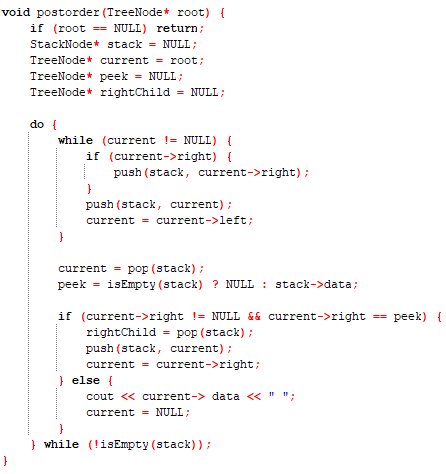
1. **treeE3**

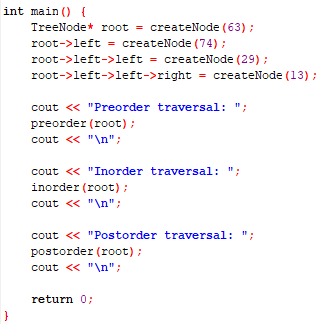
Source Code in C++



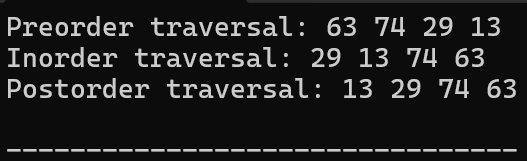






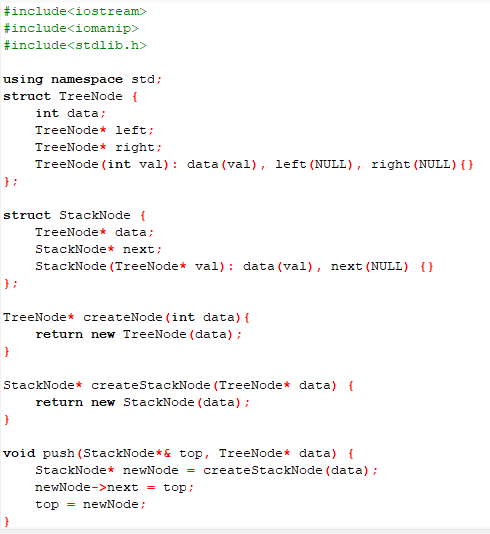


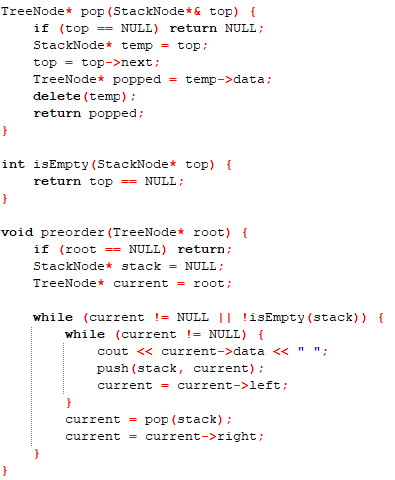
Output

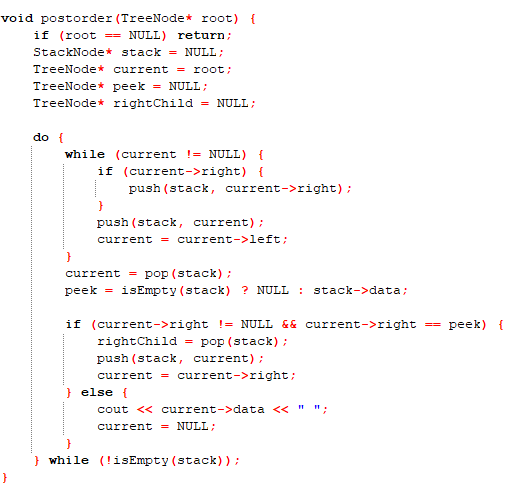
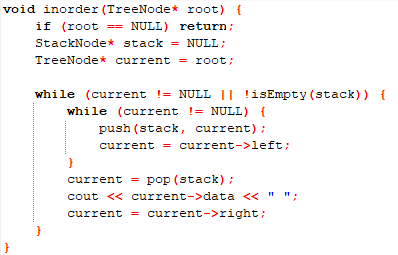


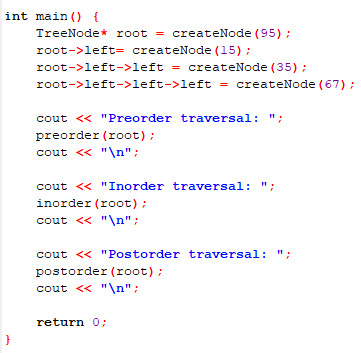
1. **treeE4**

Source Code in C++

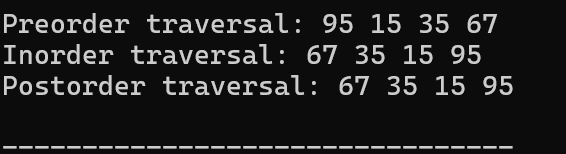






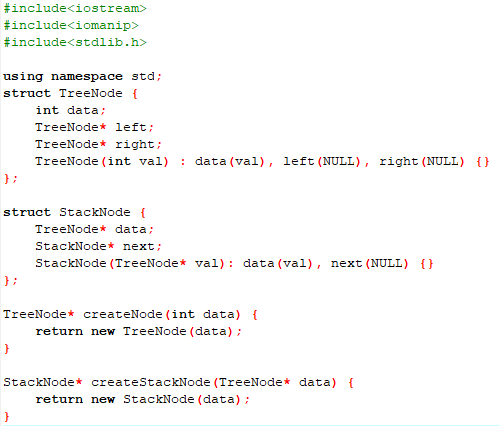


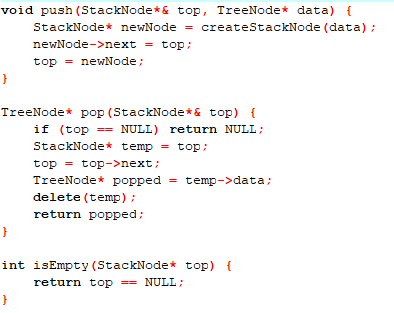
Output

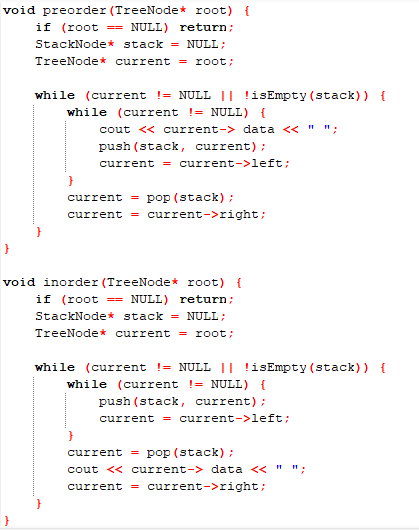


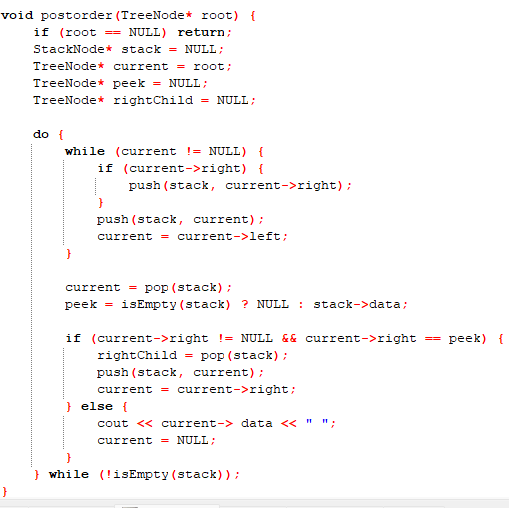
1. **treeE5**

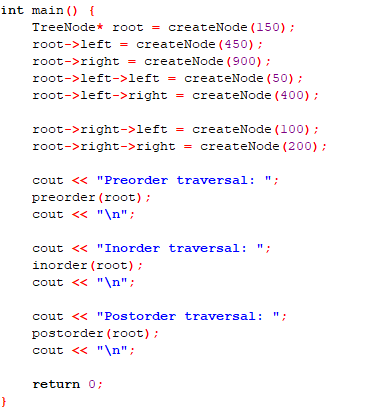
Source Code in C++











Output

