**Submitted to**: Mam Irsha Qureshi

**Submitted by:** Eman Tahir

**Roll no**. 2023-bs-ai-015

**Section**: A

**Department:** Artificial Intelligence

**Subject:** Data Structure

**Final Assignment**

**Doubly Linked List**

•Write a program to delete the first node in a doubly linked list.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node\* prev;

};

void deleteNode(Node\*\* head\_node, int position) {

if (\*head\_node== nullptr) return;

Node\* temp = \*head\_node;

if (position == 0) {

\*head\_node = temp->next;

if (\*head\_node != nullptr) {

(\*head\_node)->prev = nullptr;

}

delete temp;

return;

}

for (int i = 0; temp != nullptr && i < position; i++) {

temp = temp->next;

}

if (temp == nullptr) return;

if (temp->next != nullptr) {

temp->next->prev = temp->prev;

}

if (temp->prev != nullptr) {

temp->prev->next = temp->next;

}

delete temp;

}

void printList(Node\* node) {

while (node != nullptr) {

cout << node->data << " <-> ";

node = node->next;

}

}

int main() {

Node\* head = nullptr;

head = new Node();

head->data = 10;

head->next = new Node();

head->next->data = 20;

head->next->prev = head;

head->next->next = new Node();

head->next->next->data = 30;

head->next->next->prev = head->next;

head->next->next->next = nullptr;

cout << "Original list: \n";

printList(head);

cout<<"\n";

deleteNode(&head, 0);

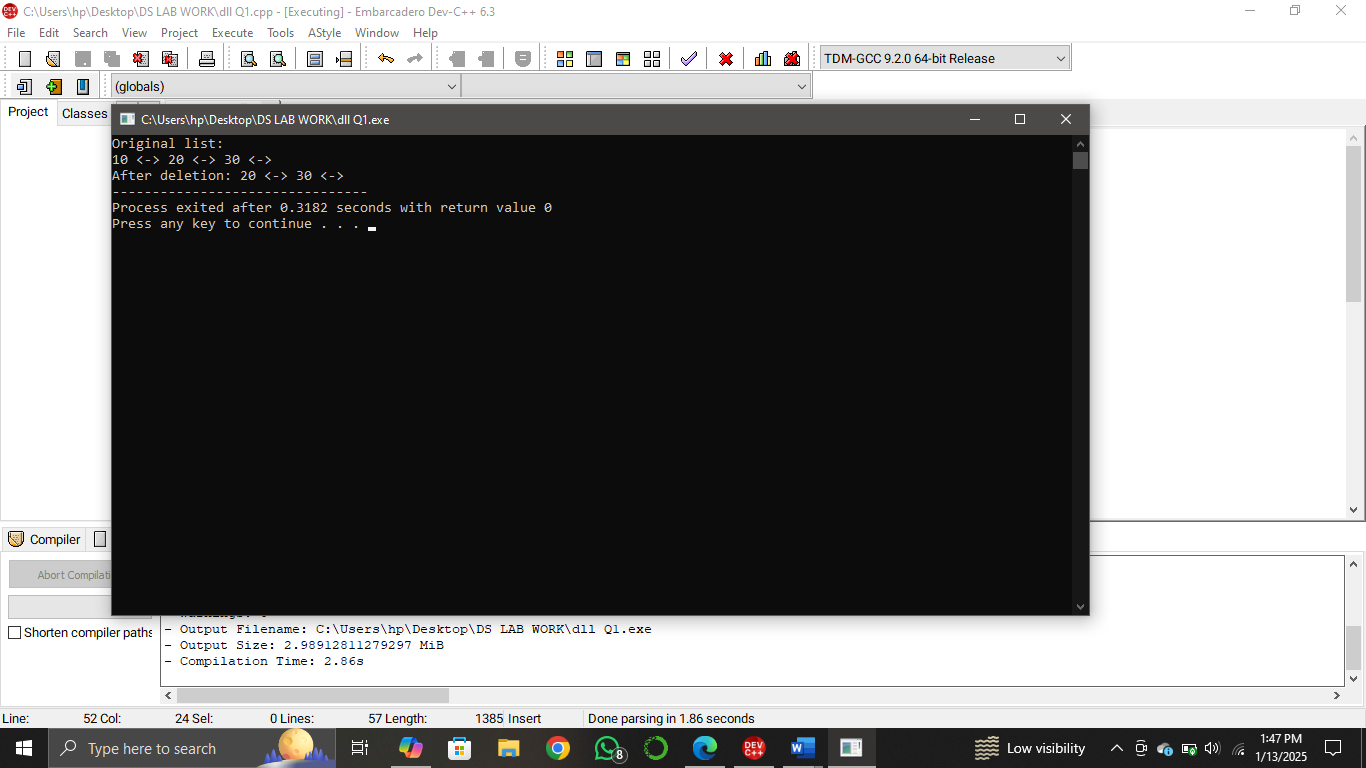
cout << "After deletion: ";

printList(head);

return 0;

}

**Output**:



• How can you delete the last node in a doubly linked list? Write the code.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node\* prev;

};

void deleteNode(Node\*\* head\_node, int position) {

if (\*head\_node== nullptr) return;

Node\* temp = \*head\_node;

if (position == 0) {

\*head\_node = temp->next;

if (\*head\_node != nullptr) {

(\*head\_node)->prev = nullptr;

}

delete temp;

return;

}

for (int i = 0; temp != nullptr && i < position; i++) {

temp = temp->next;

}

if (temp == nullptr) return;

if (temp->next != nullptr) {

temp->next->prev = temp->prev;

}

if (temp->prev != nullptr) {

temp->prev->next = temp->next;

}

delete temp;

}

void printList(Node\* node) {

while (node != nullptr) {

cout << node->data << " <-> ";

node = node->next;

}

}

int main() {

Node\* head = nullptr;

head = new Node();

head->data = 10;

head->next = new Node();

head->next->data = 20;

head->next->prev = head;

head->next->next = new Node();

head->next->next->data = 30;

head->next->next->prev = head->next;

head->next->next->next = nullptr;

cout << "Original list: \n";

printList(head);

cout<<"\n";

deleteNode(&head, 2);

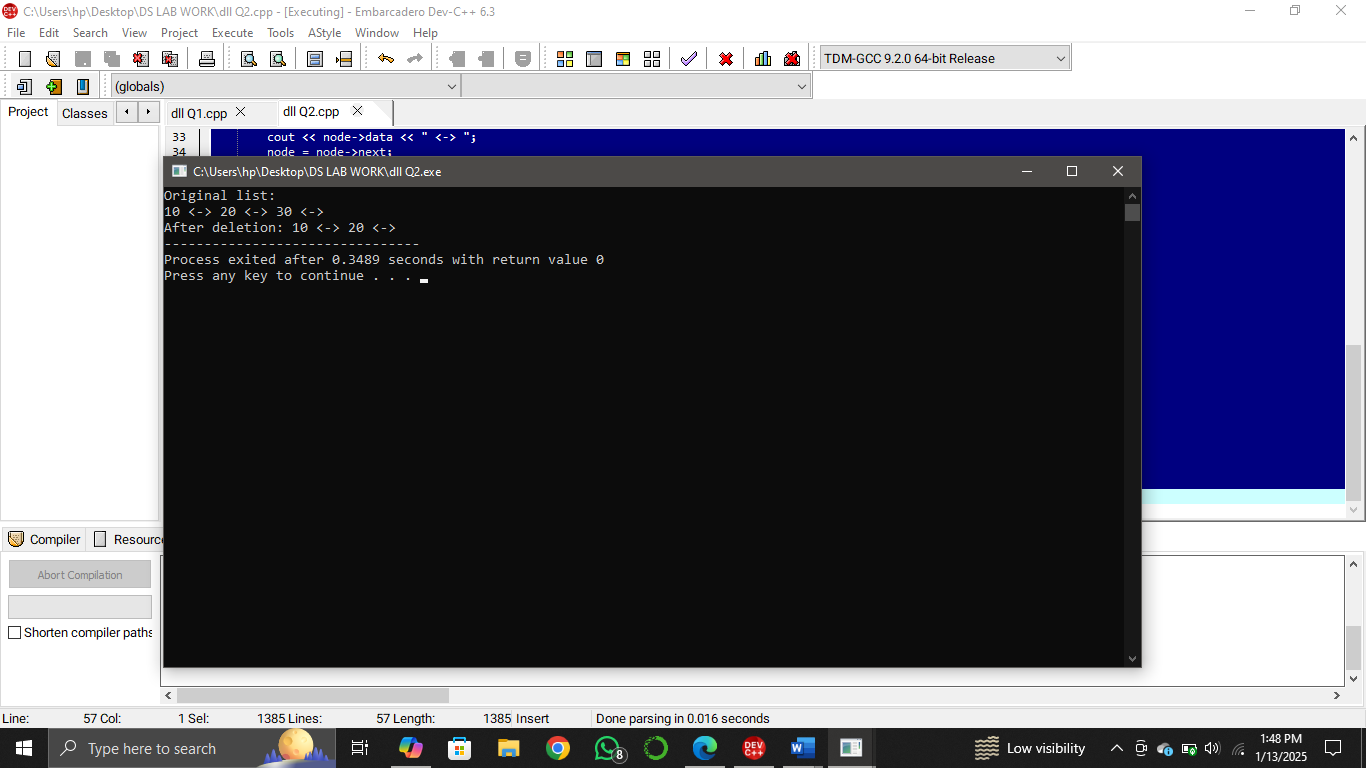
cout << "After deletion: ";

printList(head);

return 0;

}

**Output**:



• Write code to delete a node by its value in a doubly linked list.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node\* prev;

};

void deleteByValue(Node\*\* head\_node, int value) {

if (\*head\_node == nullptr) return;

Node\* temp = \*head\_node;

if (temp != nullptr && temp->data == value) {

\*head\_node = temp->next;

if (\*head\_node != nullptr) {

(\*head\_node)->prev = nullptr;

}

delete temp;

return;

}

while (temp != nullptr && temp->data != value) {

temp = temp->next;

}

if (temp == nullptr) return;

if (temp->next != nullptr) {

temp->next->prev = temp->prev;

}

if (temp->prev != nullptr) {

temp->prev->next = temp->next;

}

delete temp;

}

void printList(Node\* node) {

while (node != nullptr) {

cout << node->data << " <-> ";

node = node->next;

}

}

int main() {

Node\* head = nullptr;

head = new Node();

head->data = 10;

head->next = new Node();

head->next->data = 20;

head->next->prev = head;

head->next->next = new Node();

head->next->next->data = 30;

head->next->next->prev = head->next;

head->next->next->next = nullptr;

cout << "Original list: \n";

printList(head);

cout << "\n";

deleteByValue(&head, 10);

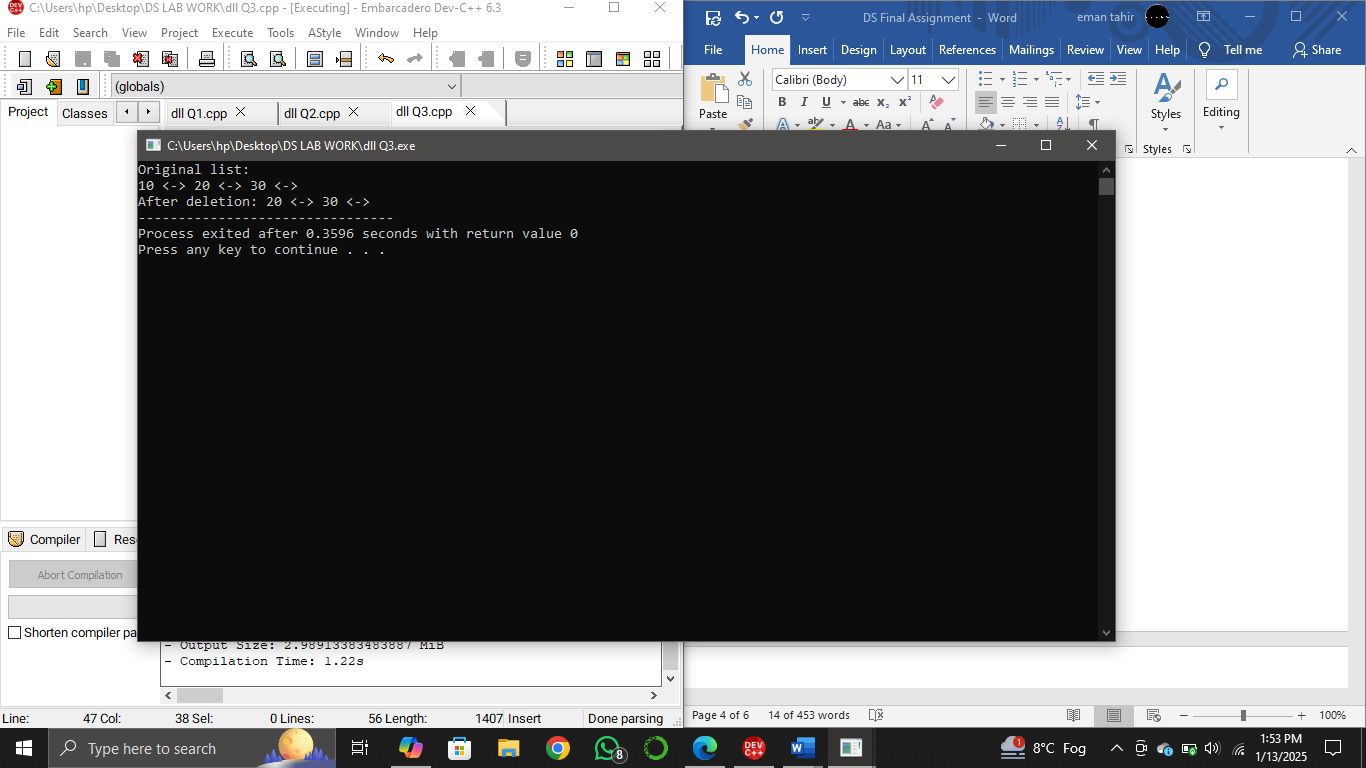
cout << "After deletion: ";

printList(head);

return 0;

}

**Output**:



• How would you delete a node at a specific position in a doubly linked list? Show it in code.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node\* prev;

};

void deleteNode(Node\*\* head\_node, int position) {

if (\*head\_node== nullptr) return;

Node\* temp = \*head\_node;

if (position == 0) {

\*head\_node = temp->next;

if (\*head\_node != nullptr) {

(\*head\_node)->prev = nullptr;

}

delete temp;

return;

}

for (int i = 0; temp != nullptr && i < position; i++) {

temp = temp->next;

}

if (temp == nullptr) return;

if (temp->next != nullptr) {

temp->next->prev = temp->prev;

}

if (temp->prev != nullptr) {

temp->prev->next = temp->next;

}

delete temp;

}

void printList(Node\* node) {

while (node != nullptr) {

cout << node->data << " <-> ";

node = node->next;

}

}

int main() {

Node\* head = nullptr;

head = new Node();

head->data = 10;

head->next = new Node();

head->next->data = 20;

head->next->prev = head;

head->next->next = new Node();

head->next->next->data = 30;

head->next->next->prev = head->next;

head->next->next->next = nullptr;

cout << "Original list: \n";

printList(head);

cout<<"\n";

deleteNode(&head, 0);

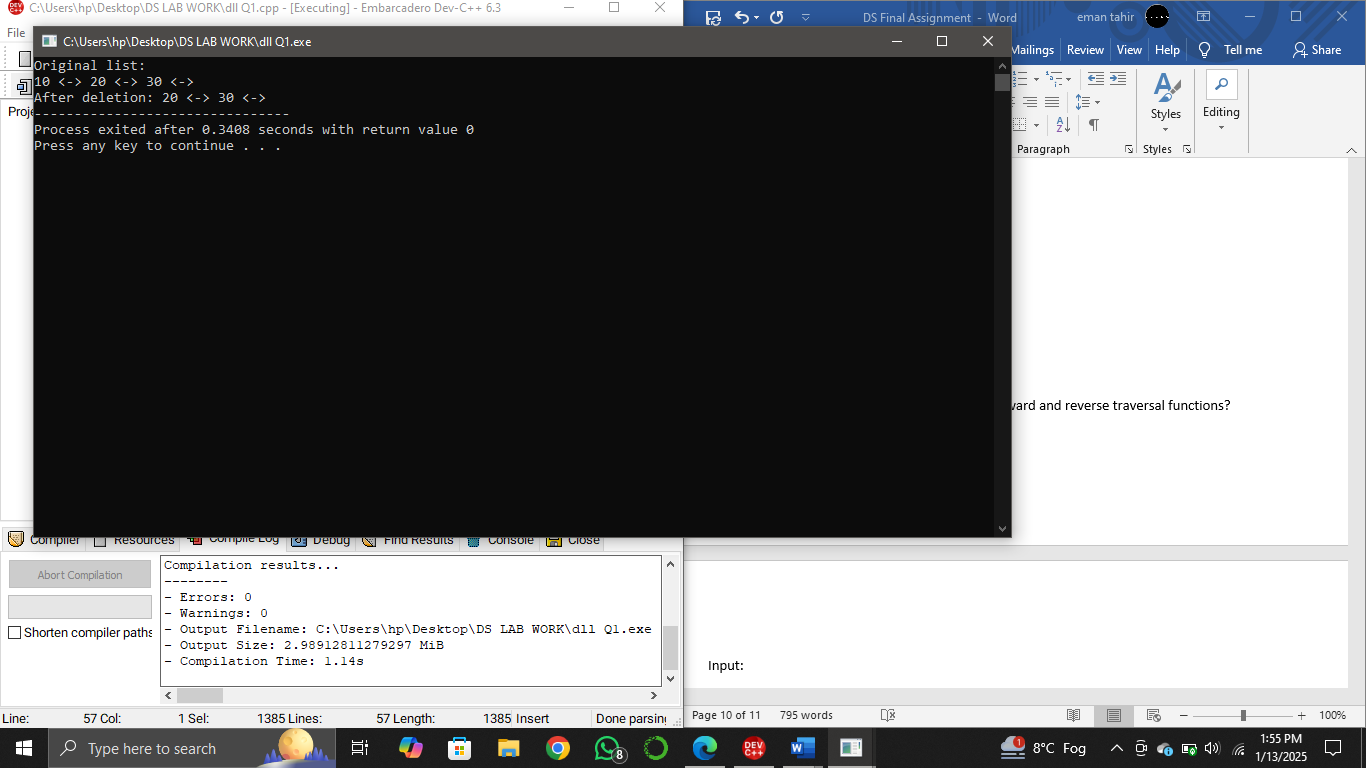
cout << "After deletion: ";

printList(head);

return 0;

}

**Output**:



• After deleting a node, how will you write the forward and reverse traversal functions?

**Solution**: In case of **forward traversal**, it starts from the head of the list to the end

forward\_traversal(head):

current = head

while current:

print(current.data, end=' ')

current = current.next

print()

whereas in case of **reverse traversal**, it starts from the last node and traverse backwards.

reverse\_traversal(tail):

current = tail

while current:

print(current.data, end=' ')

current = current.prev

print()

**Circular Linked List**

• Write a program to delete the first node in a circular linked list.

**Solution:**

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

void create(Node\*\* head\_node, int new\_data) {

Node\* new\_node = new Node();

new\_node->data = new\_data;

new\_node->next = \*head\_node;

if (\*head\_node == nullptr) {

\*head\_node = new\_node;

new\_node->next = new\_node;

} else {

Node\* temp = \*head\_node;

while (temp->next != \*head\_node) {

temp = temp->next;

}

temp->next = new\_node;

}

}

void deleteByValue(Node\*\* head\_node, int value) {

if (\*head\_node == nullptr) return;

Node\* current = \*head\_node;

Node\* previous = nullptr;

while (current->data != value) {

if (current->next == \*head\_node) {

cout <<" not found.\n";

return;

}

previous = current;

current = current->next;

}

if (current->next == \*head\_node && previous == nullptr) {

\*head\_node = nullptr;

delete current;

} else if (current == \*head\_node) {

previous = \*head\_node;

while (previous->next != \*head\_node) {

previous = previous->next;

}

previous->next = current->next;

\*head\_node = current->next;

delete current;

} else if (current->next == \*head\_node) {

previous->next = \*head\_node;

delete current;

} else {

previous->next = current->next;

delete current;

}

}

void display(Node\* head) {

if (head == nullptr) return;

Node\* temp = head;

do {

cout << temp->data << " -> ";

temp = temp->next;

} while (temp != head);

}

int main() {

Node\* head = nullptr;

create(&head, 10);

create(&head, 20);

create(&head, 30);

cout << "Original list: ";

display(head);

deleteByValue(&head, 10);

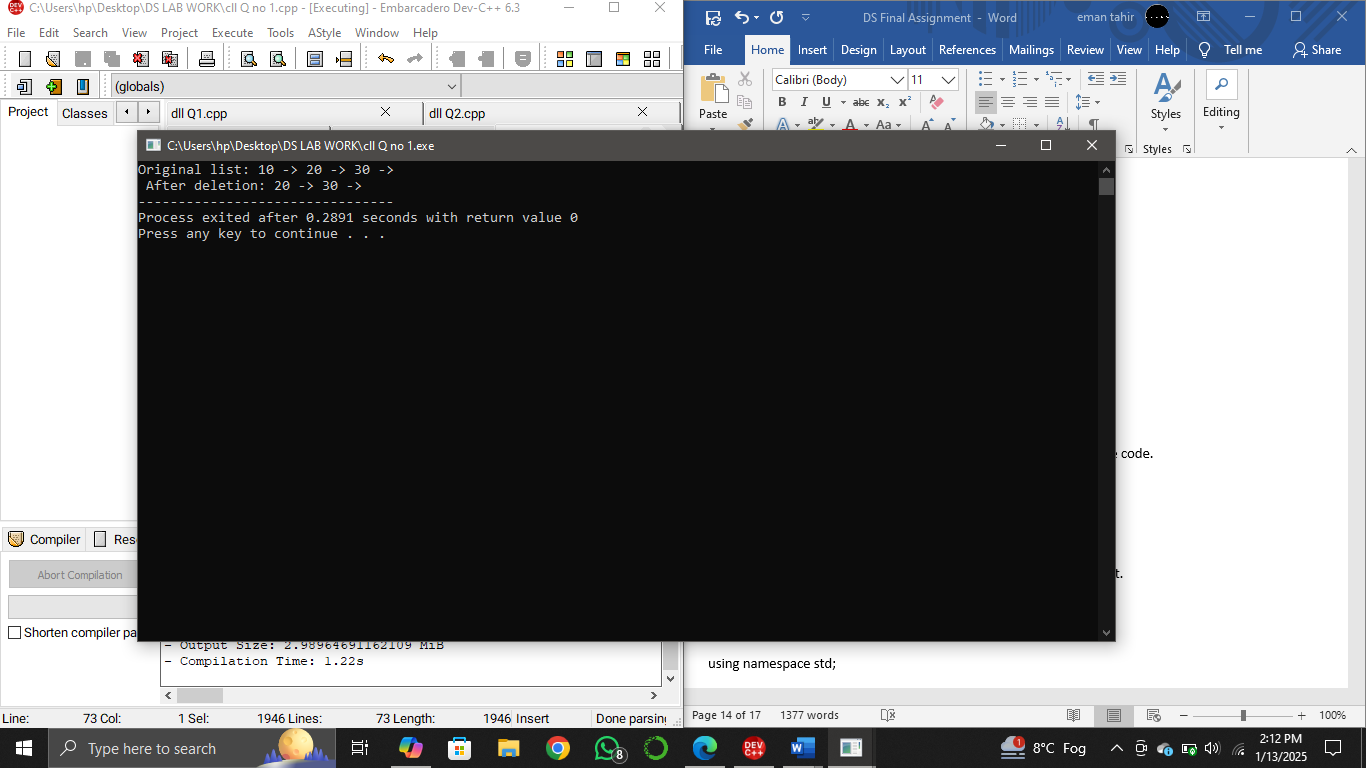
cout << "\n After deletion: ";

display(head);

return 0;

}

**Output**:



• How can you delete the last node in a circular linked list? Write the code.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

void create(Node\*\* head\_node, int new\_data) {

Node\* new\_node = new Node();

new\_node->data = new\_data;

new\_node->next = \*head\_node;

if (\*head\_node == nullptr) {

\*head\_node = new\_node;

new\_node->next = new\_node;

} else {

Node\* temp = \*head\_node;

while (temp->next != \*head\_node) {

temp = temp->next;

}

temp->next = new\_node;

}

}

void deleteByValue(Node\*\* head\_node, int value) {

if (\*head\_node == nullptr) return;

Node\* current = \*head\_node;

Node\* previous = nullptr;

while (current->data != value) {

if (current->next == \*head\_node) {

cout <<" not found.\n";

return;

}

previous = current;

current = current->next;

}

if (current->next == \*head\_node && previous == nullptr) {

\*head\_node = nullptr;

delete current;

} else if (current == \*head\_node) {

previous = \*head\_node;

while (previous->next != \*head\_node) {

previous = previous->next;

}

previous->next = current->next;

\*head\_node = current->next;

delete current;

} else if (current->next == \*head\_node) {

previous->next = \*head\_node;

delete current;

} else {

previous->next = current->next;

delete current;

}

}

void display(Node\* head) {

if (head == nullptr) return;

Node\* temp = head;

do {

cout << temp->data << " -> ";

temp = temp->next;

} while (temp != head);

}

int main() {

Node\* head = nullptr;

create(&head, 10);

create(&head, 20);

create(&head, 30);

cout << "Original list: ";

display(head);

deleteByValue(&head, 30);

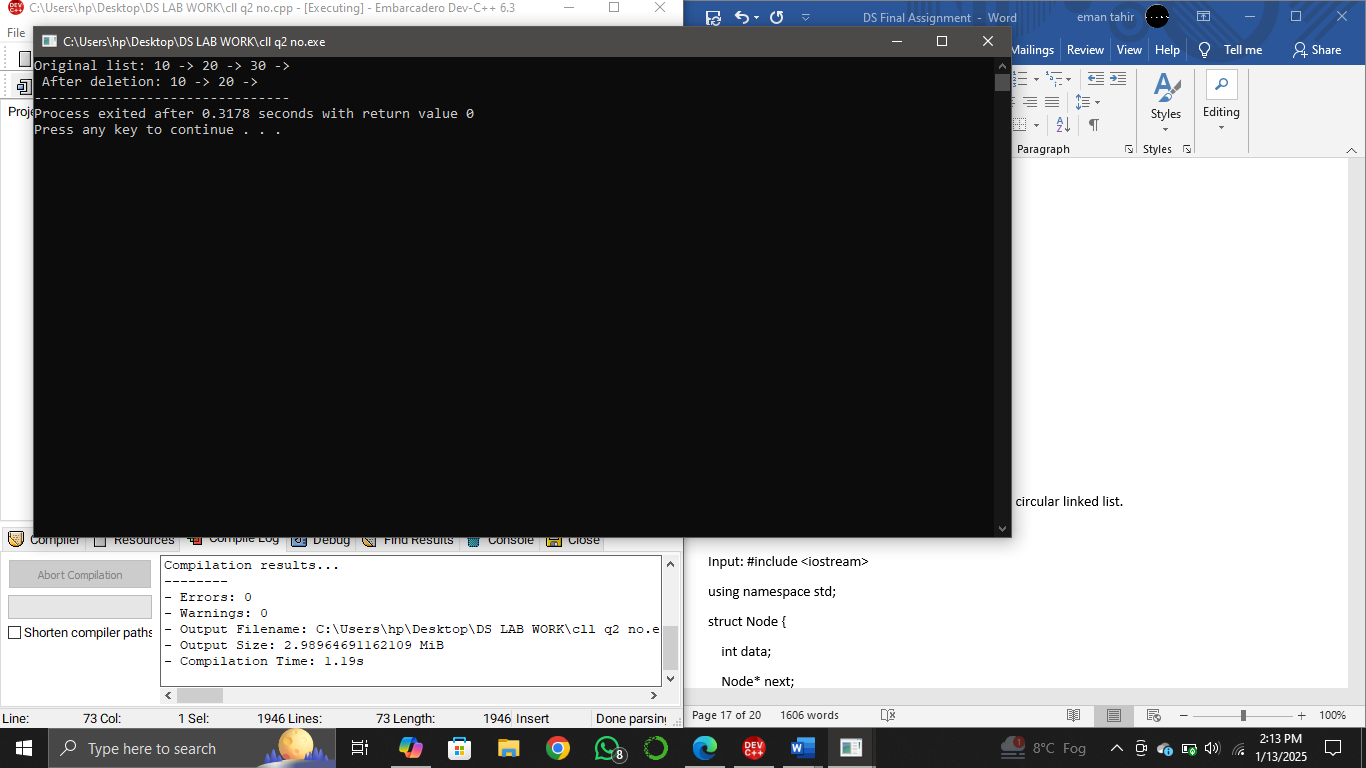
cout << "\n After deletion: ";

display(head);

return 0;

}

**Output**:



• Write a function to delete a node by its value in a circular linked list.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

void create(Node\*\* head\_node, int new\_data) {

Node\* new\_node = new Node();

new\_node->data = new\_data;

new\_node->next = \*head\_node;

if (\*head\_node == nullptr) {

\*head\_node = new\_node;

new\_node->next = new\_node;

} else {

Node\* temp = \*head\_node;

while (temp->next != \*head\_node) {

temp = temp->next;

}

temp->next = new\_node;

}

}

void deleteByValue(Node\*\* head\_node, int value) {

if (\*head\_node == nullptr) return;

Node\* current = \*head\_node;

Node\* previous = nullptr;

while (current->data != value) {

if (current->next == \*head\_node) {

cout <<" not found.\n";

return;

}

previous = current;

current = current->next;

}

if (current->next == \*head\_node && previous == nullptr) {

\*head\_node = nullptr;

delete current;

} else if (current == \*head\_node) {

previous = \*head\_node;

while (previous->next != \*head\_node) {

previous = previous->next;

}

previous->next = current->next;

\*head\_node = current->next;

delete current;

} else if (current->next == \*head\_node) {

previous->next = \*head\_node;

delete current;

} else {

previous->next = current->next;

delete current;

}

}

void display(Node\* head) {

if (head == nullptr) return;

Node\* temp = head;

do {

cout << temp->data << " -> ";

temp = temp->next;

} while (temp != head);

}

int main() {

Node\* head = nullptr;

create(&head, 10);

create(&head, 20);

create(&head, 30);

cout << "Original list: ";

display(head);

deleteByValue(&head, 10);

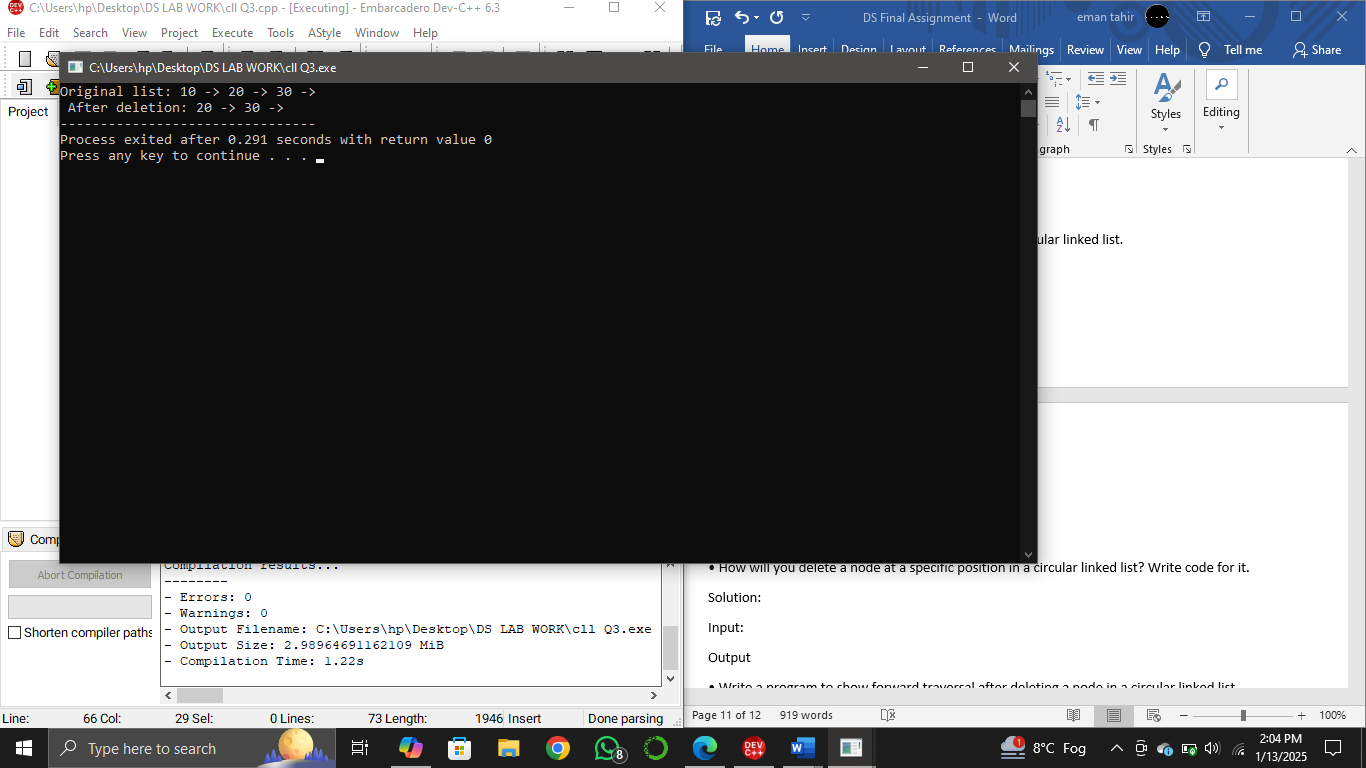
cout << "\n After deletion: ";

display(head);

return 0;

}

**Output**:



• How will you delete a node at a specific position in a circular linked list? Write code for it.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

void create(Node\*\* head\_node, int new\_data) {

Node\* new\_node = new Node();

new\_node->data = new\_data;

new\_node->next = \*head\_node;

if (\*head\_node == nullptr) {

\*head\_node = new\_node;

new\_node->next = new\_node;

} else {

Node\* temp = \*head\_node;

while (temp->next != \*head\_node) {

temp = temp->next;

}

temp->next = new\_node;

}

}

void deleteAtPosition(Node\*\* head\_node, int position) {

if (\*head\_node == nullptr || position < 0) return;

Node\* current = \*head\_node;

if (position == 0) {

Node\* last = \*head\_node;

while (last->next != \*head\_node) {

last = last->next;

}

last->next = current->next;

\*head\_node = current->next;

delete current;

return;

}

for (int i = 0; current != nullptr && i < position - 1; i++) {

current = current->next;

}

if (current == nullptr || current->next == nullptr) return;

Node\* temp = current->next;

current->next = temp->next;

delete temp;

}

void display(Node\* head) {

if (head == nullptr) return;

Node\* temp = head;

do {

cout << temp->data << " -> ";

temp = temp->next;

} while (temp != head);

}

int main() {

Node\* head = nullptr;

create(&head, 10);

create(&head, 20);

create(&head, 30);

cout << "Original list: ";

display(head);

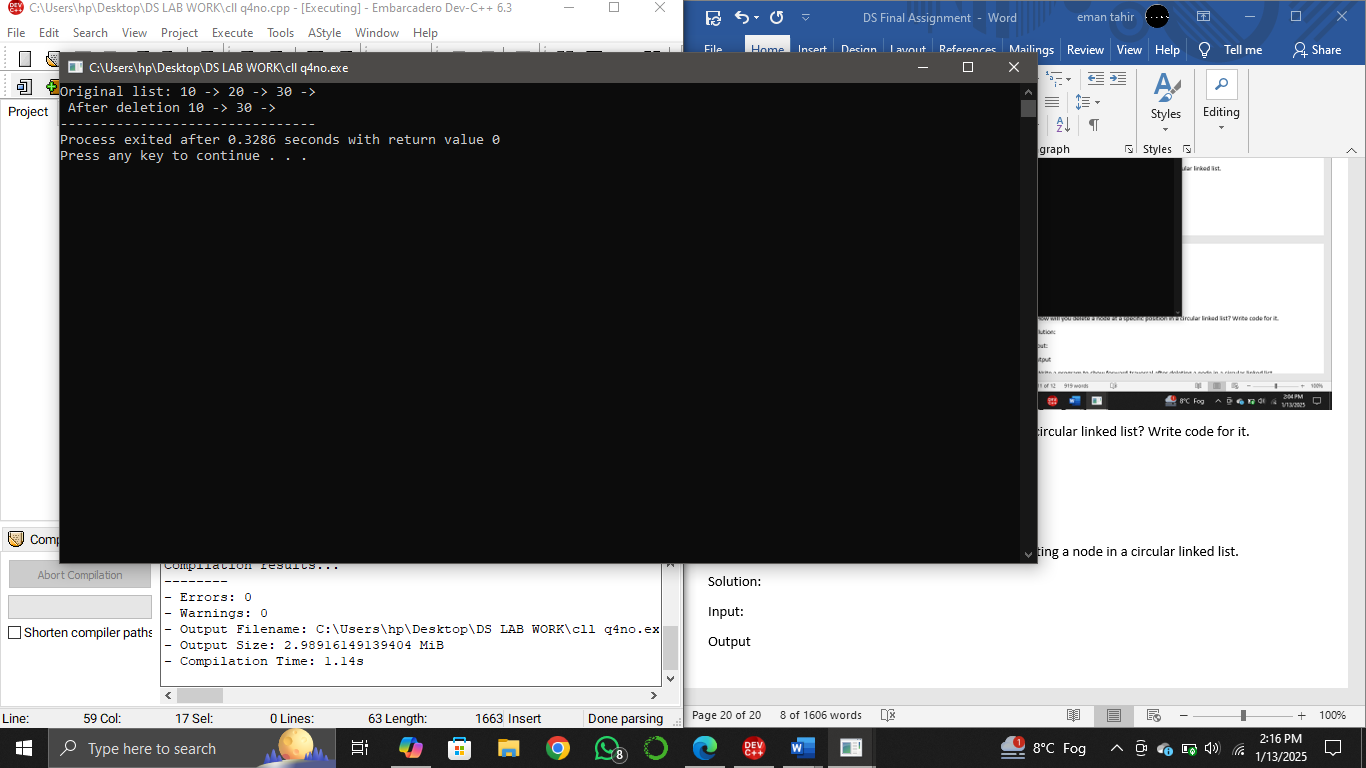
deleteAtPosition(&head, 1);

cout << "\n After deletion ";

display(head);

return 0;}

**Output**:



• Write a program to show forward traversal after deleting a node in a circular linked list.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

void create(Node\*\* head\_node, int new\_data) {

Node\* new\_node = new Node();

new\_node->data = new\_data;

new\_node->next = \*head\_node;

if (\*head\_node == nullptr) {

\*head\_node = new\_node;

new\_node->next = new\_node;

} else {

Node\* temp = \*head\_node;

while (temp->next != \*head\_node) {

temp = temp->next;

}

temp->next = new\_node;

}

}

void deleteByValue(Node\*\* head\_node, int value) {

if (\*head\_node == nullptr) return;

Node\* current = \*head\_node;

Node\* previous = nullptr;

while (current->data != value) {

if (current->next == \*head\_node) {

cout << "Node with value " << value << " not found.\n";

return;

}

previous = current;

current = current->next;

}

if (current->next == \*head\_node && previous == nullptr) {

\*head\_node = nullptr;

delete current;

} else if (current == \*head\_node) {

previous = \*head\_node;

while (previous->next != \*head\_node) {

previous = previous->next;

}

previous->next = current->next;

\*head\_node = current->next;

delete current;

} else if (current->next == \*head\_node) {

previous->next = \*head\_node;

delete current;

} else {

previous->next = current->next;

delete current;

}

}

void forwardTraversal(Node\* head) {

if (head == nullptr) return;

Node\* temp = head;

do {

cout << temp->data << " -> ";

temp = temp->next;

} while (temp != head);

}

int main() {

Node\* head = nullptr;

create(&head, 10);

create(&head, 20);

create(&head, 30);

cout << "Original list: ";

forwardTraversal(head);

deleteByValue(&head, 10);

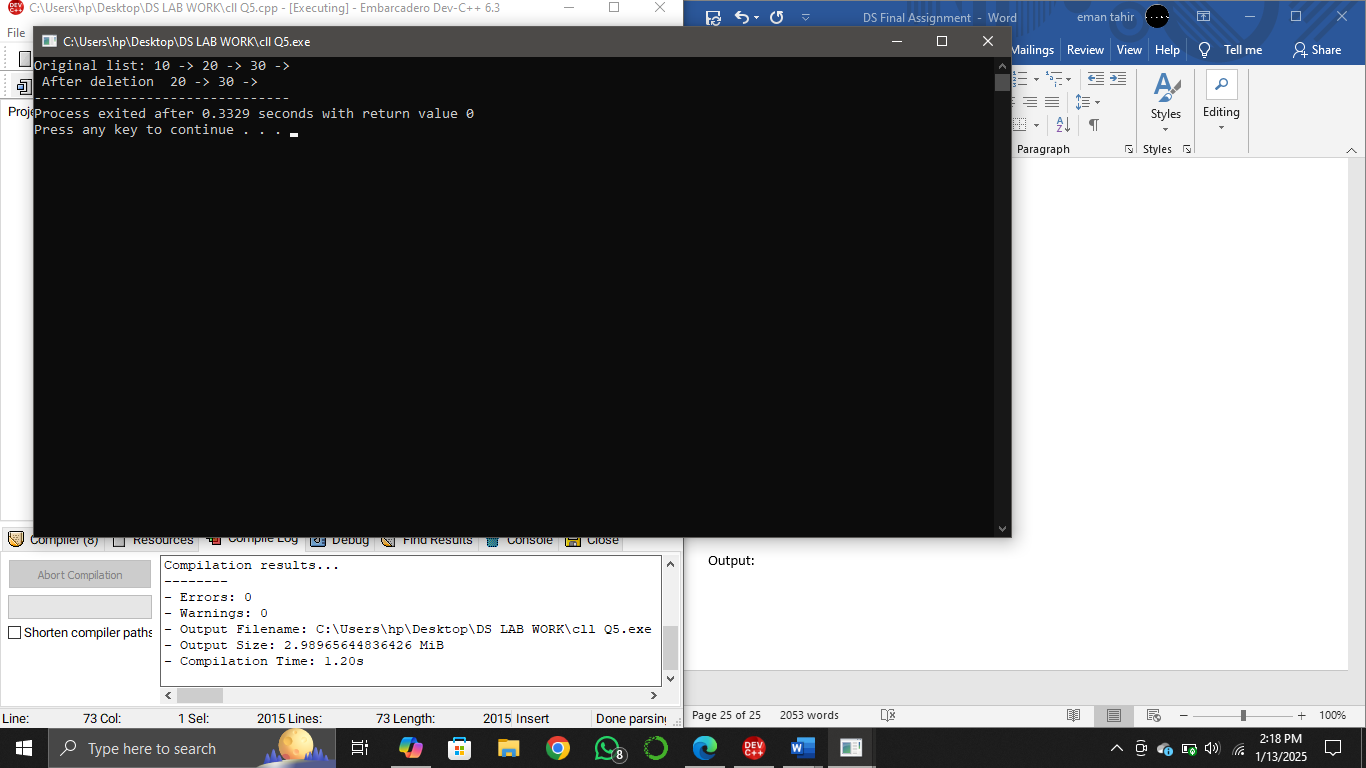
cout << "\n After deletion ";

forwardTraversal(head);

return 0;

}

**Output**:



**Binary Search Tree**

• Write a program to count all the nodes in a binary search tree.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

Node\* insert(Node\* root, int value) {

if (root == nullptr) {

return new Node(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

} else {

root->right = insert(root->right, value);

}

return root;

}

int countNodes(Node\* root) {

if (root == nullptr) {

return 0;

}

return 1 + countNodes(root->left) + countNodes(root->right);

}

int main() {

Node\* root = nullptr;

root = insert(root, 9);

insert(root, 7);

insert(root, 5);

insert(root, 3);

insert(root, 6);

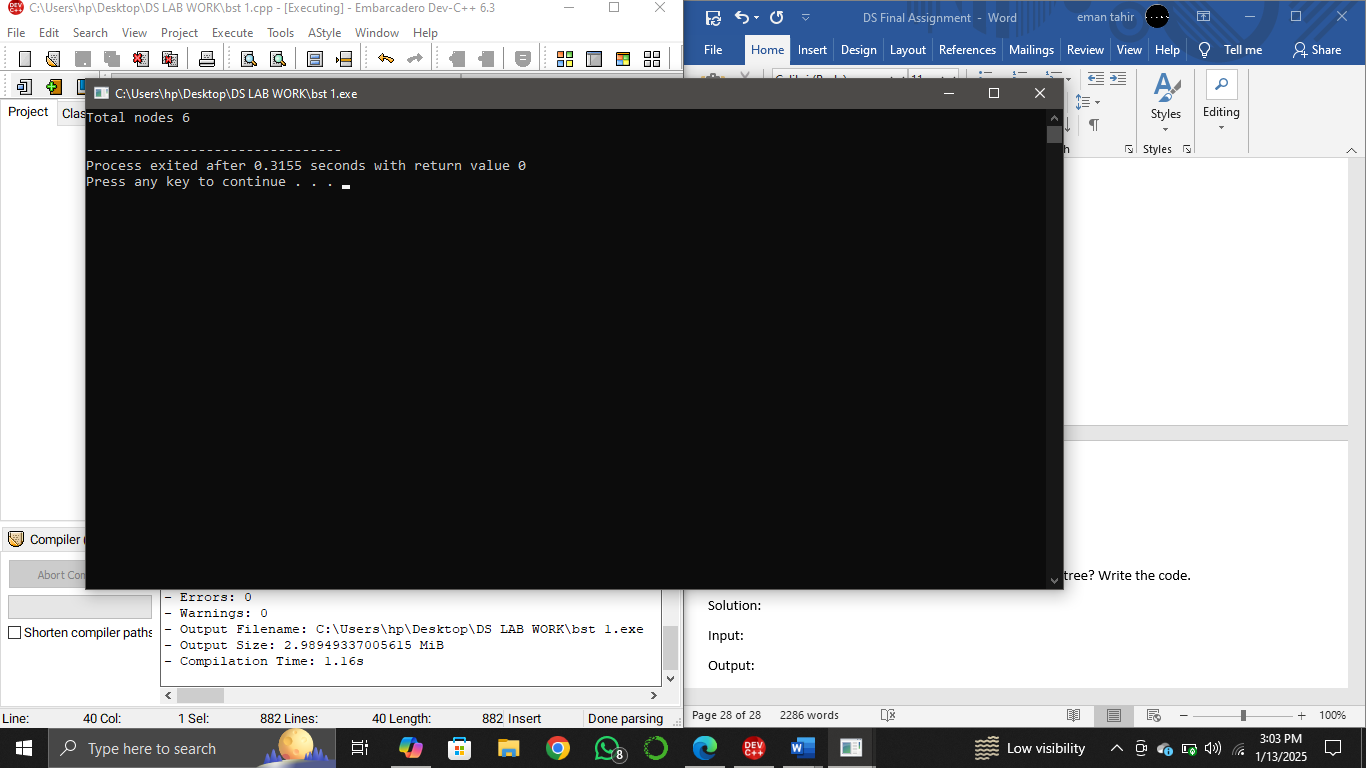
insert(root, 8);

cout<<"Total nodes " << countNodes(root)<<endl;

return 0;

}

**Output**:



• How can you search for a specific value in a binary search tree? Write the code.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

class BST {

public:

TreeNode\* insert(TreeNode\* root, int val) {

if (root == NULL) {

return new TreeNode(val);

}

if (val < root->val) {

root->left = insert(root->left, val);

} else if (val > root->val) {

root->right = insert(root->right, val);

}

return root;

}

bool search(TreeNode\* root, int key) {

if (root == NULL) {

return false;

}

if (root->val == key) {

return true;

}

if (key < root->val) {

return search(root->left, key);

} else {

return search(root->right, key);

}

}

};

void inOrderTraversal(TreeNode\* root) {

if (root != NULL) {

inOrderTraversal(root->left);

std::cout << root->val << " ";

inOrderTraversal(root->right);

}

}

int main() {

BST bst;

TreeNode\* root = NULL;

root = bst.insert(root, 5);

root = bst.insert(root, 3);

root = bst.insert(root, 8);

root = bst.insert(root, 1);

root = bst.insert(root, 4);

root = bst.insert(root, 6);

int keyToSearch = 4;

if (bst.search(root, keyToSearch)) {

cout << "Value " << keyToSearch << " found ." << endl;

} else {

cout << "Value " << keyToSearch << " not found " <<endl;

}

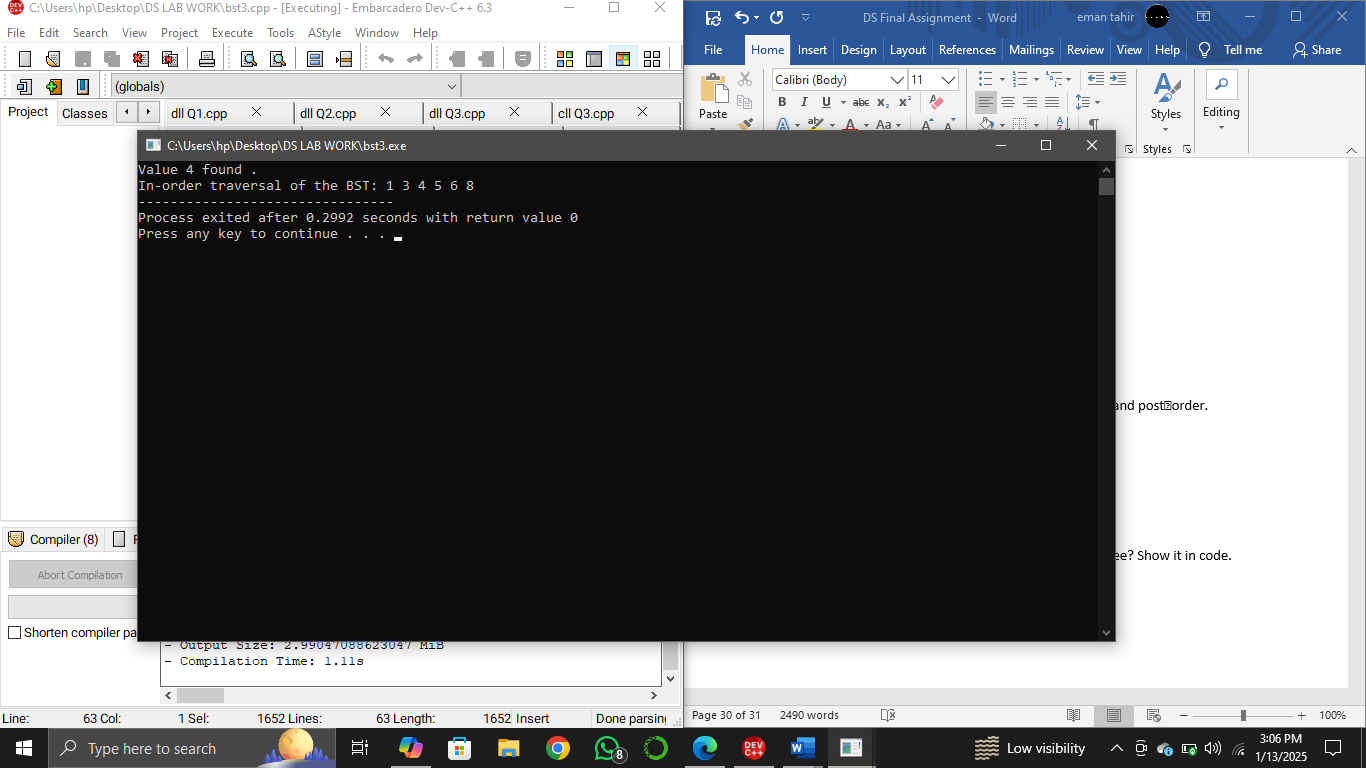
cout << "In-order traversal of the BST: ";

inOrderTraversal(root);

return 0;

}

**Output**:



• Write code to traverse a binary search tree in in-order, pre-order, and postorder.

**Solution**:

**In-order**:

**Input**: #include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

void inorder(Node\* root) {

if (root != nullptr) {

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

}

int main() {

Node\* root = nullptr;

root = new Node(2);

root->left = new Node(13);

root->right = new Node(40);

root->left->left = new Node(11);

root->left->right = new Node(14);

root->right->left = new Node(39);

root->right->right = new Node(41);

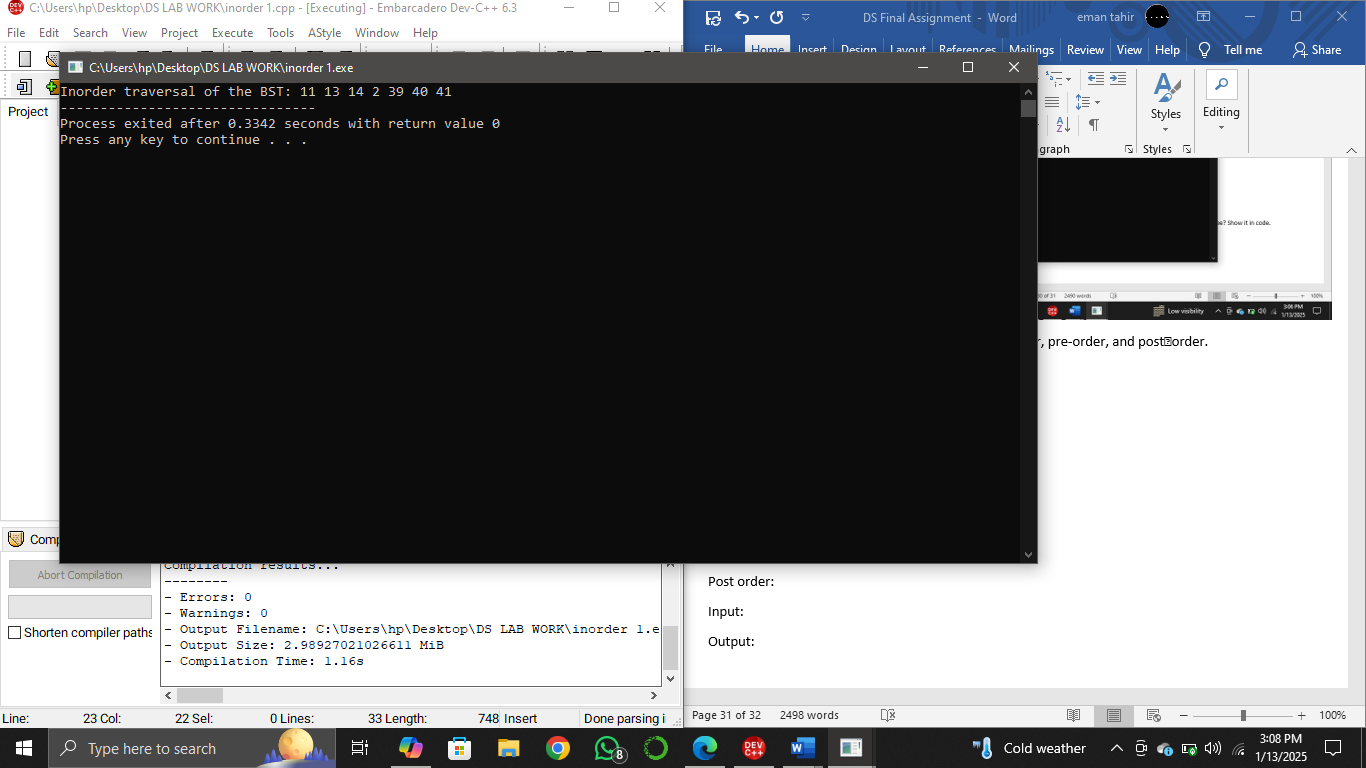
cout << "Inorder traversal of the BST: ";

inorder(root);

return 0;

}

**Output:**



**Pre-order:**

**Input**: #include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

void preorder(Node\* root) {

if (root != nullptr) {

cout << root->data << " ";

preorder(root->left);

preorder(root->right);

}

}

int main() {

Node\* root = nullptr;

root = new Node(17);

root->left = new Node(13);

root->right = new Node(40);

root->left->left = new Node(11);

root->left->right = new Node(14);

root->right->left = new Node(39);

root->right->right = new Node(41);

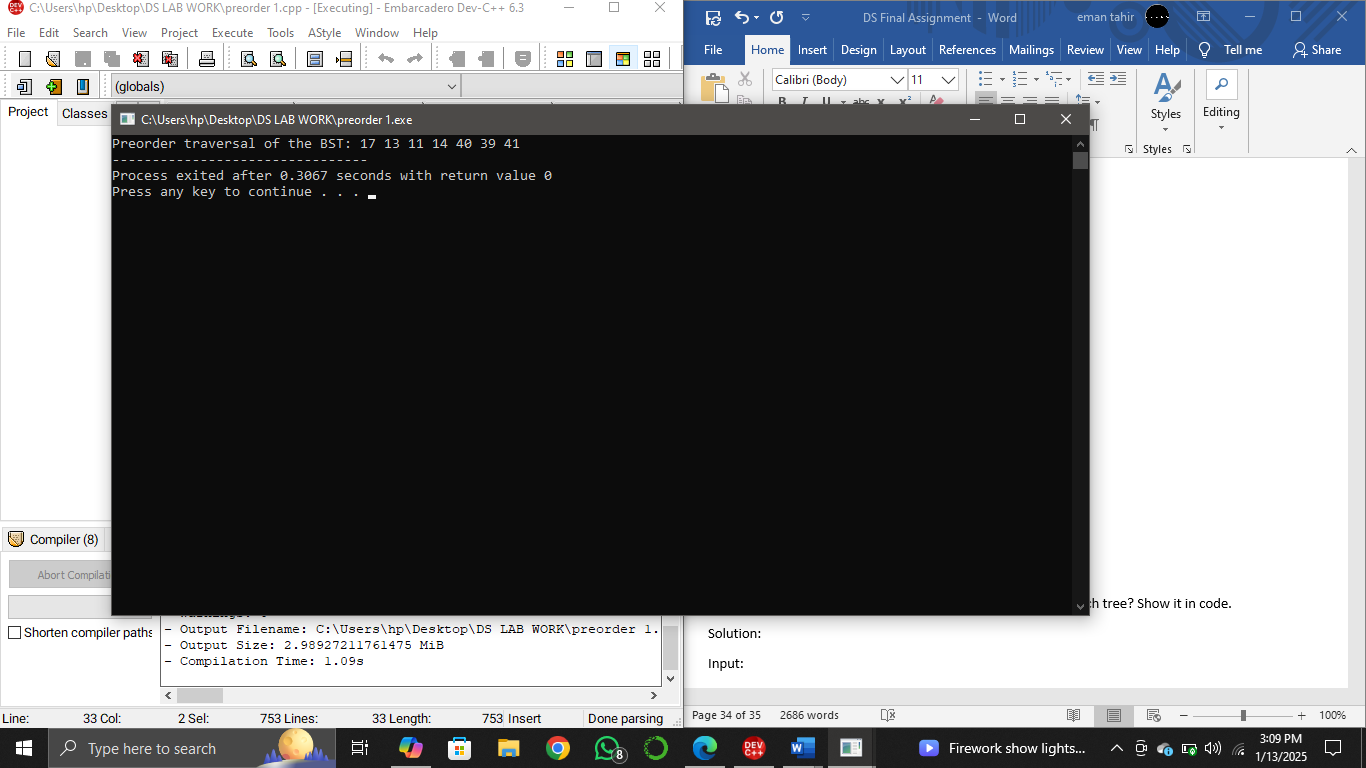
cout << "Preorder traversal of the BST: ";

preorder(root);

return 0;

}

**Output**:



**Post-order:**

**Input**: #include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

void postorder(Node\* root) {

if (root != nullptr) {

postorder(root->left);

postorder(root->right);

cout << root->data << " ";

}

}

int main() {

Node\* root = nullptr;

root = new Node(17);

root->left = new Node(13);

root->right = new Node(40);

root->left->left = new Node(11);

root->left->right = new Node(14);

root->right->left = new Node(39);

root->right->right = new Node(41);

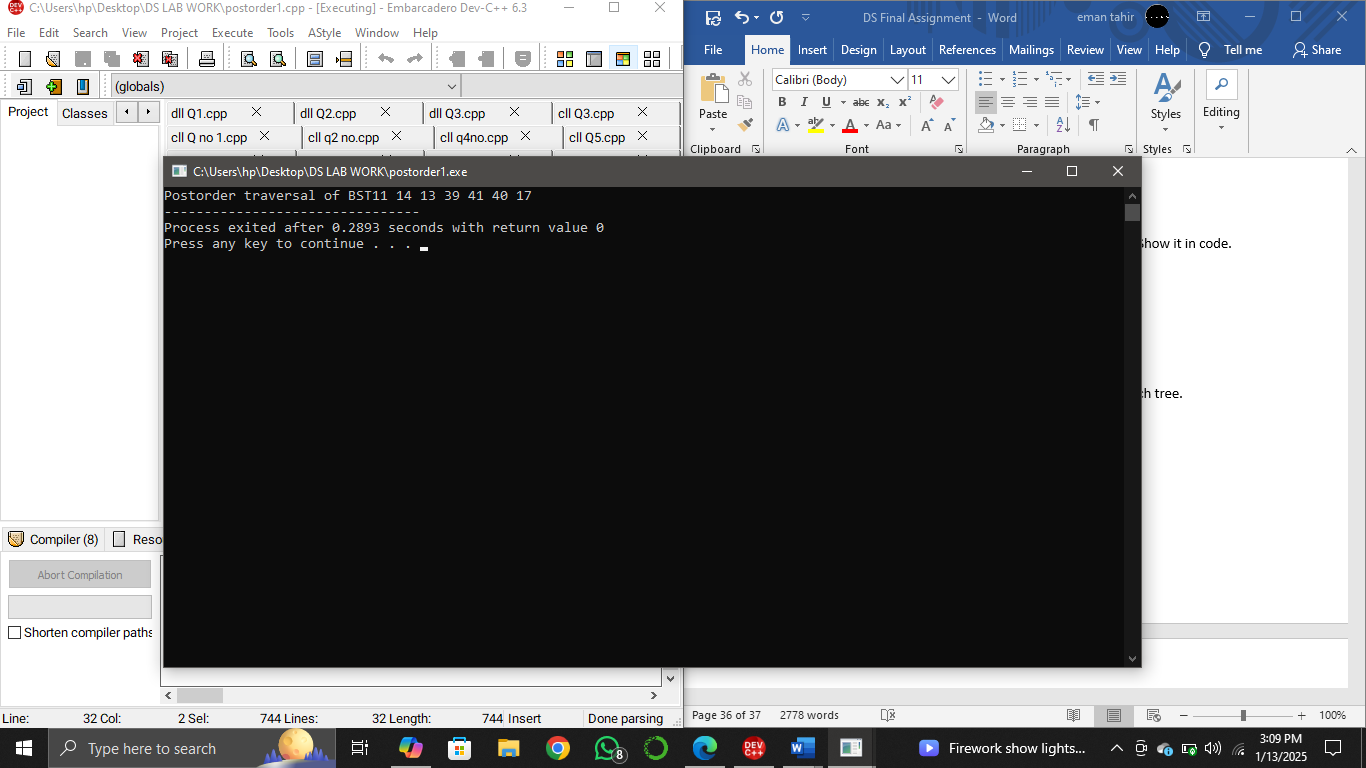
cout << "Postorder traversal of BST";

postorder(root);

return 0;

}

**Output**:



• How will you write reverse in-order traversal for a binary search tree? Show it in code.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

Node\* insert(Node\* root, int value) {

if (root == nullptr) {

return new Node(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

} else {

root->right = insert(root->right, value);

}

return root;

}

void reverseTraversal(Node\* root) {

if (root == nullptr) return;

reverseTraversal(root->right);

cout << root->data << “ “;

reverseTraversal(root->left);

}

int main() {

Node\* root = nullptr;

root = insert(root, 2);

insert(root, 13);

insert(root, 40);

insert(root, 11);

insert(root, 14);

insert(root, 39);

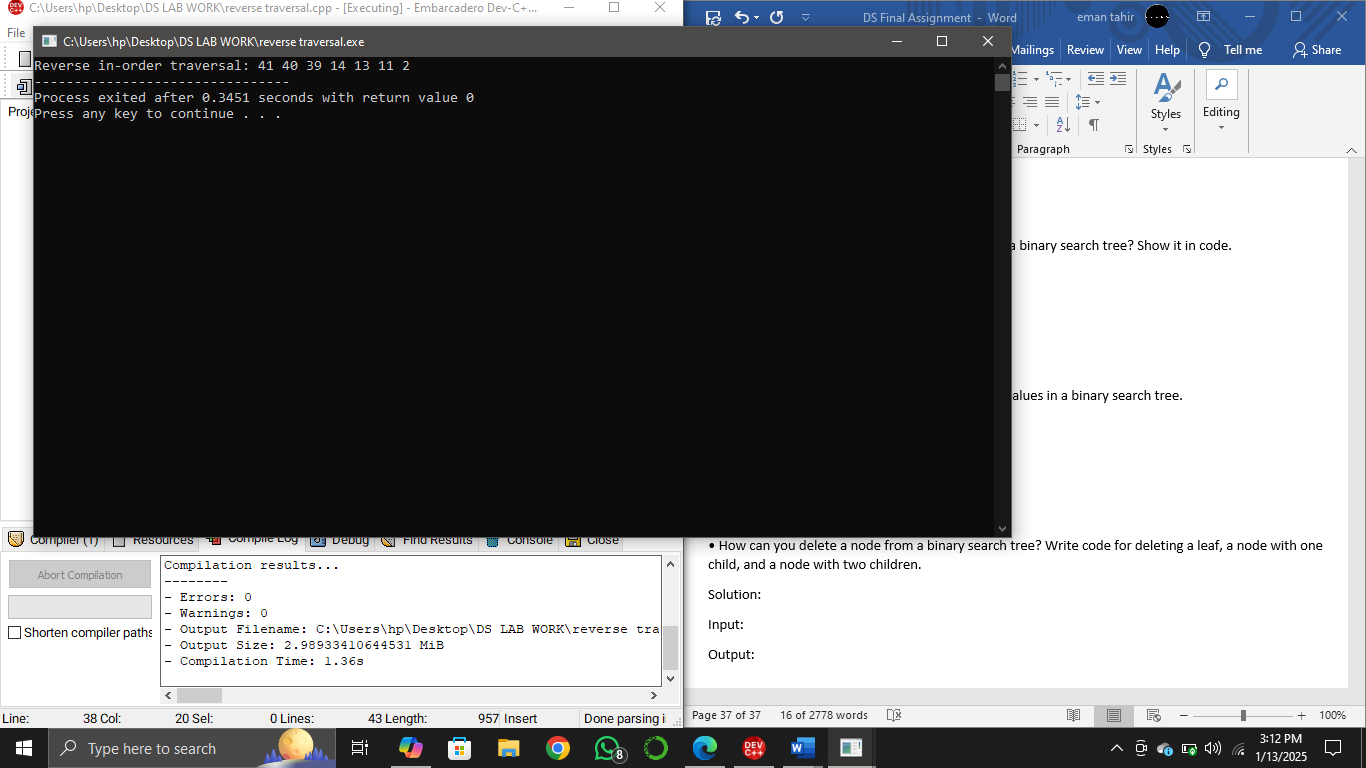
insert(root, 41);

cout << “Reverse in-order traversal: “;

reverseTraversal(root);

return 0;}

**Output**:



• Write a program to check if there are duplicate values in a binary search tree.

**Solution**:

**Input**: #include <iostream>

using namespace std;

class Node {

public:

int data;

int count;

Node\* left;

Node\* right;

Node(int value) {

data = value;

count = 1;

left = right = nullptr;

}

};

Node\* insert(Node\* root, int key) {

if (root == nullptr) {

return new Node(key);

}

if (key == root->data) {

root->count++;

return root;

}

if (key < root->data) {

root->left = insert(root->left, key);

} else if (key > root->data) {

root->right = insert(root->right, key);

}

return root;

}

void inorder(Node\* root) {

if (root != nullptr) {

inorder(root->left);

cout << root->data << “(“ << root->count << “) “;

inorder(root->right);

}

}

int main() {

Node\* root = nullptr;

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

insert(root, 30);

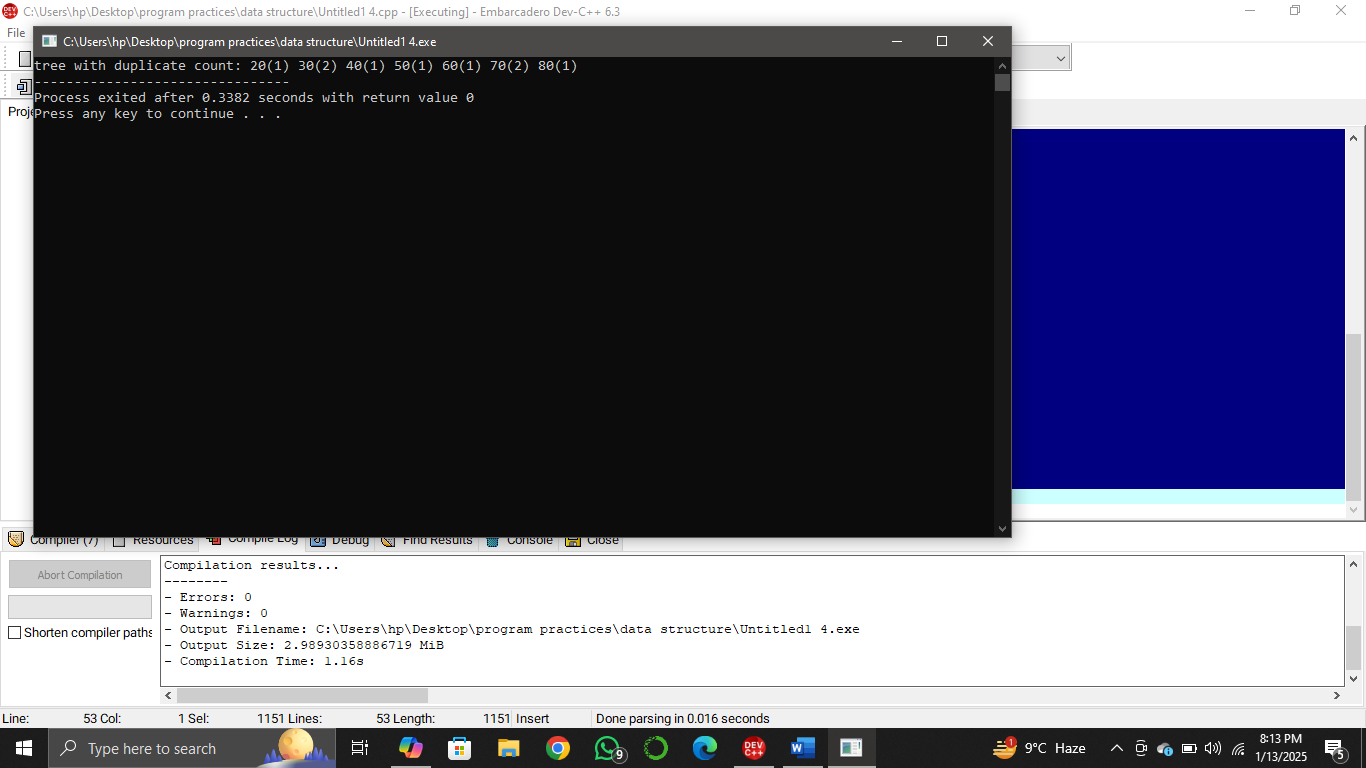
insert(root, 70);

cout << “tree with duplicate count: “;

inorder(root);

return 0;}

**Output**:



• How can you delete a node from a binary search tree? Write code for deleting a leaf, a node with one child, and a node with two children.

**Solution**:

**Input**: #include <iostream>

using namespace std;

struct TreeNode {

int value;

TreeNode\* left;

TreeNode\* right;

TreeNode(int val) : value(val), left(nullptr), right(nullptr) {}

};

TreeNode\* findMin(TreeNode\* node) {

while (node->left != nullptr) node = node->left;

return node;

}

TreeNode\* deleteNode(TreeNode\* root, int key) {

if (root == nullptr) return root;

if (key < root->value) {

root->left = deleteNode(root->left, key);

} else if (key > root->value) {

root->right = deleteNode(root->right, key);

} else {

// Node with only one child

if (root->left == nullptr) {

TreeNode\* temp = root->right;

delete root;

return temp;

} else if (root->right == nullptr) {

TreeNode\* temp = root->left;

delete root;

return temp;

}

// Node with two children

TreeNode\* temp = findMin(root->right);

root->value = temp->value;

root->right = deleteNode(root->right, temp->value);

}

return root;

}

void inOrderTraversal(TreeNode\* root) {

if (root != nullptr) {

inOrderTraversal(root->left);

std::cout << root->value << “ “;

inOrderTraversal(root->right);

}

}

int main() {

TreeNode\* root = new TreeNode(10);

root->left = new TreeNode(5);

root->right = new TreeNode(14);

root->left->left = new TreeNode(4);

root->left->right = new TreeNode(7);

root->right->left = new TreeNode(11);

root->right->right = new TreeNode(17);

std::cout << “original list “;

inOrderTraversal(root);

cout<<endl;

int keyToDelete;

// Delete a leaf node

keyToDelete = 7;

root = deleteNode(root, keyToDelete);

cout << “\nafter leaf deletion “ << keyToDelete << “: “;

inOrderTraversal(root);

cout<<endl;

// Delete a node with one child

keyToDelete = 5;

root = deleteNode(root, keyToDelete);

cout << “\nafter deleting node with one child “ << keyToDelete << “: “;

inOrderTraversal(root);

cout <<endl;

// Delete a node with two children

keyToDelete = 11;

root = deleteNode(root, keyToDelete);

cout << “\nafter deleting node with two children “ << keyToDelete << “: “;

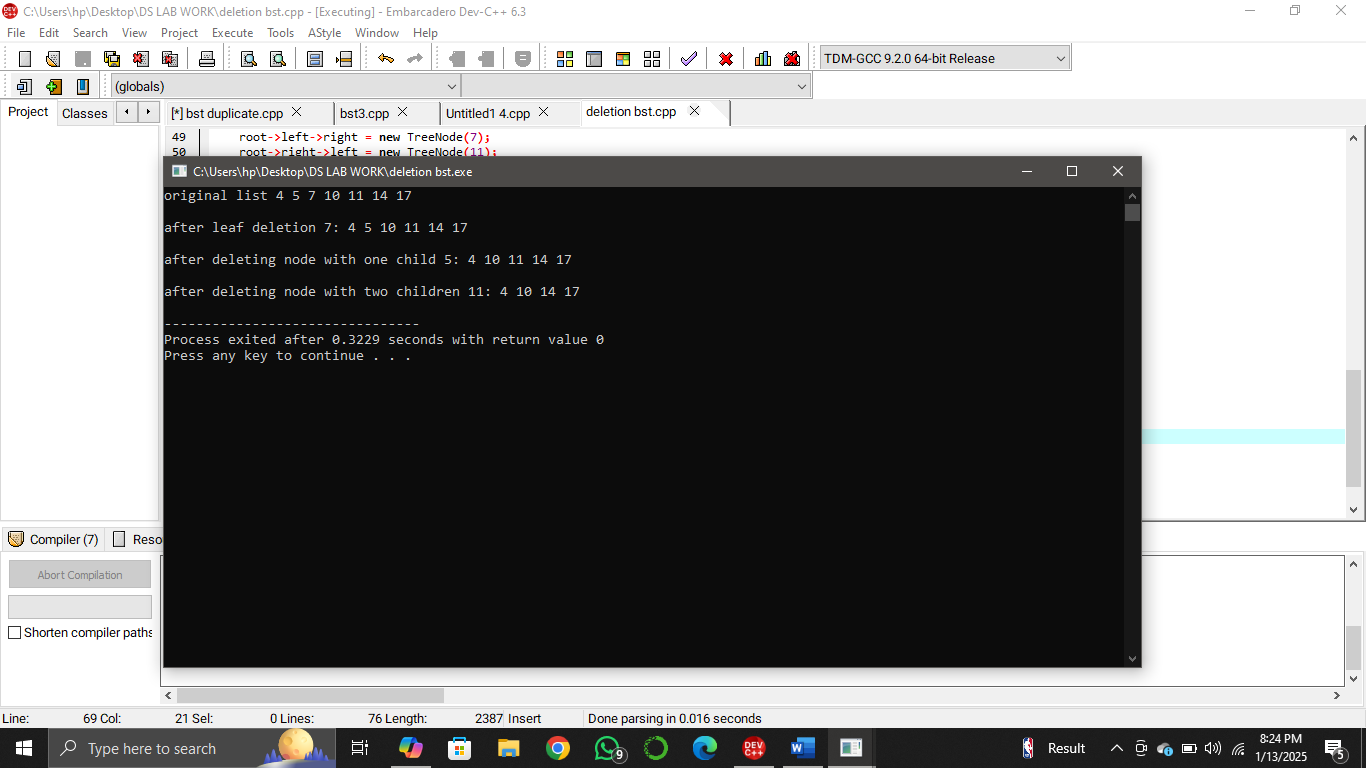
inOrderTraversal(root);

cout<<endl;

return 0;

}

**Output:**



**--------**