

**Final Assignment**

**Submitted by:** Yumna Irfan

**Registration no:** 2023-bs-ai-021

**Submitted to:** Miss Irsha Qureshi

**Department:** BS Artificial Intelligence

**Doubly Linked List**

**Question 1:** **Write a program to delete the first node in a doubly linked list.**

#include <iostream>

using namespace std;

struct Node

{

int val;

Node\* next;

Node\* prev;

Node(int data)

{

val = data;

next = nullptr;

prev = nullptr;

}

};

//function to delete first node

void deleteFirstNode(Node\*& head)

{

if (head == nullptr)

{

cout << "List is empty." << endl;

return;

}

Node\* temp = head; //saving current head

head = head->next; //moving head to next node

if (head != nullptr)

{

head->prev = nullptr; //setting new head prev pointer to null

}

delete temp;

cout << "First node deleted." << endl;

}

//function to print doubly linked list

void printList(Node\* head)

{

Node\* temp = head;

while (temp != nullptr) {

cout << temp->val << " ";

temp = temp->next;

}

cout << endl;

}

int main()

{

//making doubly linklist

Node\* head = new Node(15);

Node\* second = new Node(35);

Node\* third = new Node(50);

//Linking

head->next = second;

second->prev = head;

second->next = third;

third->prev = second;

//printing original list

cout << "Original List: ";

printList(head);

//deletion

deleteFirstNode(head);

//printing list after deletion

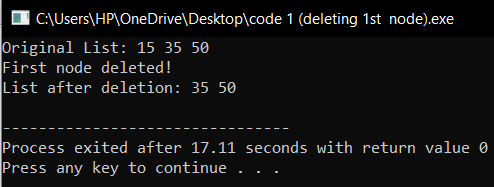
cout << "List after deletion: ";

printList(head);

return 0;

}

**Output:**



**Question 2: How can you delete the last node in a doubly linked list? Write the code.**

#include <iostream>

using namespace std;

struct Node {

int val;

Node\* next;

Node\* prev;

Node(int data) {

val = data;

next = nullptr;

prev = nullptr;

}

};

//function to delete last node

void deleteLastNode(Node\*& head) {

if (head == nullptr) {

cout << "List is empty" << endl;

return;

}

if (head->next == nullptr) { //if there is only one node

delete head;

head = nullptr;

cout << "Last node deleted! list is empty" << endl;

return;

}

Node\* temp = head;

//traverse upto last node

while (temp->next != nullptr) {

temp = temp->next;

}

//setting second-to-last node's next pointer to null

temp->prev->next = nullptr;

delete temp; //delete last node

cout << "Last node deleted!" << endl;

}

//function to print doubly linklist

void printList(Node\* head) {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->val << " ";

temp = temp->next;

}

cout << endl;

}

int main() {

//making doubly linklist

Node\* head = new Node(15);

Node\* second = new Node(35);

Node\* third = new Node(50);

//linking

head->next = second;

second->prev = head;

second->next = third;

third->prev = second;

//printing original list

cout << "Original List: ";

printList(head);

//deleting last node

deleteLastNode(head);

//list after deletion

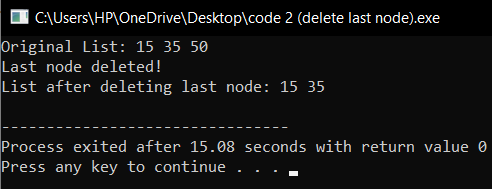
cout << "List after deleting last node: ";

printList(head);

return 0;

}

**Output:**



**Question 3: Write code to delete a node by its value in a doubly linked list.**

#include <iostream>

using namespace std;

struct Node

{

int val;

Node\* next;

Node\* prev;

Node(int data)

{

val = data;

next = nullptr;

prev = nullptr;

}

};

//function to delete node by its value

void deleteNodeByValue(Node\*& head, int value)

{

if (head == nullptr)

{

cout << "List is empty " << endl;

return;

}

Node\* temp = head;

//traverse list to find node with given value

while (temp != nullptr && temp->val != value)

{

temp = temp->next;

}

if (temp == nullptr)

{

cout << "Node by value " << value << " not found" << endl;

return;

}

//if the node to delete is the head

if (temp == head)

{

head = head->next;

if (head != nullptr)

{

head->prev = nullptr;

}

delete temp;

cout << "Node by value " << value << " deleted!" << endl;

return;

}

//if the node to delete is in the middle or end

if (temp->next != nullptr)

{

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

}

if (temp->prev != nullptr)

{

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

}

delete temp;

cout << "Node by value " << value << " deleted!" << endl;

}

//function to print doubly linklist

void printList(Node\* head)

{

Node\* temp = head;

while (temp != nullptr)

{

cout << temp->val << " ";

temp = temp->next;

}

cout << endl;

}

int main()

{

//making doubly linklist

Node\* head = new Node(15);

Node\* second = new Node(35);

Node\* third = new Node(50);

//linking

head->next = second;

second->prev = head;

second->next = third;

third->prev = second;

//printing original list

cout << "Original List: ";

printList(head);

//deleting node by value

int valueToDelete = 35;

deleteNodeByValue(head, valueToDelete);

//list after deletion

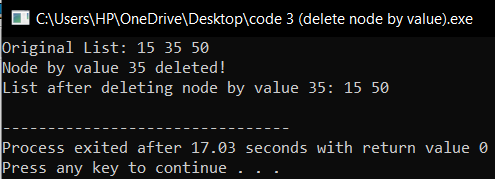
cout << "List after deleting node by value " << valueToDelete << ": ";

printList(head);

return 0;

}

**Output:**



**Question 4: How would you delete a node at a specific position in a doubly linked list? Show it in code.**

#include <iostream>

using namespace std;

struct Node {

int val;

Node\* next;

Node\* prev;

Node(int data) {

val = data;

next = nullptr;

prev = nullptr;

}

};

//function to delete node at specific position

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

if (head == nullptr) {

cout << "List is empty" << endl;

return;

}

if (position <= 0) {

cout << "Invalid position" << endl;

return;

}

Node\* temp = head;

if (position == 1) {

head = head->next;

if (head != nullptr) {

head->prev = nullptr;

}

delete temp;

cout << "Node at position 1 deleted." << endl;

return;

}

//traversing upto node at specific position

int count = 1;

while (temp != nullptr && count < position) {

temp = temp->next;

count++;

}

if (temp == nullptr) {

cout << "Position out of bounds, no node deleted" << endl;

return;

}

//if node to delete is in the middle or end

if (temp->next != nullptr) {

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

}

if (temp->prev != nullptr) {

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

}

delete temp;

cout << "Node at position " << position << " deleted!" << endl;

}

//function to print doubly linklist

void printList(Node\* head) {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->val << " ";

temp = temp->next;

}

cout << endl;

}

int main() {

//making doubly linklist

Node\* head = new Node(15);

Node\* second = new Node(35);

Node\* third = new Node(50);

//linking

head->next = second;

second->prev = head;

second->next = third;

third->prev = second;

//original list

cout << "Original List: ";

printList(head);

//deleting node at specific position

int positionToDelete = 2;

deleteAtPosition(head, positionToDelete);

//list after deletion

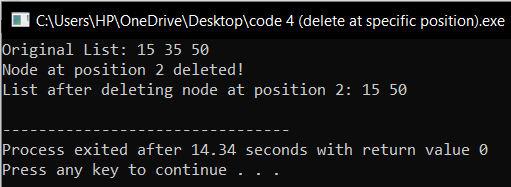
cout << "List after deleting node at position " << positionToDelete << ": ";

printList(head);

return 0;

}

**Output:**



**Question 5: After deleting a node, how will you write the forward and reverse traversal functions?**

After deleting a node, you can implement forward traversal and reverse traversal functions to print the elements of the doubly linked list in both directions. These functions traverse the list by following the next and prev pointers, respectively.

**Code:**

#include <iostream>

using namespace std;

struct Node

{

int val;

Node\* next;

Node\* prev;

Node(int data)

{

val = data;

next = nullptr;

prev = nullptr;

}

};

//function to traverse list in forward direction

void forwardTraversal(Node\* head)

{

Node\* temp = head;

cout << "Forward Traversal: ";

while (temp != nullptr)

{

cout << temp->val << " ";

temp = temp->next;

}

cout << endl;

}

//function to traverse list in reverse direction

void reverseTraversal(Node\* head)

{

if (head == nullptr)

{

cout << "List is empty." << endl;

return;

}

//moving to last node

Node\* temp = head;

while (temp->next != nullptr)

{

temp = temp->next;

}

//traverse backward using the prev pointer

cout << "Reverse Traversal: ";

while (temp != nullptr)

{

cout << temp->val << " ";

temp = temp->prev;

}

cout << endl;

}

//function to print list in forward and reverse order

void printList(Node\* head)

{

forwardTraversal(head);

reverseTraversal(head);

}

int main()

{

//making doubly linklist

Node\* head = new Node(15);

Node\* second = new Node(35);

Node\* third = new Node(50);

//linking

head->next = second;

second->prev = head;

second->next = third;

third->prev = second;

//printing list

printList(head);

//deleting the second node

if (head->next != nullptr)

{

Node\* toDelete = head->next;

head->next = toDelete->next;

if (toDelete->next != nullptr)

{

toDelete->next->prev = head;

}

delete toDelete;

}

//list after deletion

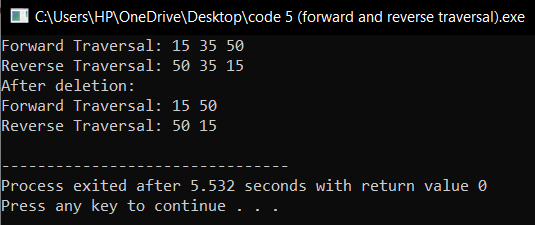
cout << "After deletion:" << endl;

printList(head);

return 0;

}

**Output:**



**Circular linked list**

**Question 1: Write a program to delete the first node in a circular linked list.**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

//function to delete first node in circular linklist

void deleteStart(Node\*\* head) {

if (\*head == NULL) {

cout << "List is empty" << endl;

return;

}

Node\* temp = \*head;

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

delete \*head;

\*head = NULL;

cout << "List is now empty" << endl;

return;

}

Node\* last = \*head;

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

last = last->next;

}

//updating head to next node

Node\* newHead = (\*head)->next;

last->next = newHead;

delete \*head; //deleting old head

\*head = newHead;

cout << "First node deleted!" << endl;

}

//function to insert node at end of circular linklist

void insertEnd(Node\*\* head, int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

newNode->next = newNode;

\*head = newNode;

} else {

Node\* temp = \*head;

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

temp = temp->next;

}

temp->next = newNode; //linking last node to new node

newNode->next = \*head;

}

}

//function to display circular linklist

void display(Node\* head) {

if (head == NULL) {

cout << "List is empty" << endl;

return;

}

Node\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

cout << endl;

}

int main() {

Node\* head = NULL;

//inserting nodes

insertEnd(&head, 45);

insertEnd(&head, 80);

insertEnd(&head, 15);

cout << "Original List: ";

display(head);

//deleting first node

deleteStart(&head);

cout << "After Deletion at Start: ";

display(head);

//deleting first node again

deleteStart(&head);

cout << "After Deleting Again: ";

display(head);

//deleting all nodes

deleteStart(&head);

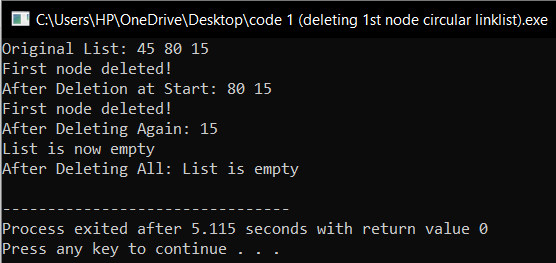
cout << "After Deleting All: ";

display(head);

return 0;

}

**Output:**



**Question 2: How can you delete the last node in a circular linked list? Write the code.**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

//function to delete last node in circular linklist

void deleteLastNode(Node\*\* head) {

if (\*head == NULL) {

cout << "List is empty" << endl;

return;

}

Node\* temp = \*head;

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

delete \*head;

\*head = NULL;

cout << "Last node deleted, List is now empty" << endl;

return;

}

Node\* prev = nullptr;

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

prev = temp;

temp = temp->next;

}

//updating second-to-last node's next pointer

prev->next = \*head;

delete temp;

cout << "Last node deleted!" << endl;

}

//function to insert node at end of circular linklist

void insertEnd(Node\*\* head, int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

newNode->next = newNode;

\*head = newNode;

} else {

Node\* temp = \*head;

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

temp = temp->next;

}

temp->next = newNode; //linking last node to new node

newNode->next = \*head;

}

}

//function to display circular linked list

void display(Node\* head) {

if (head == NULL) {

cout << "List is empty" << endl;

return;

}

Node\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

cout << endl;

}

int main() {

Node\* head = NULL;

//inserting nodes

insertEnd(&head, 10);

insertEnd(&head, 20);

insertEnd(&head, 30);

insertEnd(&head, 40);

cout << "Original List: ";

display(head);

//deleting last node

deleteLastNode(&head);

cout << "After Deleting Last Node: ";

display(head);

//deleting last node again

deleteLastNode(&head);

cout << "After Deleting Last Node Again: ";

display(head);

//deleting all nodes

deleteLastNode(&head);

deleteLastNode(&head);

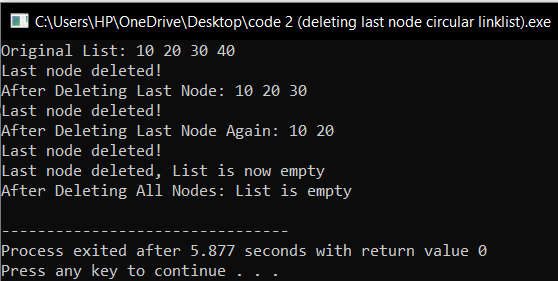
cout << "After Deleting All Nodes: ";

display(head);

return 0;

}

**Output:**



**Question 3: Write a function to delete a node by its value in a circular linked list.**

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

{

if (\*head == NULL) { // If the list is empty

cout << "List is empty." << endl;

return;

}

Node\* temp = \*head;

Node\* prev = NULL;

// if the head node is the node to be deleted

if ((\*head)->data == value)

{

if ((\*head)->next == \*head)

{

delete \*head;

\*head = NULL;

cout << "Node byvalue " << value << " deleted, List is now empty." << endl;

return;

}

Node\* last = \*head;

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

{

last = last->next;

}

// updating head

Node\* newHead = (\*head)->next;

last->next = newHead;

delete \*head;

\*head = newHead;

cout << "Node with value " << value << " deleted!" << endl;

return;

}

// traversing to find the node with given value

do

{

prev = temp;

temp = temp->next;

if (temp->data == value)

{

prev->next = temp->next; // updating previous node's next pointer

delete temp;

cout << "Node by value " << value << " deleted!" << endl;

return;

}

} while (temp != \*head);

cout << "Node with value " << value << " not found." << endl;

}

**Question 4: How will you delete a node at a specific position in a circular linked list? Write code for it.**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

//function to insert node at end

void insertEnd(Node\*\* head, int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

newNode->next = newNode;

\*head = newNode;

} else {

Node\* temp = \*head;

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

temp = temp->next;

}

temp->next = newNode;

newNode->next = \*head;

}

}

//function to display circular linklist

void display(Node\* head) {

if (head == NULL) {

cout << "List is empty." << endl;

return;

}

Node\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

cout << endl;

}

//function to delete a node at a specific position

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

if (\*head == NULL) {

cout << "List is empty." << endl;

return;

}

Node\* temp = \*head;

if (position == 1)

{

if ((\*head)->next == \*head)

{

delete \*head;

\*head = NULL;

cout << "Node at position " << position << " deleted, List is now empty" << endl;

return;

}

Node\* last = \*head;

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

{

last = last->next;

}

//updating head

Node\* newHead = (\*head)->next;

last->next = newHead;

delete \*head;

\*head = newHead;

cout << "Node at position " << position << " deleted!" << endl;

return;

}

//traverse to node at given position

Node\* prev = NULL;

for (int i = 1; i < position && temp->next != \*head; i++)

{

prev = temp;

temp = temp->next;

}

//if position is invalid

if (temp->next == \*head && position != 1)

{

cout << "Invalid position." << endl;

return;

}

//updating pointers to remove the node

prev->next = temp->next;

delete temp;

cout << "Node at position " << position << " deleted!" << endl;

}

int main()

{

Node\* head = NULL;

//inserting nodes

insertEnd(&head, 10);

insertEnd(&head, 20);

insertEnd(&head, 30);

insertEnd(&head, 40);

insertEnd(&head, 50);

cout << "Original List: ";

display(head);

//deleting node at position 1

deleteAtPosition(&head, 1);

cout << "After deleting node at position 1: ";

display(head);

//deleting node at position 3

deleteAtPosition(&head, 3);

cout << "After deleting node at position 3: ";

display(head);

//delete node at invalid position

deleteAtPosition(&head, 10);

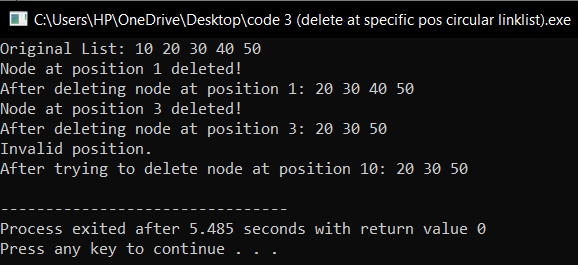
cout << "After trying to delete node at position 10: ";

display(head);

return 0;

}

**Output:**



**Question 5: Write a program to show forward traversal after deleting a node in a circular linked list.**

#include <iostream>

using namespace std;

struct Node {

int val;

Node\* next;

Node\* prev;

Node(int data) {

val = data;

next = nullptr;

prev = nullptr;

}

};

//function to traverse list in forward direction

void forwardTraversal(Node\* head) {

if (head == nullptr) {

cout << "List is empty" << endl;

return;

}

Node\* temp = head;

cout << "Forward Traversal: ";

do {

cout << temp->val << " ";

temp = temp->next;

} while (temp != head);

cout << endl;

}

//function to traverse list in reverse direction

void reverseTraversal(Node\* head) {

if (head == nullptr) {

cout << "List is empty." << endl;

return;

}

Node\* temp = head;

while (temp->next != head) {

temp = temp->next;

}

//traverse backward using prev pointer

cout << "Reverse Traversal: ";

do {

cout << temp->val << " ";

temp = temp->prev;

} while (temp != head);

cout << endl;

}

//function to print list in forward and reverse order

void printList(Node\* head) {

forwardTraversal(head);

reverseTraversal(head);

}

//function to delete node at specific position

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

if (head == nullptr) {

cout << "List is empty." << endl;

return;

}

Node\* temp = head;

//deleting head node

if (position == 1) {

if (head->next == head) {

delete head;

head = nullptr;

cout << "Node at position " << position << " deleted, List is now empty." << endl;

return;

}

//update previous node to point to second node

Node\* last = head;

while (last->next != head) {

last = last->next;

}

Node\* newHead = head->next;

last->next = newHead;

newHead->prev = last;

delete head;

head = newHead;

cout << "Node at position " << position << " deleted!" << endl;

return;

}

//traverse to the node at the given position

for (int i = 1; temp != nullptr && i < position; i++) {

temp = temp->next;

}

if (temp == nullptr || temp->next == head) {

cout << "Invalid position." << endl;

return;

}

//update pointers to remove node

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

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

delete temp;

cout << "Node at position " << position << " deleted!" << endl;

}

int main() {

Node\* head = new Node(15);

Node\* second = new Node(35);

Node\* third = new Node(50);

//linking nodes

head->next = second;

second->prev = head;

second->next = third;

third->prev = second;

third->next = head;

head->prev = third;

//original list

cout << "Original List: ";

printList(head);

//deleting the second node

deleteNode(head, 2);

//list after deletion

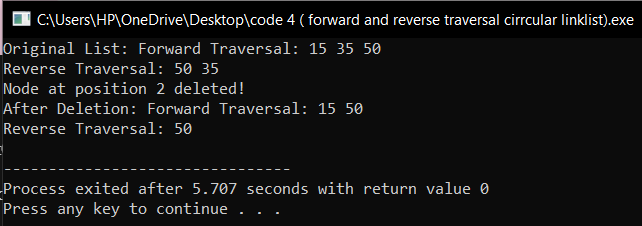
cout << "After Deletion: ";

printList(head);

return 0;

}

**Output:**



**Binary Search Tree**

**Question 1: Write a program to count all the nodes in a binary search tree.**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

//constructor to create a new node

Node(int value) {

data = value;

left = right = nullptr;

}

};

//function to insert a new node in BST

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;

}

//function to count number of nodes in the BST

int countNodes(Node\* root) {

if (root == nullptr) {

return 0;

}

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

}

int main() {

Node\* root = nullptr;

//inserting nodes in BST

root = insert(root, 15);

insert(root, 10);

insert(root, 20);

insert(root, 8);

insert(root, 12);

insert(root, 17);

insert(root, 25);

//counting and displaying total number of nodes

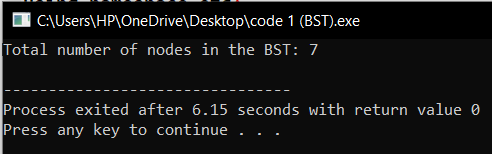
int totalNodes = countNodes(root);

cout << "Total number of nodes in the BST: " << totalNodes << endl;

return 0;

}

**Output:**



**Question 2: How can you search for a specific value in a binary search tree? Write the code.**

#include <iostream>

using namespace std;

struct parentNode

{

int data;

parentNode\* LC;

parentNode\* RC;

parentNode(int val)

{

data = val;

LC = NULL;

RC = NULL;

}

};

parentNode\* search(parentNode\* Root, int data)

{

if (Root == NULL || Root->data == data)

return Root;

if (Root->data < data)

return search(Root->RC, data);

return search(Root->LC, data);

}

int main()

{

parentNode\* Root = new parentNode(80);

Root->LC = new parentNode(20);

Root->RC = new parentNode(40);

Root->LC->LC = new parentNode(30);

Root->LC->RC = new parentNode(10);

Root->RC->LC = new parentNode(70);

Root->RC->RC = new parentNode(50);

int value;

cout << "Enter value to search: ";

cin >> value;

if (search(Root, value) != NULL)

cout << "Found" << endl;

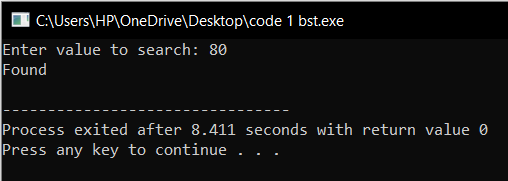
else

cout << "Not Found" << endl;

return 0;

}

**Output:**



**Question 3: Write code to traverse a binary search tree in in-order, pre-order, and postorder.**

#include <iostream>

using namespace std;

struct parentNode {

int data;

parentNode\* LC;

parentNode\* RC; };

parentNode\* createNode(int data) {

parentNode\* n = new parentNode();

n->data = data;

n->LC = nullptr;

n->RC = nullptr;

return n;

}

void preOrder(parentNode\* Root) {

if (Root != nullptr) {

cout << Root->data << " ";

preOrder(Root->LC);

preOrder(Root->RC);

}}

void postOrder(parentNode\* Root)

{

if (Root != nullptr)

{

postOrder(Root->LC);

postOrder(Root->RC);

cout << Root->data << " ";

}

}

void inOrder(parentNode\* Root)

{

if (Root != nullptr)

{

cout << Root->data << " ";

inOrder(Root->RC);

}}

int main() {

parentNode\* n = createNode(1);

parentNode\* n1 = createNode(2);

parentNode\* n2 = createNode(3);

parentNode\* n3 = createNode(4);

parentNode\* n4 = createNode(5);

parentNode\* n5 = createNode(6);

n->LC = n1;

n->RC = n2;

n1->LC = n3;

n1->RC = n4;

n2->RC = n5;

cout << "Inorder Traversal:" << endl;

inOrder(n);

cout << "\n";

cout << "Preorder Traversal:" << endl;

preOrder(n);

cout << "\n";

cout << "Postorder Traversal:" << endl;

postOrder(n);

cout << "\n";

delete n4;

delete n3;

delete n2;

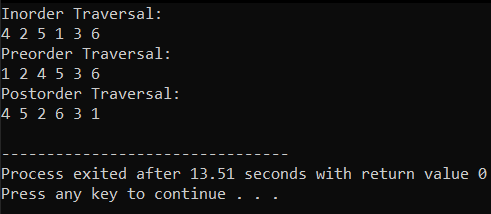
delete n1;

delete n;

return 0;

}

**Output:**



**Question 4: How will you write reverse in-order traversal for a binary search tree? Show it in code.**

#include <iostream>

using namespace std;

struct parentNode {

int data;

parentNode\* LC;

parentNode\* RC;

};

parentNode\* createNode(int data) {

parentNode\* n = new parentNode();

n->data = data;

n->LC = nullptr;

n->RC = nullptr;

return n;

}

void reverseInOrder(parentNode\* Root) {

if (Root != nullptr) {

reverseInOrder(Root->RC); // First, visit the right subtree

cout << Root->data << " "; // Then, visit the current node

reverseInOrder(Root->LC); // Finally, visit the left subtree

}

}

int main() {

parentNode\* n = createNode(1);

parentNode\* n1 = createNode(2);

parentNode\* n2 = createNode(3);

parentNode\* n3 = createNode(4);

parentNode\* n4 = createNode(5);

parentNode\* n5 = createNode(6);

n->LC = n1;

n->RC = n2;

n1->LC = n3;

n1->RC = n4;

n2->RC = n5;

cout << "Reverse Inorder Traversal:" << endl;

reverseInOrder(n);

cout << "\n";

delete n4;

delete n3;

delete n2;

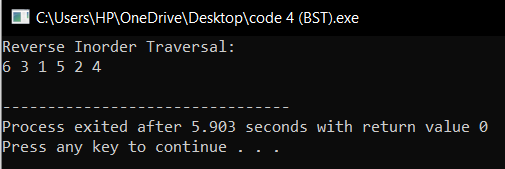
delete n1;

delete n;

return 0;

}

**Output:**



**Question 5: Write a program to check if there are duplicate values in a binary search tree.**

#include <iostream>

using namespace std;

struct parentNode {

int data;

parentNode\* LC;

parentNode\* RC;

parentNode(int val) {

data = val;

LC = nullptr;

RC = nullptr;

}

};

void duplicate(parentNode\* Root, int val) {

if (Root == nullptr) {

cout << "Duplicate not found" << endl;

return;

}

if (Root->data == val) {

cout << "Duplicate found" << endl;

return;

}

if (val < Root->data) {

duplicate(Root->LC, val);

}

else {

duplicate(Root->RC, val);

}

}

int main() {

parentNode\* Root = new parentNode(50);

Root->LC = new parentNode(30);

Root->RC = new parentNode(70);

Root->LC->LC = new parentNode(20);

Root->LC->RC = new parentNode(40);

Root->RC->LC = new parentNode(60);

Root->RC->RC = new parentNode(80);

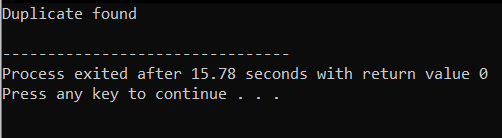
int val = 40;

duplicate(Root, val);

return 0;

}

**Output:**



**Question 6: 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.**

#include <iostream>

using namespace std;

struct Node {

int value;

Node\* left;

Node\* right;

Node(int key) {

value = key;

left = right = nullptr;

}

};

Node\* findMin(Node\* node) {

Node\* current = node;

while (current && current->left != nullptr) {

current = current->left;

}

return current;

}

Node\* deleteNode(Node\* 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 {

if (root->left == nullptr && root->right == nullptr) {

delete root;

return nullptr;

}

else if (root->left == nullptr) {

Node\* temp = root->right;

delete root;

return temp;

}

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

Node\* temp = root->left;

delete root;

return temp;

}

else {

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

root->value = temp->value;

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

}

}

return root;

}

void inorder(Node\* root) {

if (root != nullptr) {

inorder(root->left);

cout << root->value << " ";

inorder(root->right);

}

}

int main() {

Node\* root = new Node(50);

root->left = new Node(30);

root->right = new Node(70);

root->left->left = new Node(20);

root->left->right = new Node(40);

root->right->left = new Node(60);

root->right->right = new Node(80);

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

inorder(root);

cout << endl;

int key = 50;

root = deleteNode(root, key);

cout << "\nInorder traversal after deleting node " << key <<endl;

inorder(root);

cout << endl;

return 0;

}

**Output:**

