Final Assignment

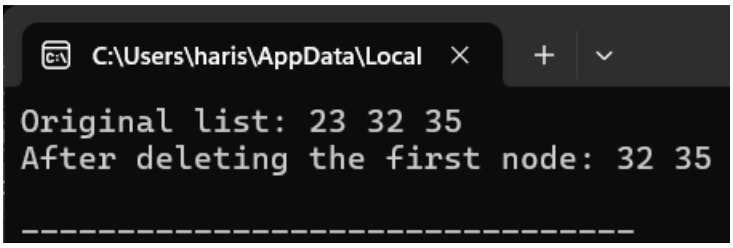
Data structure and algorithm

Submitted to: Mam Irsha

Submitted by : Haris Awan

Reg no: 2023-BS-AI-023

Program 1 **output:**



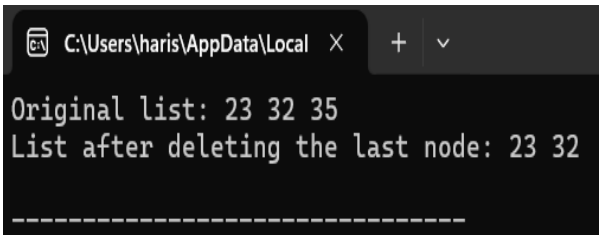

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* prev;
    Node* next;
    Node(int val) : data(val), prev(nullptr), next(nullptr) {}
};
// Function which delete the first node
void deleteFirstNode(Node*& head) {
    if (head == nullptr) { // check If the list is empty
        cout << "The list is already empty." << endl;
        return;
    }
    Node* temp = head;
    head = head->next;
    if (head != nullptr) {
        head->prev = nullptr;
    }
    delete temp;
}
// Function which display the linked list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
```

```
}
// Function which insert a node
void insert(Node*& head, int data)
{
    Node* newNode = new Node(data);
    if (head == nullptr) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
int main()
{
    Node* head = nullptr;
    insert(head, 23);
    insert(head, 32);
    insert(head, 35);
    cout << "Original list: ";
    displayList(head);
    deleteFirstNode(head);
    cout << "After deleting the first node: ";
    displayList(head);
    return 0;
}
```

Program 2:

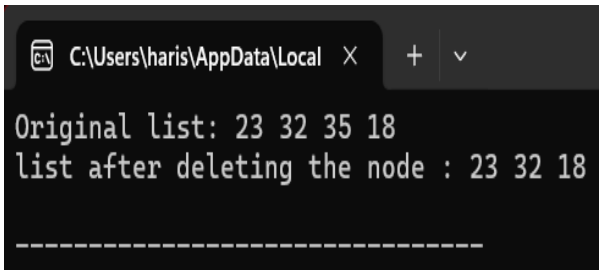
```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* prev;
    Node* next;

    Node(int val) : data(val), prev(nullptr), next(nullptr) {}
};
// Function which delete the last node
void deleteLastNode(Node*& head) {
    if (head == nullptr) { // If the list is empty
        cout << "The list is already empty." << endl;
        return;
    }
    if (head->next == nullptr) { // If the list has only one node
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->prev->next = nullptr;
    delete temp;
}
// Function which display the link list
void displayList(Node* head) {
```

output:



```
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
// Function to insert a node
void insert(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) {
        head = newNode;
        return;         }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;         }
int main() {
    Node* head = nullptr;
    insert(head, 23);
    insert(head, 32);
    insert(head, 35);
    cout << "Original list: ";
    displayList(head);
    deleteLastNode(head);
    cout << "List after deleting the last node: ";
    displayList(head);
    return 0;        }
```

Program 3: **output:**



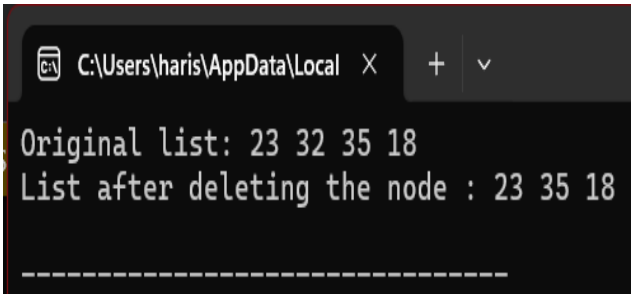

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* prev;
    Node* next;
    Node(int val) : data(val), prev(nullptr), next(nullptr) {}
};
// Function which delete a node
void deleteNodeByValue(Node*& head, int value) {
    if (head == nullptr) {
        cout << "The list is empty." << endl;
        return;
    }
    Node* temp = head;
    // Search the node with the specified value
    while (temp != nullptr && temp->data != value) {
        temp = temp->next;
    }
    if (temp == nullptr) {
        cout << "Value " << value << " not found in the list." << endl;
        return;
    }
    if (temp == head) {
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        }
    } else if (temp->next == nullptr) {
```

```
        temp->prev->next = nullptr;
    } else {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
    }
    delete temp;
}
// Function which display the link list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
// Function which insert a node
void insert(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
```

```
int main() {
    Node* head = nullptr;
    insert(head, 23);
    insert(head, 32);
    insert(head, 35);
    insert(head, 18);
    cout << "Original list: ";
    displayList(head);
    deleteNodeByValue(head, 35);
    cout<<"list after deleting the node : ";
    displayList(head);
    return 0;
}
```

Program 4:

output:




```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* prev;
    Node* next;

    Node(int val) : data(val), prev(nullptr), next(nullptr) {}
};
// Function which delete a node
void deleteNodeAtPosition(Node*& head, int position) {
    if (head == nullptr) {
        cout << "The list is empty." << endl;
        return;
    }
    if (position <= 0) {
```

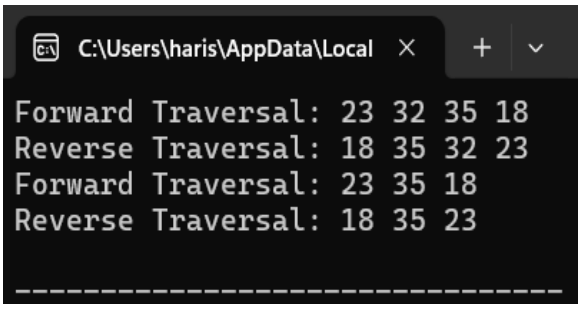
```
        cout << "Invalid position. Position should be greater than 0." << endl;
        return;
    }
    Node* temp = head;
    int currentIndex = 1;
    while (temp != nullptr && currentIndex < position) {
        temp = temp->next;
        currentIndex++;
    }
    if (temp == nullptr) {
        cout << "Position " << position << " exceeds the list size." << endl;
        return;
    }
    if (temp == head) {
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        }
    } else if (temp->next == nullptr) {
        temp->prev->next = nullptr;
    } else {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
    }
    delete temp;
}
// Function which display the link list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
```

```
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
// Function to insert a node
void insert(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;            }
    temp->next = newNode;
    newNode->prev = temp;
}
int main() {
    Node* head = nullptr;
    insert(head, 23);
    insert(head, 32);
    insert(head, 35);
    insert(head, 18);
    cout << "Original list: ";
    displayList(head);
    deleteNodeAtPosition(head, 2);
    cout << "List after deleting the node : ";
    displayList(head);
    return 0;        }
```

Program 5:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* prev;
    Node* next;
    Node(int val) : data(val), prev(nullptr), next(nullptr) {}
};
// Function which perform forward traversal
void forward(Node* head) {
    cout << "Forward Traversal: ";
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
// Function which perform reverse traversal
void reverse(Node* head) {
    if (head == nullptr) {
        cout << "Reverse Traversal: List is empty." << endl;
        return;
    }
    // Move to the last node
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
```

output:



```
    // Traverse backward from the last node
    cout << "Reverse Traversal: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->prev;
    }
    cout << endl;
}
// Function to insert a node
void insert(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) { // If the list is empty
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
// Function to delete a node
void deleteNodeAtPosition(Node*& head, int position) {
    if (head == nullptr) { // If the list is empty
        cout << "The list is empty." << endl;
        return;
    }
    if (position <= 0) { // Invalid position
        cout << "Invalid position." << endl;
```

```
        return;
    }
    Node* temp = head;
    int currentIndex = 1;
    // Traverse to the node at the specified position
    while (temp != nullptr && currentIndex < position) {
        temp = temp->next;
        currentIndex++;
    }
    if (temp == nullptr) {
        cout << "Position " << position << " exceeds the size." << endl;
        return;
    }
    if (temp == head) { // Deleting the head node
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        }
    } else if (temp->next == nullptr) { // Deleting the last node
        temp->prev->next = nullptr;
    } else { // Deleting a middle node
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
    }
    delete temp;
}
int main() {
    Node* head = nullptr;
    insert(head, 23);
    insert(head, 32);
```

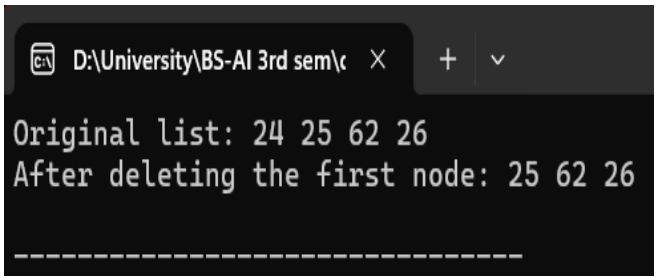
```
    insert(head, 35);
    insert(head, 18);
    // Perform forward and reverse traversal before deletion
    forward(head);
    reverse(head);
    deleteNodeAtPosition(head, 2);
    forward(head);
    reverse(head);
    return 0;
}
```

Circular link list

Program 1:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
// Function which delete the node
void deleteFirstNode(Node*& head) {
    if (head == nullptr) {
        cout << "List is empty. Nothing to delete." << endl;
        return;
    }
    if (head->next == head) {
        delete head;
        head = nullptr;
        return;
    }
```

output:

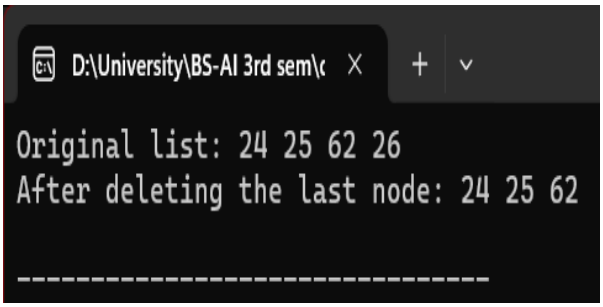


```
    Node* last = head;
    while (last->next != head) {
        last = last->next;
    }
    Node* temp = head;
    head = head->next;
    last->next = head;

    delete temp;
}
// Function which insert a node
void insert(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = head;
}
// Function which display the link list
void display(Node* head) {
    if (head == nullptr) {
        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 = nullptr;
    insert(head, 24);
    insert(head, 25);
    insert(head, 62);
    insert(head, 26);
    cout << "Original list: ";
    display(head);
    deleteFirstNode(head);
    cout << "After deleting the first node: ";
    display(head);
    return 0;
}
```

Program 2:

output:




```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
// Function which delete the last node
```

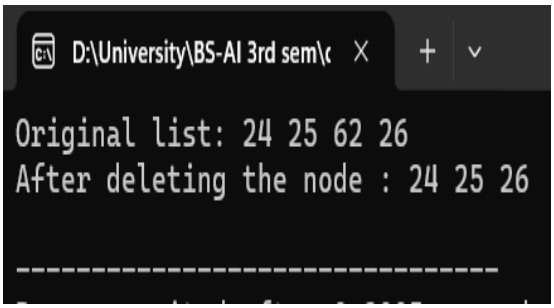
```
void deleteLastNode(Node*& head) {
    if (head == nullptr) {
        cout << "List is empty. Nothing to delete." << endl;
        return;
    }
    if (head->next == head) {
        delete head;
        head = nullptr;
        return;
    }
    Node* current = head;
    while (current->next->next != head) {
        current = current->next;
    }
    Node* last = current->next;
    current->next = head;
    delete last;
}
// Function which insert a node
void insert(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
```

```
    }
    temp->next = newNode;
    newNode->next = head;
}
// Function which display the link list
void display(Node* head) {
    if (head == nullptr) {
        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 = nullptr;
    insert(head, 24);
    insert(head, 25);
    insert(head, 62);
    insert(head, 26);
    cout << "Original list: ";
    display(head);
    deleteLastNode(head);
    cout << "After deleting the last node: ";
    display(head);
    return 0;
}
```

Program 3:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
// Function which delete a node
void deleteNodeByValue(Node*& head, int value) {
    if (head == nullptr) { // List is empty
        cout << "List is empty. Nothing to delete." << endl;
        return;
    }
    Node* current = head;
    Node* previous = nullptr;
    // Case 1: The node to be deleted is the only node in the list
    if (head->data == value && head->next == head) {
        delete head;
        head = nullptr;
        return;
    }
    // Case 2: The node to be deleted is the head node
    if (head->data == value) {
        while (current->next != head) {
            current = current->next;
        }
        Node* temp = head;
        head = head->next;
        current->next = head;
        delete temp;
```

output:



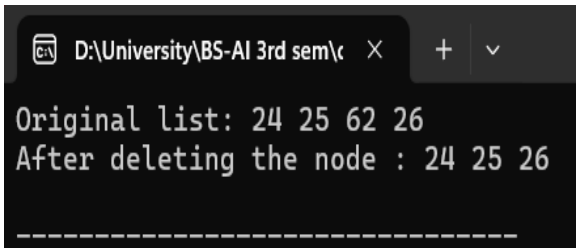
```
        return;
    }
    // Case 3: The node to be deleted is in the middle or end of the list
    do {
        previous = current;
        current = current->next;

        if (current->data == value) {
            previous->next = current->next;
            delete current;
            return;
        }
    } while (current != head);
    // If the value was not found
    cout << "Value " << value << " not found." << endl;
}
// Function which insert a node
void insert(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
    }
    temp->next = newNode;
```

```
    newNode->next = head;
}
// Function which display the link list
void display(Node* head) {
    if (head == nullptr) {
        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 = nullptr;
    insert(head, 24);
    insert(head, 25);
    insert(head, 62);
    insert(head, 26);
    cout << "Original list: ";
    display(head);
    deleteNodeByValue(head, 62);
    cout << "After deleting the node : ";
    display(head);
    return 0;
}
```

Program 4:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
// Function which delete a node
void deleteNodeAtPosition(Node*& head, int position) {
    if (head == nullptr) {
        cout << "List is empty. Nothing to delete." << endl;
        return;
    }
    // If the position is 0, delete the head node
    if (position == 0) {
        if (head->next == head) {
            delete head;
            head = nullptr;
        } else {
            Node* last = head;
            while (last->next != head) {
                last = last->next;
            }
            Node* temp = head;
            head = head->next;
            last->next = head;
            delete temp;
        }
        return;
    }
```

output:



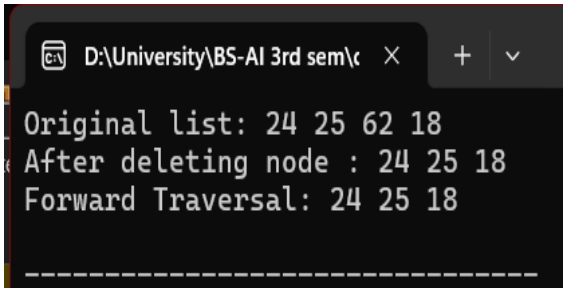
```
    Node* current = head;
    Node* previous = nullptr;
    int count = 0;
    while (current->next != head && count < position) {
        previous = current;
        current = current->next;
        count++;
    }
    if (current->next == head && count < position) {
        cout << "Position out of bounds." << endl;
        return;
    }
    previous->next = current->next;
    delete current;
}
// Function which insert a node
void insert(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = head;
```

```
}
// Function which display the link list
void display(Node* head) {
    if (head == nullptr) {
        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 = nullptr;
    insert(head, 24);
    insert(head, 25);
    insert(head, 62);
    insert(head, 26);
    cout << "Original list: ";
    display(head);
    deleteNodeAtPosition(head, 2);
    cout << "After deleting the node : ";
    display(head);
    return 0;
}
```

Program 5:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
// Function which delete a node
void deleteNodeAtPosition(Node*& head, int position) {
    if (head == nullptr) { // List is empty
        cout << "List is empty. Nothing to delete." << endl;
        return;
    }
    // If the position is 0, delete the head node
    if (position == 0) {
        // If there's only one node
        if (head->next == head) {
            delete head;
            head = nullptr;
        } else {
            Node* last = head;
            while (last->next != head) { // Find the last node
                last = last->next;
            }
            Node* temp = head;
            head = head->next; // Move the head pointer
            last->next = head; // Adjust the last node's next pointer
            delete temp; // Delete the old head
        }
        return;
```

output:



```
    }
    Node* current = head;
    Node* previous = nullptr;
    int count = 0;
    // Traverse to the desired position
    while (current->next != head && count < position) {
        previous = current;
        current = current->next;
        count++;
    }
    // If position is out of bounds
    if (current->next == head && count < position) {
        cout << "Position out of bounds." << endl;
        return;
    }
    // Delete the node
    previous->next = current->next;
    delete current;
}
// Function which insert a node
void insert(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
        return;        }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;  }
```

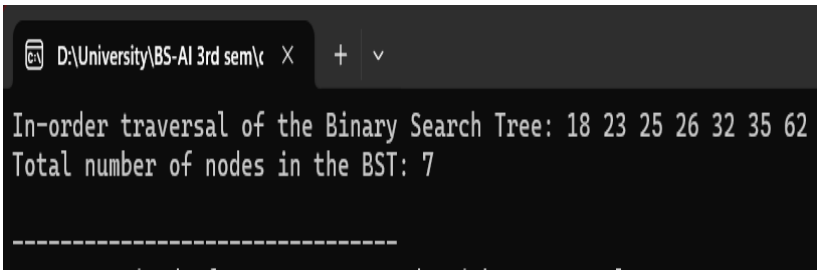
```
    temp->next = newNode;
    newNode->next = head;          }
// Function to display the circular linked list
void display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
int main() {
    Node* head = nullptr;
    insert(head, 24);
    insert(head, 25);
    insert(head, 62);
    insert(head, 18);
    cout << "Original list: ";
    display(head);
    deleteNodeAtPosition(head, 2);
    cout << "After deleting node : ";
    display(head);
    cout << "Forward Traversal: ";
    display(head);
    return 0;
}
```

Binary search tree

Program 1:

```
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;


    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};
// Function which insert a node in the Tree
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
// Function which counts the nodes
int countNodes(Node* root) {
    if (root == nullptr) {
        return 0;
```

output:

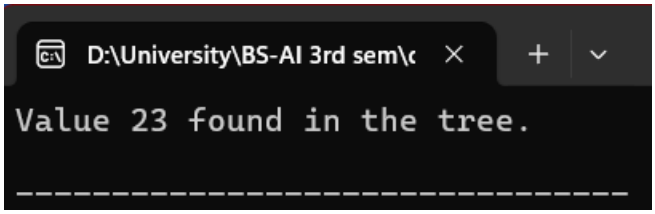


```
    }
    // Recursively count nodes in the left and right subtrees, and add 1 for the current node
    return 1 + countNodes(root->left) + countNodes(root->right);
}
// Function to perform an in-order traversal and print the tree
void inorderTraversal(Node* root) {
    if (root != nullptr) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }
}
int main() {
    Node* root = nullptr;
    root = insert(root, 18);
    root = insert(root, 23);
    root = insert(root, 25);
    root = insert(root, 32);
    root = insert(root, 35);
    root = insert(root, 62);
    root = insert(root, 26);
    cout << "In-order traversal of the Binary Search Tree: ";
    inorderTraversal(root);
    cout << endl;
    int nodeCount = countNodes(root);
    cout << "Total number of nodes in the BST: " << nodeCount << endl;

    return 0;
}
```

Program 2:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};
// Function to search for a value
bool search(Node* root, int key) {
    if (root == NULL) return false;
    if (root->data == key) return true;
    if (key < root->data) return search(root->left, key);
    return search(root->right, key);          }
int main() {
    Node* root = new Node(23);
    root->left = new Node(32);
    root->right = new Node(35);
    int searchKey = 23;
    if (search(root, searchKey)) {
        cout << "Value " << searchKey << " found in the tree." << endl;
    } else {
        cout << "Value " << searchKey << " not found in the tree." << endl;
    }
    return 0;       }
```

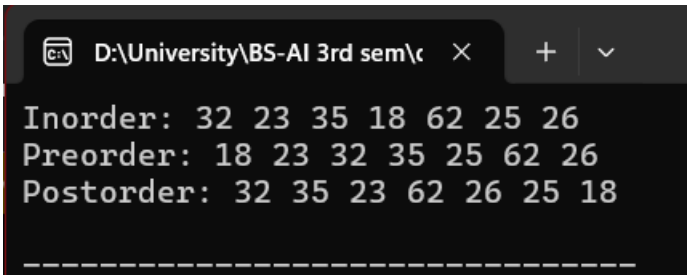
output:



Program 3:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};
// Traversal functions
void inorder(Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}
void preorder(Node* root) {
    if (root == NULL) return;
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}
void postorder(Node* root) {
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
```

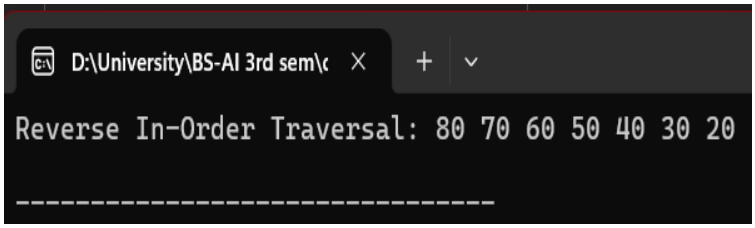
output:



```
    cout << root->data << " ";
}
int main() {
    Node* root = new Node(18);
    root->left = new Node(23);
    root->right = new Node(25);
    root->left->left = new Node(32);
    root->left->right = new Node(35);
    root->right->left = new Node(62);
    root->right->right = new Node(26);
    cout << "Inorder: ";
    inorder(root);
    cout << endl;
    cout << "Preorder: ";
    preorder(root);
    cout << endl;
    cout << "Postorder: ";
    postorder(root);
    cout << endl;
    return 0;
}
```

Program 4:

output:




```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int value) : data(value), left(nullptr), right(nullptr) {}
};
```

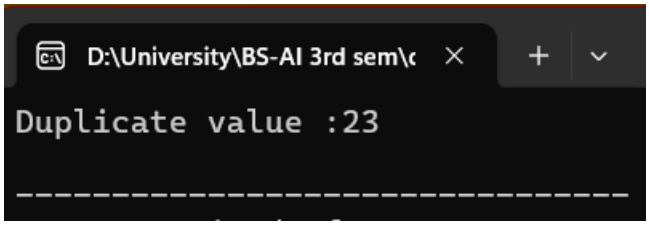
```
// Function to perform reverse in-order traversal
void reverseInOrderTraversal(Node* root) {
    if (!root) return;
    // Traverse the right subtree first
    reverseInOrderTraversal(root->right);
    // Visit the root (current node)
    cout << root->data << " ";
    // Traverse the left subtree
    reverseInOrderTraversal(root->left);
}
Node* insertNode(Node* root, int value) {
    if (!root) {
        return new Node(value);
    }
    if (value < root->data) {
        root->left = insertNode(root->left, value);
    } else {
        root->right = insertNode(root->right, value);
    }
    return root;
}
int main() {
    Node* root = nullptr;
    root = insertNode(root, 50);
    root = insertNode(root, 30);
    root = insertNode(root, 70);
    root = insertNode(root, 20);
    root = insertNode(root, 40);
    root = insertNode(root, 60);
    root = insertNode(root, 80);
```

```
    cout << "Reverse In-Order Traversal: ";
    reverseInOrderTraversal(root);
    cout << endl;
    return 0;
}
```

Program 5:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};
// Function to insert a value
Node* insert(Node* root, int val) {
    if (root == NULL) {
        return new Node(val);
    }
    if (val < root->data) {
        root->left = insert(root->left, val);
    } else if (val > root->data) {
        root->right = insert(root->right, val);
    } else {
        cout << "Duplicate value :" << val <<endl;
    }
```

output:

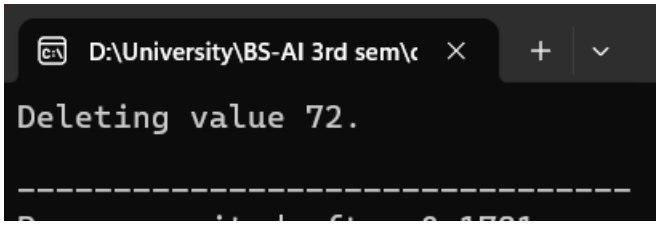


```
    return root;
}
int main() {
    Node* root = NULL;
    root = insert(root, 18);
    root = insert(root, 23);
    root = insert(root, 25);
    root = insert(root, 32);
    root = insert(root, 35);
    root = insert(root, 62);
    root = insert(root, 23);
    root = insert(root, 19);
    return 0;
}
```

Program 6:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};
// Function to find the minimum value node
Node* findMin(Node* root) {
    while (root && root->left != NULL) {
```

output:



```
        root = root->left;
    }
    return root;
}
// Function to delete a node
Node* deleteNode(Node* root, int key) {
    if (root == NULL) return root;
    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    } else {
        if (root->left == NULL) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
            delete root;
            return temp;
        }
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
int main() {
    Node* root = new Node(18);
    root->left = new Node(23);
```

```
    root->right = new Node(35);
            root->left->left = new Node(32);
    root->left->right = new Node(72);
    root->right->left = new Node(62);
    root->right->right = new Node(26);
    cout << "Deleting value 72." << endl;
    root = deleteNode(root, 72);
    return 0;
}
```