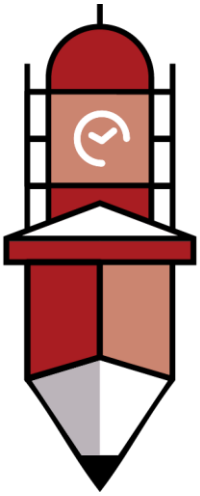


The University of Faisalabad

Data Structures

Final Assignment

Submitted by:

Muhammad Zain

Submitted to:

Ms. Irsha Qureshi

Registration no:

2023-BSAI-052(III)A

Department:

CS(Artifical Intelligence)

Doubly link list

1-Delete the First Node:

Code:

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
   int data;
   Node* next;
   Node* prev;
};
// Function to create a new node
Node* createNode(int data) {
   Node* newNode = new Node();
   newNode->data = data;
   newNode->next = newNode->prev = NULL;
   return newNode;
}
// Function to insert a new node at the end
void insertNode(Node** head, int data) {
   Node* newNode = createNode(data);
   if (*head == NULL) {
      *head = newNode;
      return;
   }

   Node* lastNode = *head;
   while (lastNode->next != NULL) {
      lastNode = lastNode->next;
```

```
    }
    lastNode->next = newNode;
    newNode->prev = lastNode;
}
// Function to delete the first node
void deleteFirstNode(Node** head) {
    if (*head == NULL) {
        cout << "List is empty" << endl;
        return;
    }
    Node* temp = *head;
    *head = (*head)->next;

    if (*head != NULL) {
        (*head)->prev = NULL;
    }
    delete temp;
}
// Function to print the list
void printList(Node* head) {
    while (head != NULL) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    insertNode(&head, 10);
```

```
    insertNode(&head, 20);
    insertNode(&head, 30);
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Original List: ";
    printList(head);
    deleteFirstNode(&head);
    cout << "List after deleting first node: ";
    printList(head);
    return 0;
}
```

Your Output

```
Original List: 10 20 30 40 50
List after deleting first node: 20 30 40 50
```

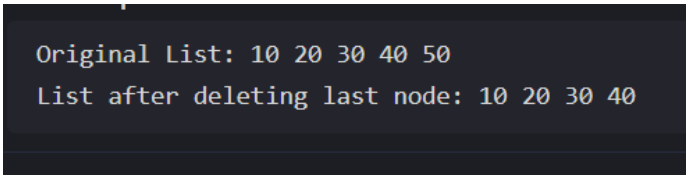
2-Delete the Last Node

Code:

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
    Node* prev;
};
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
```

```
    newNode->data = data;
    newNode->next = newNode->prev = NULL;
    return newNode;
}
// Function to insert a new node at the end
void insertNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* lastNode = *head;
    while (lastNode->next != NULL) {
        lastNode = lastNode->next;
    }
    lastNode->next = newNode;
    newNode->prev = lastNode;
}
// Function to delete the last node
void deleteLastNode(Node** head) {
    if (*head == NULL) {
        cout << "List is empty" << endl;
        return;
    }
    if ((*head)->next == NULL) {
        delete *head;
        *head = NULL;
        return;
    }
```

```
    Node* lastNode = *head;
    while (lastNode->next != NULL) {
        lastNode = lastNode->next;
    }
    lastNode->prev->next = NULL;
    delete lastNode;
}
// Function to print the list
void printList(Node* head) {
    while (head != NULL) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    insertNode(&head, 10);
    insertNode(&head, 20);
    insertNode(&head, 30);
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Original List: ";
    printList(head);
    deleteLastNode(&head);
    cout << "List after deleting last node: ";
    printList(head);
    return 0;
}
```



3-Delete a Node by Value

Code:

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

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = newNode->prev = NULL;
    return newNode;
}

// Function to insert a new node at the end
void insertNode(Node** head, int data) {
    Node* newNode = createNode(data);

    if (*head == NULL) {
        *head = newNode;
```

```
        return;
    }

    Node* lastNode = *head;
    while (lastNode->next != NULL) {
        lastNode = lastNode->next;
    }

    lastNode->next = newNode;
    newNode->prev = lastNode;
}

// Function to delete a node by its value
void deleteNodeByValue(Node** head, int value) {
    if (*head == NULL) {
        cout << "List is empty" << endl;
        return;
    }

    // Check if the node to be deleted is the head node
    if ((*head)->data == value) {
        Node* temp = *head;
        *head = (*head)->next;

        if (*head != NULL) {
            (*head)->prev = NULL;
        }

        delete temp;
```

```
        return;
    }

    Node* temp = *head;
    while (temp->next != NULL) {
        if (temp->next->data == value) {
            Node* nodeToDelete = temp->next;
            temp->next = temp->next->next;

            if (temp->next != NULL) {
                temp->next->prev = temp;
            }

            delete nodeToDelete;
            return;
        }
        temp = temp->next;
    }

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

// Function to print the list in forward order
void printForward(Node* head) {
    while (head != NULL) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
```

```
}
// Function to print the list in reverse order
void printReverse(Node* head) {
    if (head == NULL) {
        cout << "List is empty" << endl;
        return;
    }
    Node* lastNode = head;
    while (lastNode->next != NULL) {
        lastNode = lastNode->next;
    }
    while (lastNode != NULL) {
        cout << lastNode->data << " ";
        lastNode = lastNode->prev;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    insertNode(&head, 10);
    insertNode(&head, 20);
    insertNode(&head, 30);
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Original List (Forward): ";
    printForward(head);
    cout << "Original List (Reverse): ";
    printReverse(head);
```

```
    deleteNodeByValue(&head, 30);
    cout << "List after deleting node with value 30 (Forward): ";
    printForward(head);
    cout << "List after deleting node with value 30 (Reverse): ";
    printReverse(head);
    return 0;
}
```

```
Original List (Forward): 10 20 30 40 50
Original List (Reverse): 50 40 30 20 10
List after deleting node with value 30 (Forward): 10 20 40 50
List after deleting node with value 30 (Reverse): 50 40 20 10
```

4-Delete a Node at a Specific Position

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
    Node* prev;
};
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = newNode->prev = NULL;
    return newNode;
}
// Function to insert a new node at the end
void insertNode(Node** head, int data) {
```

```
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* lastNode = *head;
    while (lastNode->next != NULL) {
        lastNode = lastNode->next;
    }
    lastNode->next = newNode;
    newNode->prev = lastNode;
}
// Function to delete a node at a specific position
void deleteNodeAtPosition(Node** head, int position) {
    if (*head == NULL) {
        cout << "List is empty" << endl;
        return;
    }
    if (position == 0) {
        Node* temp = *head;
        *head = (*head)->next;

        if (*head != NULL) {
            (*head)->prev = NULL;
        }
        delete temp;
        return;
    }
    Node* temp = *head;
```

```
    int count = 0;
    while (temp->next != NULL) {
        if (count == position - 1) {
            Node* nodeToDelete = temp->next;
            temp->next = temp->next->next;
            if (temp->next != NULL) {
                temp->next->prev = temp;
            }
            delete nodeToDelete;
            return;
        }
        temp = temp->next;
        count++;
    }
    cout << "Position " << position << " not found" << endl;
}
// Function to print the list
void printList(Node* head) {
    while (head != NULL) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    insertNode(&head, 10);
    insertNode(&head, 20);
    insertNode(&head, 30);
```

```
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Original List: ";
    printList(head);
    deleteNodeAtPosition(&head, 2);
    cout << "List after deleting node at position 2: ";
    printList(head);
    return 0;
}
```

```
Original List: 10 20 30 40 50
List after deleting node at position 2: 10 20 40 50
```

5-Forward and reverse traversal functions

Code:

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
    Node* prev;
};
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = newNode->prev = NULL;
    return newNode;
}
```

```
// Function to insert a new node at the end
void insertNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* lastNode = *head;
    while (lastNode->next != NULL) {
        lastNode = lastNode->next;
    }
    lastNode->next = newNode;
    newNode->prev = lastNode;
}

// Function to delete a node at a specific position
void deleteNodeAtPosition(Node** head, int position) {
    if (*head == NULL) {
        cout << "List is empty" << endl;
        return;
    }
    if (position == 0) {
        Node* temp = *head;
        *head = (*head)->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
```

```
        delete temp;
        return;
    }
    Node* temp = *head;
    int count = 0;
    while (temp->next != NULL) {
        if (count == position - 1) {
            Node* nodeToDelete = temp->next;
            temp->next = temp->next->next;

            if (temp->next != NULL) {
                temp->next->prev = temp;
            }

            delete nodeToDelete;
            return;
        }
        temp = temp->next;
        count++;
    }

    cout << "Position " << position << " not found" << endl;
}

// Function to print the list in forward order
void printForward(Node* head) {
    while (head != NULL) {
        cout << head->data << " ";
        head = head->next;
```

```
    }
    cout << endl;
}
// Function to print the list in reverse order
void printReverse(Node* head) {
    if (head == NULL) {
        cout << "List is empty" << endl;
        return;
    }
    Node* lastNode = head;
    while (lastNode->next != NULL) {
        lastNode = lastNode->next;
    }
    while (lastNode != NULL) {
        cout << lastNode->data << " ";
        lastNode = lastNode->prev;
    }
    cout << endl;
}
int main() {
    Node* head = NULL;
    insertNode(&head, 10);
    insertNode(&head, 20);
    insertNode(&head, 30);
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Original List (Forward): ";
    printForward(head);
    cout << "Original List (Reverse): ";
```

```
    printReverse(head);
    deleteNodeAtPosition(&head, 2);
    cout << "List after deleting node at position 2 (Forward): ";
    printForward(head);
    cout << "List after deleting node at position 2 (Reverse): ";
    printReverse(head);
    return 0;
}
```

Output:

```
Original List (Forward): 10 20 30 40 50
Original List (Reverse): 50 40 30 20 10
List after deleting node at position 2 (Forward): 10 20 40 50
List after deleting node at position 2 (Reverse): 50 40 20 10
```

Circular Linked List

1-Deleting the First Node in a Circular Linked List

Code:

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
// Function to insert a new node at the end
void insertNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        (*head)->next = *head;
        return;
    }
    Node* lastNode = *head;
    while (lastNode->next != *head) {
```

```
        lastNode = lastNode->next;
    }
    lastNode->next = newNode;
    newNode->next = *head;
}
// Function to delete the first node
void deleteFirstNode(Node** head) {
    if (*head == NULL) {
        cout << "List is empty" << endl;
        return;
    }
    if ((*head)->next == *head) {
        delete *head;
        *head = NULL;
        return;
    }
    Node* temp = *head;
    Node* lastNode = *head;
    while (lastNode->next != *head) {
        lastNode = lastNode->next;
    }
    lastNode->next = (*head)->next;
    *head = (*head)->next;
    delete temp;
}
// Function to print the list
void printList(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;

    insertNode(&head, 10);
    insertNode(&head, 20);
    insertNode(&head, 30);
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Original List: ";
    printList(head);
    deleteFirstNode(&head);
    cout << "List after deleting first node: ";
    printList(head);
    return 0;
}
```

Output:

```
Original List: 10 20 30 40 50
List after deleting first node: 20 30 40 50
```

2-Delete the last node in a circular linked list

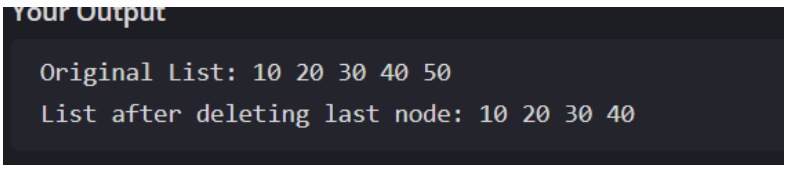
Code:

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
// Function to insert a new node at the end
void insertNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        (*head)->next = *head;
        return;
    }
    Node* lastNode = *head;
    while (lastNode->next != *head) {
        lastNode = lastNode->next;
    }
    lastNode->next = newNode;
    newNode->next = *head;
}
```

```
// Function to delete the last node
void deleteLastNode(Node** head) {
    if (*head == NULL) {
        cout << "List is empty" << endl;
        return;
    }

    if ((*head)->next == *head) {
        delete *head;
        *head = NULL;
        return;
    }
    Node* lastNode = *head;
    Node* secondLastNode = *head;
    while (lastNode->next != *head) {
        secondLastNode = lastNode;
        lastNode = lastNode->next;
    }
    secondLastNode->next = *head;
    delete lastNode;
}
// Function to print the list
void printList(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;
    insertNode(&head, 10);
    insertNode(&head, 20);
    insertNode(&head, 30);
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Original List: ";
    printList(head);
    deleteLastNode(&head);
    cout << "List after deleting last node: ";
    printList(head);
    return 0;
}
```

Output:



3-Deletes a node by its value in a circular linked list:

Code:

```
#include <iostream>
using namespace std;
// Node structure
```

```
struct Node {
   int data;
   Node* next;
};

// Function to create a new node
Node* createNode(int data) {
   Node* newNode = new Node();
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

// Function to insert a new node at the end
void insertNode(Node** head, int data) {
   Node* newNode = createNode(data);
   if (*head == NULL) {
      *head = newNode;
      (*head)->next = *head;
      return;
   }
   Node* lastNode = *head;
   while (lastNode->next != *head) {
      lastNode = lastNode->next;
   }
   lastNode->next = newNode;
   newNode->next = *head;
}
// Function to delete a node by its value
```

```
void deleteNodeByValue(Node** head, int value) {
    if (*head == NULL) {
        cout << "List is empty" << endl;
        return;
    }
    // Check if the node to be deleted is the head node
    if ((*head)->data == value) {
        if ((*head)->next == *head) {
            delete *head;
            *head = NULL;
            return;
        }
        Node* lastNode = *head;
        while (lastNode->next != *head) {
            lastNode = lastNode->next;
        }
        lastNode->next = (*head)->next;
        *head = (*head)->next;
        delete lastNode->next;
        return;
    }
    Node* temp = *head;
    while (temp->next != *head) {
        if (temp->next->data == value) {
            Node* nodeToDelete = temp->next;
            temp->next = temp->next->next;
            delete nodeToDelete;
            return;
        }
```

```
        temp = temp->next;
    }
    cout << "Node with value " << value << " not found" << endl;
}
// Function to print the list
void printList(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;
    insertNode(&head, 10);
    insertNode(&head, 20);
    insertNode(&head, 30);
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Original List: ";
    printList(head);
    deleteNodeByValue(&head, 30);
    cout << "List after deleting node with value 30: ";
    printList(head);
```

```
    return 0;
}
```

Output:

```
Original List: 10 20 30 40 50
List after deleting node with value 30: 10 20 40 50
```

4-Delete a node at a specific position in a circular linked list

Code:

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new node at the end
void insertNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        (*head)->next = *head;
```

```
        return;
    }
    Node* lastNode = *head;
    while (lastNode->next != *head) {
        lastNode = lastNode->next;
    }
    lastNode->next = newNode;
    newNode->next = *head;
}
// Function to delete a node at a specific position
void deleteNodeAtPosition(Node** head, int position) {
    if (*head == NULL) {
        cout << "List is empty" << endl;
        return;
    }

    int length = 1;
    Node* temp = *head;
    while (temp->next != *head) {
        temp = temp->next;
        length++;
    }
    if (position < 0 || position >= length) {
        cout << "Invalid position" << endl;
        return;
    }
    if (position == 0) {
        if (length == 1) {
            delete *head;
```

```
            *head = NULL;
            return;
        }
        Node* lastNode = *head;
        while (lastNode->next != *head) {
            lastNode = lastNode->next;
        }
        lastNode->next = (*head)->next;
        *head = (*head)->next;
        delete lastNode->next;
        return;
    }
    Node* temp2 = *head;
    for (int i = 0; i < position - 1; i++) {
        temp2 = temp2->next;
    }
    Node* nodeToDelete = temp2->next;
    temp2->next = temp2->next->next;
    delete nodeToDelete;
}
// Function to print the list
void printList(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;
    insertNode(&head, 10);
    insertNode(&head, 20);
    insertNode(&head, 30);
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Original List: ";
    printList(head);
    deleteNodeAtPosition(&head, 2);
    cout << "List after deleting node at position 2: ";
    printList(head);
    return 0;
}
```

Output:

```
Original List: 10 20 30 40 50
List after deleting node at position 2: 10 20 40 50
```

5-Forward traversal after deleting a node in a circular linked list:

Code:

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
```

```
    int data;
    Node* next;
};
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
// Function to insert a new node at the end
void insertNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        (*head)->next = *head;
        return;
    }
    Node* lastNode = *head;
    while (lastNode->next != *head) {
        lastNode = lastNode->next;
    }
    lastNode->next = newNode;
    newNode->next = *head;
}

// Function to delete a node by its value
void deleteNodeByValue(Node** head, int value) {
    if (*head == NULL) {
```

```
    cout << "List is empty" << endl;
    return;
}
// Check if the node to be deleted is the head node
if ((*head)->data == value) {
    if ((*head)->next == *head) {
        delete *head;
        *head = NULL;
        return;
    }
    Node* lastNode = *head;
    while (lastNode->next != *head) {
        lastNode = lastNode->next;
    }
    lastNode->next = (*head)->next;
    *head = (*head)->next;
    delete lastNode->next;
    return;
}
Node* temp = *head;
while (temp->next != *head) {
    if (temp->next->data == value) {
        Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
        delete nodeToDelete;
        return;
    }
    temp = temp->next;
}
```

```
    cout << "Node with value " << value << " not found" << endl;
}
// Function to print the list in forward order
void printForward(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;
    insertNode(&head, 10);
    insertNode(&head, 20);
    insertNode(&head, 30);
    insertNode(&head, 40);
    insertNode(&head, 50);
    cout << "Original List: ";
    printForward(head);
    deleteNodeByValue(&head, 30);
    cout << "List after deleting node with value 30: ";
    printForward(head);
    return 0;
}
```

Output:

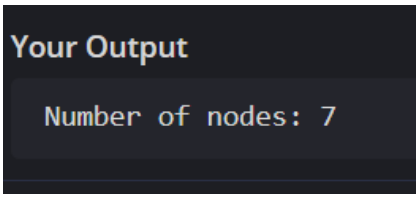
```
Original List: 10 20 30 40 50
List after deleting node with value 30: 10 20 40 50
```

Binary Search Tree

1-Counting all nodes in a Binary Search Tree

Code:

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

struct Node {
   int data;
   Node* left;
   Node* right;
   Node(int value) : data(value), left(NULL), right(NULL) {}
};

Node* insert(Node* root, int value) {
   if (root == NULL) {
      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 == NULL) return 0;
   return 1 + countNodes(root->left) + countNodes(root->right);
}

int main() {
   Node* root = NULL;
   root = insert(root, 50);
   root = insert(root, 30);
   root = insert(root, 20);
   root = insert(root, 40);
   root = insert(root, 70);
   root = insert(root, 60);
   root = insert(root, 80);

   cout << "Number of nodes: " << countNodes(root) << endl;
   return 0;
}
```

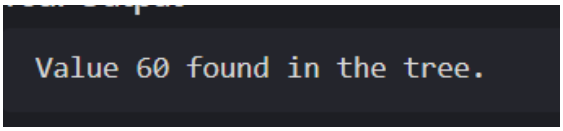


2. Searching for a specific value in a Binary Search Tree

Code:

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

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

bool search(Node* root, int value) {
   if (root == NULL) return false;
   if (root->data == value) return true;
   if (value < root->data) return search(root->left, value);
   return search(root->right, value);
}
int main() {
   Node* root = NULL;
   root = insert(root, 50);
   root = insert(root, 30);
   root = insert(root, 20);
   root = insert(root, 40);
   root = insert(root, 70);
   root = insert(root, 60);
   root = insert(root, 80);
```

```
    int value = 60;
    if (search(root, value)) {
        cout << "Value " << value << " found in the tree." << endl;
    } else {
        cout << "Value " << value << " not found in the tree." << endl;
    }
    return 0;
}
```



3. Traversing a BST in in-order, pre-order, and post-order

Code:

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

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

void inorder(Node* root) {
    if (root != NULL) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}
```

```
void preorder(Node* root) {
    if (root != NULL) {
        cout << root->data << " ";
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        cout << root->data << " ";
    }
}

int main() {
    Node* root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

    cout << "In-order traversal: ";
    inorder(root);
    cout << endl;

    cout << "Pre-order traversal: ";
    preorder(root);
    cout << endl;

    cout << "Post-order traversal: ";
    postorder(root);
    cout << endl;

    return 0;
}
```

```
In-order traversal: 20 30 40 50 60 70 80
Pre-order traversal: 50 30 20 40 70 60 80
Post-order traversal: 20 40 30 60 80 70 50
```

4- Reverse In-order Traversal for a Binary Search Tree

Code:

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

struct Node {
   int data;
   Node* left;
   Node* right;
   Node(int value) : data(value), left(NULL), right(NULL) {}
};

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

void reverseInorder(Node* root) {
   if (root != NULL) {
      reverseInorder(root->right);
      cout << root->data << " ";
      reverseInorder(root->left);
   }
}

int main() {
   Node* root = NULL;
   root = insert(root, 50);
   root = insert(root, 30);
   root = insert(root, 20);
   root = insert(root, 40);
   root = insert(root, 70);
   root = insert(root, 60);
   root = insert(root, 80);

   cout << "Reverse In-order traversal: ";
   reverseInorder(root);
   cout << endl;

   return 0;
```

```
}
```

```
Reverse In-order traversal: 80 70 60 50 40 30 20
```

5. Checking for duplicate values in a Binary Search Tree

Code:

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

Node* insert(Node* root, int value) {
    if (root == NULL) {
        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;
}

bool hasDuplicates(Node* root, int value) {
    if (root == NULL) return false;
    if (root->data == value) return true;
    if (value < root->data) return hasDuplicates(root->left, value);
    return hasDuplicates(root->right, value);
}

bool insertAndCheck(Node*& root, int value) {
    if (hasDuplicates(root, value)) {
        return true;
    }
    root = insert(root, value);
    return false;
}

int main() {
    Node* root = NULL;
```

```
    if (insertAndCheck(root, 50)) {
        cout << "Duplicate value found!" << endl;
    } else {
        cout << "Value inserted successfully." << endl;
    }
    if (insertAndCheck(root, 30)) {
        cout << "Duplicate value found!" << endl;
    } else {
        cout << "Value inserted successfully." << endl;
    }
    if (insertAndCheck(root, 50)) {
        cout << "Duplicate value found!" << endl;
    } else {
        cout << "Value inserted successfully." << endl;
    }

    return 0;
}
```

```
Value inserted successfully.
Value inserted successfully.
Duplicate value found!
```

6. Deleting a node from a Binary Search Tree

Code:

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

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int value) : data(value), left(NULL), right(NULL) {}
};

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

```
    }
    return root;
}

Node* minValueNode(Node* root) {
    Node* current = root;
    while (current && current->left != NULL) {
        current = current->left;
    }
    return current;
}

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 = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

int main() {
    Node* root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);
```

```
    cout << "Deleting node 20." << endl;
    root = deleteNode(root, 20);

    cout << "Deleting node 30." << endl;
    root = deleteNode(root, 30);

    cout << "Deleting node 50." << endl;
    root = deleteNode(root, 50);

    return 0;
}
```

```
Deleting node 20.
Deleting node 30.
Deleting node 50.
```