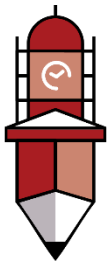

The University of Faisalabad

FINAL ASSIGNMENT

SUBMITTED TO: MISS IRSHA QURESHI

SUBMITTED BY: MISHAL NADEEM

REGISTRATION NO: 2023-BS-AI-020

DEPARTMENT: COMPUTER SCIENCE

DATA STRUCTURES

Doubly Linked List

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

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node* prev;};
void deleteFirst(Node*& head) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return; }
    Node* temp = head;
    head = head->next;
    if (head != nullptr) {
        head->prev = nullptr;   }
    delete temp;
    cout << "Pehla node delete ho gaya." << endl;}
void appendNode(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr, nullptr};
    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;
    deleteFirst(head);  // First node delete karne ka function call
    return 0;}
```

Output

```
List khali hai.
```

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

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node* prev;};
void deleteLast(Node*& head) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return;  }
```

```
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;    }
    if (temp->prev != nullptr) {
        temp->prev->next = nullptr;
    } else {
        head = nullptr;}  // Agar list mein sirf ek node tha
    delete temp;
    cout << "Aakhri node delete ho gaya." << endl;}
void appendNode(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr, nullptr};
    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;
   appendNode(head, 10);
    appendNode(head, 20);
    deleteLast(head);  // Aakhri node delete karne ka function call
    return 0;}
```

Output

Aakhri node delete ho gaya.

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

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node* prev;};
void deleteByValue(Node*& head, int value) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return;    }
    Node* temp = head;
    while (temp != nullptr && temp->data != value) {
        temp = temp->next;  }
    if (temp == nullptr) {
        cout << "Node nahi mila." << endl;
        return;    }
    if (temp->prev != nullptr) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next;    }
```

```
    if (temp->next != nullptr) {
        temp->next->prev = temp->prev;    }
    delete temp;
    cout << "Node with value " << value << " delete ho gaya." << endl;}
void appendNode(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr, nullptr};
    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;
        appendNode(head, 7);
    appendNode(head, 45);
    appendNode(head, 10);
    appendNode(head, 40);
    deleteByValue(head, 10);  // Node with value 10 ko delete karne ka function call
    return 0;}
```

Output

Node with value 10 delete ho gaya.

How would you delete a node at a specific position in a doubly linked list?

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node* prev;};
void deleteAtPosition(Node*& head, int position) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return; }
    Node* temp = head;
    int count = 1;
    while (temp != nullptr && count < position) {
        temp = temp->next;
        count++;    }
    if (temp == nullptr) {
        cout << "Position galat hai." << endl;
        return;    }
    if (temp->prev != nullptr) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next;    }
```

```
    if (temp->next != nullptr) {
        temp->next->prev = temp->prev;    }
    delete temp;
    cout << "Position " << position << " par node delete ho gaya." << endl;}
void appendNode(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr, nullptr};
    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;
   appendNode(head, 10);
    appendNode(head, 20);
    appendNode(head, 30);
   deleteAtPosition(head, 2);  // Position 2 par node ko delete karne ka function call
    return 0;}
```

Output

Position 2 par node delete ho gaya.

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

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node* prev;};
// Forward traversal function
void forwardTraversal(Node* head) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return; }
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next; }
    cout << endl;}
void reverseTraversal(Node* head) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return;  }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;    }
```

```
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->prev;  }
    cout << endl;}
void appendNode(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr, nullptr};
    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;
    appendNode(head, 10);
    appendNode(head, 20);
    appendNode(head, 30);    forwardTraversal(head);   // Forward traversal function call
    reverseTraversal(head);   // Reverse traversal function call
    return 0;}
```

Output

```
Forward traversal: 10 20 30
Reverse traversal: 30 20 10
```

Circular Linked List

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

```
#include <iostream>
using namespace std;
// Node ka structure banayen
struct Node {
    int data;
    Node* next;};
// Function to delete the first node in a circular linked list
void deleteFirst(Node*& head) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return;    }
    if (head->next == head) {
        delete head;  // Agar sirf ek node hai
        head = nullptr;
        cout << "Pehla node delete ho gaya." << endl;
        return;    }
    Node* temp = head;
    // Traverse to the last node
    while (temp->next != head) {
        temp = temp->next;    }
    temp->next = head->next;
```

```
    delete head;
    head = temp->next;
    cout << "Pehla node delete ho gaya." << endl;}
// Function to add a node at the end
void appendNode(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr, nullptr};
    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;
  appendNode(head, 10);
    appendNode(head, 20);
    appendNode(head, 30);
    deleteFirst(head);  // First node delete karne ka function call
    return 0;}
```

Output

Pehla node delete ho gaya.

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 the last node in a circular linked list
void deleteLast(Node*& head) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return;    }
    if (head->next == head) {
        delete head;
        head = nullptr;
        cout << "Aakhri node delete ho gaya." << endl;
        return;    }
    Node* temp = head;
    while (temp->next->next != head) {
        temp = temp->next;    }
    delete temp->next;
    temp->next = head;
    cout << "Aakhri node delete ho gaya." << endl;}
```

```
// Function to add a node at the end
void appendNode(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr, nullptr};
    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;
   appendNode(head, 67);
    appendNode(head, 27);
    appendNode(head, 33);
    deleteLast(head);  // Aakhri node delete karne ka function call
    return 0;}
```

Output

Aakhri node delete ho gaya.

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

```
#include <iostream>
```

```
using namespace std;
struct Node {
    int data;
    Node* next;};
// Function to delete a node by its value in a circular linked list
void deleteByValue(Node*& head, int value) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return;    }
    Node* temp = head;
    Node* prev = nullptr;
    do {
        if (temp->data == value) {
            if (prev == nullptr) {  // Agar head node hai
                if (temp->next == head) {  // Agar ek hi node hai
                    delete temp;
                    head = nullptr;
                } else {
                    prev = head;
                    while (prev->next != head) {
                        prev = prev->next;  }
                    prev->next = head->next;
                    delete head;
                    head = prev->next; }
            } else {
```

```
            prev->next = temp->next;
            delete temp;            }
        cout << "Node with value " << value << " delete ho gaya." << endl;
        return;        }
    prev = temp;
    temp = temp->next;
  } while (temp != head);
  cout << "Node nahi mila." << endl;}
// Function to add a node at the end
void appendNode(Node*& head, int value) {
  Node* newNode = new Node{value, nullptr, nullptr};
  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;
  appendNode(head, 22);
  appendNode(head, 78);
  appendNode(head, 97);
  deleteByValue(head, 10);  // Node with value 10 ko delete karne ka function call
```

```
    return 0;}
```

Output

```
Node nahi mila.
```

How will you delete a node at a specific position in a circular linked list?

```
#include <iostream>
using namespace std;
// Node ka structure banayen
struct Node {
    int data;
    Node* next;};
// Function to delete a node at a specific position in a circular linked list
void deleteAtPosition(Node*& head, int position) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return;    }
    if (position == 1) {
        deleteFirst(head);
        return;   }
    Node* temp = head;
    Node* prev = nullptr;
    int count = 1;
```

```
    do {
        if (count == position) {
            if (prev != nullptr) {
                prev->next = temp->next;      }
            delete temp;
            cout << "Position " << position << " par node delete ho gaya." << endl;
            return;     }
        prev = temp;
        temp = temp->next;
        count++;
    } while (temp != head);
    cout << "Position galat hai." << endl;}
// Function to add a node at the end
void appendNode(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr, nullptr};
    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;
```

```
    appendNode(head, 10);
    appendNode(head, 20);
    appendNode(head, 30);
    deleteAtPosition(head, 2);  // Position 2 par node ko delete karne ka function call
    return 0;}
```

Output

```
Position 2 par node delete ho gaya.
```

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

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;};
// Forward traversal function for a circular linked list
void forwardTraversal(Node* head) {
    if (head == nullptr) {
        cout << "List khali hai." << endl;
        return;    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
```

```
        temp = temp->next;
    } while (temp != head);
    cout << endl;}
// Function to add a node at the end
void appendNode(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr, nullptr};
    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;
   appendNode(head, 45);
    appendNode(head, 31);
    appendNode(head, 98);    forwardTraversal(head);  // Forward traversal function call
    return 0;}
```

Output

45 31 98

Binary Search Tree

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;};

int countNodes(Node* root) {

    if (root == nullptr) {

        return 0;    }

    return 1 + countNodes(root->left) + countNodes(root->right);  // Node count karne ka formula}

// Function to insert a node in the BST

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

    if (root == nullptr) {

        return new Node(value);}  // Agar tree khali hai, to naya node bana kar return karain

    if (value < root->data) {

        root->left = insert(root->left, value);  // Left subtree mein insert karain

    } else {

        root->right = insert(root->right, value); } // Right subtree mein insert karain}


    return root; } // Root ko return karain, jise recursive calls ke dauran update hota hai
```

```
int main() {
    Node* root = nullptr;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 5);
    root = insert(root, 15);
    root = insert(root, 30);
    cout << "Total nodes: " << countNodes(root) << endl;
    return 0;}
```

Output

```
Total nodes: 5
```

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

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;};
Node* search(Node* root, int value) {
    if (root == nullptr || root->data == value) {
        return root;  // Agar value mil gayi ya tree khali hai  }
```

```
    if (value < root->data) {
        return search(root->left, value); } // Left subtree mein search karain
    return search(root->right, value); } // Right subtree mein search karain
// Function to insert a node in the BST
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);}  // Agar tree khali hai, to naya node bana kar return
    if (value < root->data) {
        root->left = insert(root->left, value);  // Left subtree mein insert karain
    } else {
        root->right = insert(root->right, value);  // Right subtree mein insert karain  }
    return root; } // Root ko return karain, jise recursive calls ke dauran update hota hai
int main() {
    Node* root = nullptr;
   root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 5);
    root = insert(root, 15);
    root = insert(root, 30);
    int value = 20;
    Node* result = search(root, value);
    if (result != nullptr) {
        cout << "Value " << value << " tree mein mil gaya." << endl;
    } else {
        cout << "Value " << value << " tree mein nahi mila." << endl; }
```

```
    return 0;}
```

Output

Value 20 tree mein mil gaya.

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

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;};
void inorder(Node* root) {
    if (root != nullptr) {
        inorder(root->left);  // Left subtree
        cout << root->data << " ";  // Root node
        inorder(root->right); }} // Right subtree
void preorder(Node* root) {
    if (root != nullptr) {
        cout << root->data << " ";  // Root node
        preorder(root->left);  // Left subtree
        preorder(root->right); }} // Right subtree
void postorder(Node* root) {
```

```
    if (root != nullptr) {
        postorder(root->left);  // Left subtree
        postorder(root->right);  // Right subtree
        cout << root->data << " "; }} // Root node
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value); } // Agar tree khali hai, to naya node bana kar return karai
    if (value < root->data) {
        root->left = insert(root->left, value);  // Left subtree mein insert karain
    } else {
        root->right = insert(root->right, value);}  // Right subtree mein insert karain
    return root;  }// Root ko return karain, jise recursive calls ke dauran update hota hai
int main() {
    Node* root = nullptr;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 5);
    root = insert(root, 15);
    root = insert(root, 30);
    cout << "In-order traversal: ";
    inorder(root);
    cout << "\nPre-order traversal: ";
    preorder(root);
    cout << "\nPost-order traversal: ";
    postorder(root);
```

```
    cout << endl;
    return 0;}
```

Output

```
In-order traversal: 5 10 15 20 30
Pre-order traversal: 10 5 20 15 30
Post-order traversal: 5 15 30 20 10
```

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

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;};
void reverseInorder(Node* root) {
    if (root != nullptr) {
        reverseInorder(root->right);  // Right subtree
        cout << root->data << " ";  // Root node
        reverseInorder(root->left);  }}// Left subtree
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);} // Agar tree khali hai, to naya node bana kar return karain
    if (value < root->data) {
```

```
        root->left = insert(root->left, value);  // Left subtree mein insert karain
    } else {
        root->right = insert(root->right, value); } // Right subtree mein insert karain
    return root;}  // Root ko return karain, jise recursive calls ke dauran update hota hai
int main() {
    Node* root = nullptr;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 5);
    root = insert(root, 15);
    root = insert(root, 30);
    cout << "Reverse in-order traversal: ";
    reverseInorder(root);
    cout << endl;
    return 0;}
```

Output

Reverse in-order traversal: 30 20 15 10 5

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

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

```
    int data;
    Node* left;
    Node* right;};
bool isDuplicate(Node* root, int value) {
    if (root == nullptr) {
        return false;    }
    if (root->data == value) {
        return true;}  // Agar value mil jaye to duplicate hai
    if (value < root->data) {
        return isDuplicate(root->left, value); } // Left subtree mein check karain
    return isDuplicate(root->right, value);}  // Right subtree mein check karain
// Function to insert a node in the BST
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);  }// Agar tree khali hai, to naya node bana kar return karai
    if (value < root->data) {
        root->left = insert(root->left, value);  // Left subtree mein insert karain
    } else {
        root->right = insert(root->right, value);}  // Right subtree mein insert karain
    return root;  }// Root ko return karain, jise recursive calls ke dauran update hota hai
int main() {
    Node* root = nullptr;
   root = insert(root, 98);
    root = insert(root, 57);
    root = insert(root, 52);
```

```
    root = insert(root, 15);
    root = insert(root, 30);
    int value = 20;
    if (isDuplicate(root, value)) {
        cout << "Value " << value << " duplicate hai." << endl;
    } else {
        cout << "Value " << value << " duplicate nahi hai." << endl;    }
    return 0;}
```

Output

Value 20 duplicate nahi hai.

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 data;
    Node* left;
    Node* right;};
// Function to find the minimum value node in a BST
Node* findMin(Node* root) {
    while (root->left != nullptr) {
        root = root->left;    }
```

```
    return root;}
Node* deleteNode(Node* root, int value) {
    if (root == nullptr) {
        return root; } // Agar node nahi milti    }
    if (value < root->data) {
        root->left = deleteNode(root->left, value);  // Left subtree mein search karain
    } else if (value > root->data) {
        root->right = deleteNode(root->right, value);  // Right subtree mein search karain
    } else {  // Agar node mil gayi
        // Case 1: Leaf node ko delete karna
        if (root->left == nullptr && root->right == nullptr) {
            delete root;
            root = nullptr;        }
        // Case 2: Node with one child
        else if (root->left == nullptr) {
            Node* temp = root;
            root = root->right;
            delete temp;        }
        else if (root->right == nullptr) {
            Node* temp = root;
            root = root->left;
            delete temp;        }
        // Case 3: Node with two children
        else {
            Node* temp = findMin(root->right);  // Inorder successor find karain
```

```
            root->data = temp->data;
            root->right=deleteNode(root->right, temp->data);// Successor node ko del karain} }
    return root;}
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);}// Agar tree khali hai, to naya node bana kar return karain
      if (value < root->data) {
        root->left = insert(root->left, value);  // Left subtree mein insert karain
    } else {
        root->right = insert(root->right, value); } // Right subtree mein insert karain
    return root;}  // Root ko return karain, jise recursive calls ke dauran update hota hai
int main() {
    Node* root = nullptr;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 5);
    root = insert(root, 15);
    root = insert(root, 30);
    int value = 20;
    root = deleteNode(root, value);  // Node ko delete karne ka function call
    return 0;}
```

Output

```
Node 20  deleted.
```