RAMANUJAN COLLEGE

UNIVERSITY OF DELHI

SOUTH CAMPUS IN KALKAJI 110019

PRACTICAL FILE

Data Structures

Submitted By: Yuvraj Singh

Examination Roll Number: 23020570056

Class Roll Number: 20231471

Submitted to: Dr. Sahil Pathak

Index

Acknowledgment

I would like to express my sincere gratitude to my instructor, Dr. Sahil Pathak, for providing me with the opportunity to work on this series of practical. Their invaluable guidance and support have been instrumental in understanding the core concepts of computer graphics.

I would also like to thank Ramanujan College, University of Delhi, for offering such a rich curriculum that fosters technical learning and hands-on experience. The practical applications of various algorithms and transformation techniques have deepened my knowledge in the field.

Furthermore, I would like to thank my peers for their encouragement and collaboration during the course of this project. Their inputs have enhanced my learning experience and contributed to a stimulating environment for problem-solving.

1. Write a program to implement singly linked list as an ADT that supports the following operations:
   i. Insert an element x at the beginning and at i^{th} position of the singly linked list
   ii. Remove an element from the beginning and from i^{th} position of the singly linked list
   iii. Search for an element x in the singly linked list and return its pointer

Code:

```cpp
#include <iostream>
using namespace std;
You, 6 minutes ago | 1 author (You)
class Node {
public:
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};
You, 6 minutes ago | 1 author (You)
class SinglyLinkedList {
private:
    Node* head;
public:
    SinglyLinkedList() : head(nullptr) {}
    void insertAtBeginning(int x) {
        Node* newNode = new Node(x);
        newNode->next = head;
        head = newNode;}
    void insertAtPosition(int x, int pos) {
        if (pos == 0) {
            insertAtBeginning(x);
            return;}
        Node* newNode = new Node(x);
        Node* current = head;
        int count = 0;
        while (current != nullptr && count < pos - 1) {
            current = current->next;
            count++;}
        if (current == nullptr) {
            throw out_of_range("Position out of bounds");}
        newNode->next = current->next;
        current->next = newNode;}
    void removeFromBeginning() {
        if (head == nullptr) {      You, 6 minutes ago • Uncommitted c
            throw out_of_range("List is empty");}
        Node* temp = head;
        head = head->next;
        delete temp;}
    void removeFromPosition(int pos) {
        if (head == nullptr) {
            throw out_of_range("List is empty");}
        if (pos == 0) {
            removeFromBeginning();
            return;}
```

```
        Node* current = head;
        int count = 0;
        while (current != nullptr && count < pos - 1) {
            current = current->next;
            count++;}
        if (current == nullptr || current->next == nullptr) {
            throw out_of_range("Position out of bounds");}
        Node* temp = current->next;
        current->next = temp->next;
        delete temp;}
    Node* search(int x) {
        Node* current = head;
        while (current != nullptr) {
            if (current->data == x) {
                return current;}
            current = current->next;}
        return nullptr;
    }
    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " -> ";
            current = current->next;}
        cout << "nullptr" << endl;}
};
int main() {
    SinglyLinkedList sll;
    sll.insertAtBeginning(10);
    sll.insertAtBeginning(20);
    sll.insertAtBeginning(20);
    sll.insertAtBeginning(30);
    sll.insertAtPosition(40, 1);
    sll.display();
    Node* node = sll.search(30);
    if (node) {
        cout << "Element found: " << node->data << endl;}
    else {
        cout << "Element not found" << endl;}
    sll.removeFromBeginning();
    sll.removeFromPosition(1);
    sll.display();
    return 0;}
```

Output:

```
30 -> 40 -> 20 -> 20 -> 10 -> nullptr
Element found: 30
40 -> 20 -> 10 -> nullptr
```

2. Write a program to implement doubly linked list as an ADT that supports the following operations:
   i. Insert an element x at the beginning and at the end of the doubly linked list
   ii. Remove an element from the beginning and from the end of the doubly linked list

02_Doubly_Linked_list.cpp > ...

You, 9 seconds ago | 1 author (You)

```cpp
#include <iostream>
using namespace std;        You, last week • Doubly linked list
```

You, 9 seconds ago | 1 author (You)

```cpp
class Node {
public:
    int data;
    Node* prev;
    Node* next;
    Node(int val) : data(val), prev(nullptr), next(nullptr) {}
};
```

You, 9 seconds ago | 1 author (You)

```cpp
class DoublyLinkedList {
private:
    Node* head;
    Node* tail;
public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}
    void insertAtBeginning(int x) {
        Node* newNode = new Node(x);
        if (head == nullptr) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }
    void insertAtEnd(int x) {
        Node* newNode = new Node(x);
        if (tail == nullptr) {
            head = tail = newNode;
        } else {
            newNode->prev = tail;
            tail->next = newNode;
            tail = newNode;
        }
    }
    void removeFromBeginning() {
        if (head == nullptr) {
            throw out_of_range("List is empty");
        }
```

```
39                head = tail = nullptr;}
40            else {
41                head = head->next;
42                head->prev = nullptr;}
43            delete temp;}
44        void removeFromEnd() {
45            if (tail == nullptr) {
46                throw out_of_range("List is empty");}
47            Node* temp = tail;
48            if (head == tail) {
49                head = tail = nullptr;} else {
50                tail = tail->prev;
51                tail->next = nullptr;}
52            delete temp;}
53        void display() {···
61    };
62    int main() {
63        DoublyLinkedList dll;
64        dll.insertAtBeginning(10);
65        dll.insertAtBeginning(20);
66        dll.insertAtEnd(30);
67        dll.display();
68        dll.removeFromBeginning();
69        dll.removeFromEnd();
70        dll.display();
71        return 0;
72    }
```

Output:

```
20 <-> 10 <-> 30 <-> nullptr
10 <-> nullptr
```

3. Write a program to implement circular linked list as an ADT which supports the following operations

   a. Insert an element x in the list
   b. Remove an element from the list
   c. Search for an element x in the list and return its pointer

03_circular_linked_list.cpp > CircularLinkedList > insert(int)

You, 5 seconds ago | 1 author (You)

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

You, 3 seconds ago | 1 author (You)

```cpp
class Node {
public:
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};
```

You, 1 second ago | 1 author (You)

```cpp
class CircularLinkedList {
private:
    Node* last;
public:
    CircularLinkedList() : last(nullptr) {}
    void insert(int x) {
        Node* newNode = new Node(x);
        if (last == nullptr) {
            last = newNode;
            last->next = last;
        } else {
            newNode->next = last->next;
            last->next = newNode;
            last = newNode;}      You, 1 second ago • Uncommitted changes
    }
    void remove(int x) {
        if (last == nullptr) {
            throw out_of_range("List is empty");}
        Node* current = last->next;
        Node* previous = last;
        do {
            if (current->data == x) {
                if (current == last) {
                    if (last->next == last) {
                        delete last;
                        last = nullptr;
                    } else {
                        previous->next = current->next;
                        last = previous;
                        delete current;}
                } else {
                    previous->next = current->next;
                    delete current;}
                return;}
```

```
43                previous = current;
44                current = current->next;
45            } while (current != last->next);
46
47            throw invalid_argument("Element not found");
48        }
49        Node* search(int x) {
50            if (last == nullptr) return nullptr;
51            Node* current = last->next;
52            do {
53                if (current->data == x) {
54                    return current;}
55                current = current->next;
56            } while (current != last->next);
57            return nullptr;}
58        void display() {
59            if (last == nullptr) {
60                cout << "List is empty" << endl;
61                return;}
62            Node* current = last->next;
63            do {
64                cout << current->data << " -> ";
65                current = current->next;
66            } while (current != last->next);
67            cout << " (back to start)" << endl;}
68    };
69    int main() {
70        CircularLinkedList cll;
71        cll.insert(10);
72        cll.insert(20);
73        cll.insert(30);
74        cll.display();
75        Node* node = cll.search(20);
76        if (node) {
77            cout << "Element found: " << node->data << endl;
78        } else {
79            cout << "Element not found" << endl;}
80        cll.remove(20);
81        cll.display();
82        return 0;
83    }
```

Output:

```
10 -> 20 -> 30 ->  (back to start)
Element found: 20
10 -> 30 ->  (back to start)
```

4. Implement Stack as an ADT and use it to evaluate a prefix/postfix expression.

04_Stack.cpp > main()

You, 1 second ago | 1 author (You)

```cpp
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
You, 1 second ago | 1 author (You)
class Node {
public:
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;}
};
You, 1 second ago | 1 author (You)
class Stack {
public:
    Node* head = nullptr;
    void push(int value) {
        Node* newNode = new Node(value);
        newNode->next = head;
        head = newNode;}
    void pop() {
        if (head == nullptr) {
            cout << "Stack is empty" << endl;
            return;}
        Node* temp = head;
        head = head->next;
        delete temp;}
    int top() {
        if (head == nullptr) {
            cout << "Stack is empty" << endl;
            return -1;
        } else {
            return head->data;}}
    bool isEmpty() {
        return head == nullptr;}
    void display() {
        if (head == nullptr) {
            cout << "Stack is empty" << endl;
            return;}
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;}
        cout << endl;}
};
int evaluatePostfix(const string& expression) {
    Stack stack;
    istringstream iss(expression);
    string token;
```

```
    while (iss >> token) {
        if (isdigit(token[0])) {
            stack.push(stoi(token));
        } else {
            int b = stack.top(); stack.pop();
            int a = stack.top(); stack.pop();
            switch (token[0]) {
                case '+': stack.push(a + b); break;
                case '-': stack.push(a - b); break;
                case '*': stack.push(a * b); break;
                case '/': stack.push(a / b); break;
            }}}
    return stack.top();}
int evaluatePrefix(const string& expression) {
    Stack stack;
    istringstream iss(expression);
    vector<string> tokens;
    string token;
    while (iss >> token) {
        tokens.push_back(token);}
    reverse(tokens.begin(), tokens.end());
    for (const auto& tok : tokens) {
        if (isdigit(tok[0])) {
            stack.push(stoi(tok));
        } else {
            int a = stack.top(); stack.pop();
            int b = stack.top(); stack.pop();

            switch (tok[0]) {
                case '+': stack.push(a + b); break;
                case '-': stack.push(a - b); break;
                case '*': stack.push(a * b); break;
                case '/': stack.push(a / b); break;}}}
    return stack.top();
}
int main() {
    string postfix = "5 6 + 4 *";
    string prefix = "* + 5 6 4";
    cout << "Postfix Evaluation: " << evaluatePostfix(postfix) << endl;
    cout << "Prefix Evaluation: " << evaluatePrefix(prefix) << endl;
    return 0;
}
```

Output:

```
Postfix Evaluation: 44
Prefix Evaluation: 44
```

5. Implement Queue as an ADT.

Code:

05_Queue.cpp > Queue > Queue()

```
You, 3 days ago | 1 author (You)
#include <iostream>
using namespace std;
You, 3 days ago | 1 author (You)
class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }
};
You, 3 days ago | 1 author (You)
class Queue {
private:
    Node* front;
    Node* rear;
public:
    Queue() {
        front = nullptr;
        rear = nullptr;
    }       You, 3 days ago • Added BST and AVL
    void Enqueue(int value) {
        Node* newNode = new Node(value);
        if (rear == nullptr) {
            front = rear = newNode;
            return;
        }
        rear->next = newNode;
        rear = newNode;
    }
    void Dequeue() {
        if (front == nullptr) {
            cout << "Queue is empty" << endl;
            return;
        }
        Node* temp = front;
        front = front->next;

        if (front == nullptr) {
            rear = nullptr;
        }
```

```
43          delete temp;
44      }
45      void Front() {
46          if (front == nullptr) {
47              cout << "Queue is empty" << endl;
48          } else {
49              cout << "Front element: " << front->data << endl;
50          }
51      }
52      void Display() {
53          if (front == nullptr) {
54              cout << "Queue is empty" << endl;
55              return;
56          }
57          Node* temp = front;
58          while (temp != nullptr) {
59              cout << temp->data << " ";
60              temp = temp->next;
61          }
62          cout << endl;
63      }
64  };
65  int main() {
66      Queue queue;
67      queue.Enqueue(1);
68      queue.Enqueue(2);
69      queue.Enqueue(3);
70      queue.Display();
71      queue.Dequeue();
72      queue.Display();
73      queue.Front();
74      return 0;
75  }
```

Output:

```
1 2 3
2 3
Front element: 2
```

6. Write a program to implement Binary Search Tree as an ADT which supports the following operations:

   i. Insert an element x

   ii. Delete an element x

   iii. Search for an element x in the BST

   iv. Display the elements of the BST in preorder, inorder, and postorder traversal

Output:

```cpp
#include <iostream>
using namespace std;
You, 3 days ago | 1 author (You)
class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int value) {
        data = value;
        left = nullptr;
        right = nullptr;
    }
};
You, 3 days ago | 1 author (You)
class BST {
private:
    Node* root;
    Node* insert(Node* node, int value) {
        if (node == nullptr) {
            return new Node(value);
        }
        if (value < node->data) {
            node->left = insert(node->left, value);
        } else if (value > node->data) {
            node->right = insert(node->right, value);
        }
        return node;
    }
    Node* search(Node* node, int value) {
        if (node == nullptr || node->data == value) {
            return node;
        }
        if (value < node->data) {
            return search(node->left, value);
        }
        return search(node->right, value);
    }
    Node* deleteNode(Node* node, int value) {
        if (node == nullptr) {
            return node;
        }
        if (value < node->data) {
            node->left = deleteNode(node->left, value);
        } else if (value > node->data) {
            node->right = deleteNode(node->right, value);
        } else {
            if (node->left == nullptr) {
                Node* temp = node->right;
                delete node;
                return temp;
            } else if (node->right == nullptr) {
                Node* temp = node->left;
                delete node;
```

```
53                 return temp;
54                 }
55                 Node* temp = minValueNode(node->right);
56                 node->data = temp->data;
57                 node->right = deleteNode(node->right, temp->data);
58             }
59             return node;
60         }
61         Node* minValueNode(Node* node) {
62             Node* current = node;
63             while (current && current->left != nullptr) {
64                 current = current->left;
65             }
66             return current;
67         }
68         void inorder(Node* node) {
69             if (node == nullptr) {
70                 return;
71             }
72             inorder(node->left);
73             cout << node->data << " ";
74             inorder(node->right);
75         }
76         void preorder(Node* node) {
77             if (node == nullptr) {
78                 return;
79             }
80             cout << node->data << " ";
81             preorder(node->left);
82             preorder(node->right);
83         }
84         void postorder(Node* node) {
85             if (node == nullptr) {
86                 return;
87             }
88             postorder(node->left);
89             postorder(node->right);
90             cout << node->data << " ";
91         }
92     public:
93         BST() {
94             root = nullptr;
95         }
96         void insert(int value) {
97             root = insert(root, value);
98         }
99         void deleteNode(int value) {
100            root = deleteNode(root, value);
101        }
```

```
102     bool search(int value) {
103         return search(root, value) != nullptr;
104     }
105     void inorder() {
106         inorder(root);
107         cout << endl;
108     }
109     void preorder() {
110         preorder(root);
111         cout << endl;
112     }
113     void postorder() {
114         postorder(root);
115         cout << endl;
116     }
117 };
118 
119 int main() {
120     BST bst;
121     bst.insert(50);
122     bst.insert(30);
123     bst.insert(20);
124     bst.insert(40);
125     bst.insert(70);
126     bst.insert(60);
127     bst.insert(80);
128     cout << "Inorder traversal: ";
129     bst.inorder();
130     cout << "Preorder traversal: ";
131     bst.preorder();
132     cout << "Postorder traversal: ";
133     bst.postorder();
134     cout << "Searching for 40: " << (bst.search(40) ? "Found" : "Not Found") << endl;
135     bst.deleteNode(20);
136     cout << "Inorder traversal after deleting 20: ";
137     bst.inorder();
138     return 0;
139 }
```

Output:

```
Inorder traversal: 20 30 40 50 60 70 80
Preorder traversal: 50 30 20 40 70 60 80
Postorder traversal: 20 40 30 60 80 70 50
Searching for 40: Found
Inorder traversal after deleting 20: 30 40 50 60 70 80
```

7. Write a program to implement insert and search operation in AVL trees.

Code:

07_AVL.cpp > AVLTree > leftRotate(Node *)

You, 1 second ago | 1 author (You)

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

You, 1 second ago | 1 author (You)

```cpp
class Node {
public:
    int data;
    Node* left;
    Node* right;
    int height;

    Node(int value) {
        data = value;
        left = nullptr;
        right = nullptr;
        height = 1; }
};
```

You, 1 second ago | 1 author (You)

```cpp
class AVLTree {
private:
    Node* root;
    int height(Node* node) {
        return node ? node->height : 0;}
    int getBalance(Node* node) {
        return node ? height(node->left) - height(node->right) : 0;}
    Node* rightRotate(Node* y) {
        Node* x = y->left;
        Node* T2 = x->right;
        x->right = y;
        y->left = T2;
        y->height = max(height(y->left), height(y->right)) + 1;
        x->height = max(height(x->left), height(x->right)) + 1;
        return x;}
    Node* leftRotate(Node* x) {
        Node* y = x->right;
        Node* T2 = y->left;
        y->left = x;       You, 3 days ago • Added BST and AVL
        x->right = T2;
        x->height = max(height(x->left), height(x->right)) + 1;
        y->height = max(height(y->left), height(y->right)) + 1;
        return y;}
    Node* insert(Node* node, int value) {
        if (!node) {
            return new Node(value);}
        if (value < node->data) {
            node->left = insert(node->left, value);
        } else if (value > node->data) {
            node->right = insert(node->right, value);
        } else {
            return node;}
        node->height = 1 + max(height(node->left), height(node->right));
        int balance = getBalance(node);
        if (balance > 1 && value < node->left->data) {
            return rightRotate(node);}
        if (balance < -1 && value > node->right->data) {
```

```
51          return rightRotate(node);}
52      if (balance < -1 && value > node->right->data) {
53          return leftRotate(node);}
54      if (balance > 1 && value > node->left->data) {
55          node->left = leftRotate(node->left);
56          return rightRotate(node);}
57      if (balance < -1 && value < node->right->data) {
58          node->right = rightRotate(node->right);
59          return leftRotate(node);}
60      return node;}
61  Node* search(Node* node, int value) {
62      if (node == nullptr || node->data == value) {
63          return node;}
64      if (value < node->data) {
65          return search(node->left, value);}
66      return search(node->right, value);}
67  public:
68      AVLTree() {
69          root = nullptr;}
70      void insert(int value) {
71          root = insert(root, value);}
72      bool search(int value) {
73          return search(root, value) != nullptr;}
74      void inorderTraversal(Node* root) {
75          if (root == nullptr) {
76              return;}
77          inorderTraversal(root->left);
78          cout << root->data << " ";
79          inorderTraversal(root->right);}
80      void display() {
81          inorderTraversal(root);
82          cout << endl;}
83  };
84  int main() {
85      AVLTree avl;
86      avl.insert(10);
87      avl.insert(20);
88      avl.insert(30);
89      avl.insert(40);
90      avl.insert(50);
91      avl.insert(25);
92      cout << "Inorder traversal of the constructed AVL tree is: ";
93      avl.display();
94      cout << "Searching for 30: " << (avl.search(30) ? "Found" : "Not Found") << endl;
95      cout << "Searching for 60: " << (avl.search(60) ? "Found" : "Not Found") << endl;
96      return 0;
97  }
```

Output:

```
Inorder traversal of the constructed AVL tree is: 10 20 25 30 40 50
Searching for 30: Found
Searching for 60: Not Found
```