ATMA RAM SANATAN DHARMA COLLEGE

(UNIVERSITY OF DELHI)

DATA STRUCTURES

(PRACTICAL FILE)

SUBMITTED BY

NAME:**SUKET ROHILLA**

ROLL NO:**24/48069**

DATE:**26-10-2025**

SUBMITTED TO

Mr Jag Mohan Sir

1. Write a program to implement singly linked list as an ADT that supports the following operations:

- Insert an element x at the beginning of the singly linked list
- Insert an element x at ith position in the singly linked list
- Remove an element from the beginning of the doubly linked list
- Remove an element from ith position in the singly linked list.
- Search for an element x in the singly linked list and return its pointer

INPUT

main.cpp Share Run

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

class Node {
public:
    int data;
    Node *next;
};

class LinkedList {
private:
    Node *first;

public:
    LinkedList() { first = NULL; }
    LinkedList(int A[], int n);
    ~LinkedList();

    void Display();
    void Insert(int index, int x);
    int Delete(int index);
    Node* Search(int key);
    Node* Search(Node* p, int key);
};

LinkedList::LinkedList(int A[], int n) {
    Node *last, *t;
    first = new Node;
    first->data = A[0];
    first->next = NULL;
    last = first;

```

```
    for (int i = 1; i < n; i++) {
        t = new Node;
        t->data = A[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
}

LinkedList::~LinkedList() {
    Node *p = first;
    while (first) {
        first = first->next;
        delete p;
        p = first;
    }
}

void LinkedList::Insert(int index, int x) {
    Node *p = first;
    if (index == 0) {
        Node *t = new Node;
        t->data = x;
        t->next = first;
        first = t;
    } else {
        Node *t = new Node;
        t->data = x;
        for (int i = 0; i < index - 1 && p; i++)
            p = p->next;
        if (p) {
            t->next = p->next;
            p->next = t;
        }
    }
}

int LinkedList::Delete(int index) {
    Node *p = first, *q = NULL;
    int x = -1;

    if (index < 1) {
        cout << "Error: Invalid index" << endl;
        return -1;
    }

    if (index == 1) {
        first = first->next;
        x = p->data;
        delete p;
    } else {
        for (int i = 0; i < index - 1 && p; i++) {
            q = p;
            p = p->next;
        }
        if (p) {
            q->next = p->next;
            x = p->data;
            delete p;
        }
    }

    return x;
}
```

```
97
98  Node* LinkedList::Search(int key) {
99      return Search(first, key);
100 }
101
102 Node* LinkedList::Search(Node* p, int key) {
103     while (p != NULL) {
104         if (key == p->data) {
105             cout << "Found at node with value: " << p->data << endl;
106             return p;
107         }
108         p = p->next;
109     }
110     cout << "Element not found" << endl;
111     return NULL;
112 }
113
114 void LinkedList::Display() {
115     Node *p = first;
116     while (p) {
117         cout << p->data << " ";
118         p = p->next;
119     }
120     cout << endl;
121 }
122
123 int main() {
124     int A[] = {1, 2, 3, 4, 5};
125     LinkedList l(A, 5);
126
127     l.Insert(0, 20);
128     l.Display();
```

```
129
130     l.Insert(3, 10);
131     l.Display();
132
133     l.Delete(1);
134     l.Display();
135
136     l.Delete(3);
137     l.Display();
138
139     l.Search(10);
140     l.Display();
141
142     return 0;
143 }
144
```

OUTPUT

```
Output                                                    Clear
20 1 2 3 4 5
20 1 2 10 3 4 5
1 2 10 3 4 5
1 2 3 4 5
Element not found
1 2 3 4 5


=== Code Execution Successful ===
```

2. Write a program to implement doubly linked list as an ADT that supports the following operations:

- Insert an element x at the beginning of the doubly linked list
- Insert an element x at the end of the doubly linked list
- Remove an element from the beginning of the doubly linked list
- Remove an element from the end of the doubly linked list

INPUT

main.cpp — Share — Run

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

class Node {
public:
    int data;
    Node *prev;
    Node *next;
};

class DoublyLinkedList {
private:
    Node *head;
    Node *tail;

public:
    DoublyLinkedList() { head = tail = NULL; }
    DoublyLinkedList(int A[], int n);
    ~DoublyLinkedList();

    void Display();
    void Insert(int index, int x);
    int Delete(int index);
};

DoublyLinkedList::DoublyLinkedList(int A[], int n) {
    Node *last, *t;
    head = new Node;
    head->data = A[0];
    head->prev = NULL;
    head->next = NULL;
    last = head;
```

```
    tail = head;

    for (int i = 1; i < n; i++) {
        t = new Node;
        t->data = A[i];
        t->prev = last;
        t->next = NULL;
        last->next = t;
        last = t;
    }
    tail = last;
}

DoublyLinkedList::~DoublyLinkedList() {
    Node *p = head;
    while (head) {
        head = head->next;
        delete p;
        p = head;
    }
}

void DoublyLinkedList::Insert(int index, int x) {
    Node *p = head;
    Node *t = new Node;
    t->data = x;
    t->prev = t->next = NULL;

    if (index == 0) {
        t->next = head;
        if (head)
            head->prev = t;
        else
            tail = t;
        head = t;
    } else {
        for (int i = 0; i < index - 1 && p; i++)
            p = p->next;
        if (p) {
            t->next = p->next;
            t->prev = p;
            if (p->next)
                p->next->prev = t;
            else
                tail = t;
            p->next = t;
        }
    }
}

int DoublyLinkedList::Delete(int index) {
    Node *p = head;
    int x = -1;

    if (index < 0) {
        cout << "Error: Invalid index" << endl;
        return -1;
    }

    if (index == 0 && head) {
        x = head->data;
        p = head;
        head = head->next;
        if (head)
```

```
            head->prev = NULL;
        else
            tail = NULL;
        delete p;
    } else {
        for (int i = 0; i < index && p; i++)
            p = p->next;
        if (p) {
            x = p->data;
            if (p->prev)
                p->prev->next = p->next;
            if (p->next)
                p->next->prev = p->prev;
            if (p == tail)
                tail = p->prev;
            delete p;
        }
    }

    return x;
}

void DoublyLinkedList::Display() {
    Node *p = head;
    while (p) {
        cout << p->data << " ";
        p = p->next;
    }
    cout << endl;
}

int main() {
```

```
    int A[] = {4, 6, 7, 8, 9};
    DoublyLinkedList dl(A, 5);

    dl.Insert(0, 20);
    dl.Display();

    dl.Insert(3, 10);
    dl.Display();

    dl.Delete(0);
    dl.Display();

    dl.Delete(3);
    dl.Display();

    return 0;
}
```

OUTPUT

```
Output                                                    Clear
20 4 6 7 8 9
20 4 6 10 7 8 9
4 6 10 7 8 9
4 6 10 8 9


=== Code Execution Successful ===
```

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

- Insert an element x in the list
- Remove an element from the list
- Search for an element x in the list and return its pointer

INPUT

main.cpp Share Run

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

class Node {
public:
    int data;
    Node *next;
};

class CircularLinkedList {
private:
    Node *head;

public:
    CircularLinkedList() { head = NULL; }
    CircularLinkedList(int A[], int n);
    ~CircularLinkedList();

    void Display();
    void Insert(int x);
    void Delete(int x);
    Node* Search(int x);
};

CircularLinkedList::CircularLinkedList(int A[], int n) {
    head = NULL;
    for (int i = 0; i < n; i++) {
        Insert(A[i]);
    }
}

CircularLinkedList::~CircularLinkedList() {
```

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

void CircularLinkedList::Insert(int x) {
    Node *t = new Node;
    t->data = x;
    t->next = NULL;

    if (!head) {
        head = t;
        head->next = head;
    } else {
        Node *p = head;
        while (p->next != head)
            p = p->next;
        p->next = t;
        t->next = head;
    }
}

void CircularLinkedList::Delete(int x) {
    if (!head) return;

    Node *p = head, *q = NULL;
```

```

    // Special case: deleting head
    if (head->data == x) {
        if (head->next == head) {
            delete head;
            head = NULL;
        } else {
            while (p->next != head)
                p = p->next;
            Node *temp = head;
            p->next = head->next;
            head = head->next;
            delete temp;
        }
        return;
    }

    p = head;
    do {
        q = p;
        p = p->next;
        if (p->data == x) {
            q->next = p->next;
            delete p;
            return;
        }
    } while (p != head);
}

Node* CircularLinkedList::Search(int x) {
    if (!head) return NULL;

```

```
    Node *p = head;
    do {
        if (p->data == x) {
            cout << "Found at node with value: " << p->data << endl;
            return p;
        }
        p = p->next;
    } while (p != head);

    cout << "Element not found" << endl;
    return NULL;
}

void CircularLinkedList::Display() {
    if (!head) return;

    Node *p = head;
    do {
        cout << p->data << " ";
        p = p->next;
    } while (p != head);
    cout << endl;
}

int main() {
    int A[] = {3, 5, 6, 7, 8};
    CircularLinkedList cl(A, 5);

    cl.Display();

    cl.Insert(10);
    cl.Display();
```

```

     cl.Delete(3);
     cl.Display();

     cl.Search(4);
     cl.Display();

     return 0;
}

```

OUTPUT

```
Output                                                    Clear
3 5 6 7 8
3 5 6 7 8 10
5 6 7 8 10
Element not found
5 6 7 8 10


=== Code Execution Successful ===
```

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

INPUT

main.cpp

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

class Stack {
private:
    int top;
    int size;
    int *S;

public:
    Stack(int sz) {
        size = sz;
        top = -1;
        S = new int[size];
    }

    ~Stack() {
        delete[] S;
    }

    void Push(int x);
    int Pop();
    bool isEmpty();
};

void Stack::Push(int x) {
    if (top == size - 1)
        cout << "Stack Overflow" << endl;
    else
        S[++top] = x;
}
```

```
33
34  int Stack::Pop() {
35      if (top == -1) {
36          cout << "Stack Underflow" << endl;
37          return -1;
38      } else
39          return S[top--];
40  }
41
42  bool Stack::isEmpty() {
43      return top == -1;
44  }
45
46  int EvaluatePostfix(const char *exp) {
47      Stack st(strlen(exp));
48      for (int i = 0; exp[i]; i++) {
49          if (isdigit(exp[i])) {
50              st.Push(exp[i] - '0');
51          } else {
52              int b = st.Pop();
53              int a = st.Pop();
54              switch (exp[i]) {
55                  case '+': st.Push(a + b); break;
56                  case '-': st.Push(a - b); break;
57                  case '*': st.Push(a * b); break;
58                  case '/': st.Push(a / b); break;
59              }
60          }
61      }
62      return st.Pop();
63  }
64
```

```
65  int EvaluatePrefix(const char *exp) {
66      Stack st(strlen(exp));
67      for (int i = strlen(exp) - 1; i >= 0; i--) {
68          if (isdigit(exp[i])) {
69              st.Push(exp[i] - '0');
70          } else {
71              int a = st.Pop();
72              int b = st.Pop();
73              switch (exp[i]) {
74                  case '+': st.Push(a + b); break;
75                  case '-': st.Push(a - b); break;
76                  case '*': st.Push(a * b); break;
77                  case '/': st.Push(a / b); break;
78              }
79          }
80      }
81      return st.Pop();
82  }
83
84  int main() {
85      const char *postfix = "534*+";
86      const char *prefix = "+5*34";
87
88      cout << "Postfix Expression: " << postfix << endl;
89      cout << "Postfix Evaluation: " << EvaluatePostfix(postfix) << endl;
90
91      cout << "Prefix Expression: " << prefix << endl;
92      cout << "Prefix Evaluation: " << EvaluatePrefix(prefix) << endl;
93
94      return 0;
95  }
96
```

OUTPUT

```
Output                                                    Clear
Postfix Expression: 534*+
Postfix Evaluation: 17
Prefix Expression: +5*34
Prefix Evaluation: 17


=== Code Execution Successful ===
```

5. Implement Queue as an ADT.

INPUT

main.cpp Share Run

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

class Queue {
private:
    int front;
    int rear;
    int size;
    int *Q;

public:
    Queue(int sz) {
        size = sz;
        front = rear = -1;
        Q = new int[size];
    }

    ~Queue() {
        delete[] Q;
    }

    void Enqueue(int x);
    int Dequeue();
    void Display();
    bool isEmpty();
    bool isFull();
};

void Queue::Enqueue(int x) {
    if (rear == size - 1)
        cout << "Queue Overflow" << endl;
    else {
```

```
        rear++;
        Q[rear] = x;
    }
}

int Queue::Dequeue() {
    if (front == rear) {
        cout << "Queue Underflow" << endl;
        return -1;
    } else {
        front++;
        return Q[front];
    }
}

void Queue::Display() {
    for (int i = front + 1; i <= rear; i++)
        cout << Q[i] << " ";
    cout << endl;
}

bool Queue::isEmpty() {
    return front == rear;
}

bool Queue::isFull() {
    return rear == size - 1;
}

int main() {
    Queue q(5);

    q.Enqueue(10);
    q.Enqueue(20);
    q.Enqueue(30);
    q.Display();

    q.Dequeue();
    q.Display();

    q.Enqueue(40);
    q.Enqueue(50);
    q.Display();

    return 0;
}
```

OUTPUT

```
Output                                                    Clear
10 20 30
20 30
20 30 40 50


=== Code Execution Successful ===
```

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

- Insert an element x
- Delete an element x
- Search for an element x in the BST
- Display the elements of the BST in preorder, inorder, and postorder traversal

INPUT

main.cpp Share Run

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

class Node {
public:
    int data;
    Node *left;
    Node *right;
};

class BST {
private:
    Node *root;

public:
    BST() { root = NULL; }

    void Insert(int x);
    void Delete(int x);
    Node* Search(int x);
    void Inorder();
    void Preorder();
    void Postorder();

private:
    Node* Insert(Node *p, int x);
    Node* Delete(Node *p, int x);
    Node* Search(Node *p, int x);
    void Inorder(Node *p);
    void Preorder(Node *p);
    void Postorder(Node *p);
    Node* FindMin(Node *p);
```

```
};

void BST::Insert(int x) {
    root = Insert(root, x);
}

Node* BST::Insert(Node *p, int x) {
    if (p == NULL) {
        p = new Node;
        p->data = x;
        p->left = p->right = NULL;
    } else if (x < p->data)
        p->left = Insert(p->left, x);
    else if (x > p->data)
        p->right = Insert(p->right, x);
    return p;
}

void BST::Delete(int x) {
    root = Delete(root, x);
}

Node* BST::Delete(Node *p, int x) {
    if (p == NULL) return NULL;

    if (x < p->data)
        p->left = Delete(p->left, x);
    else if (x > p->data)
        p->right = Delete(p->right, x);
    else {
        if (p->left == NULL && p->right == NULL) {
            delete p;
            return NULL;
        } else if (p->left == NULL) {
            Node *temp = p->right;
            delete p;
            return temp;
        } else if (p->right == NULL) {
            Node *temp = p->left;
            delete p;
            return temp;
        } else {
            Node *temp = FindMin(p->right);
            p->data = temp->data;
            p->right = Delete(p->right, temp->data);
        }
    }
    return p;
}

Node* BST::FindMin(Node *p) {
    while (p && p->left)
        p = p->left;
    return p;
}

Node* BST::Search(int x) {
    return Search(root, x);
}

Node* BST::Search(Node *p, int x) {
    while (p) {
        if (x == p->data) {
            cout << "Found: " << p->data << endl;
```

```
97              return p;
98          } else if (x < p->data)
99              p = p->left;
100         else
101             p = p->right;
102     }
103     cout << "Not found" << endl;
104     return NULL;
105 }
106
107 void BST::Inorder() {
108     Inorder(root);
109     cout << endl;
110 }
111
112 void BST::Inorder(Node *p) {
113     if (p) {
114         Inorder(p->left);
115         cout << p->data << " ";
116         Inorder(p->right);
117     }
118 }
119
120 void BST::Preorder() {
121     Preorder(root);
122     cout << endl;
123 }
124
125 void BST::Preorder(Node *p) {
126     if (p) {
127         cout << p->data << " ";
128         Preorder(p->left);
```

```
129         Preorder(p->right);
130     }
131 }
132
133 void BST::Postorder() {
134     Postorder(root);
135     cout << endl;
136 }
137
138 void BST::Postorder(Node *p) {
139     if (p) {
140         Postorder(p->left);
141         Postorder(p->right);
142         cout << p->data << " ";
143     }
144 }
145
146 int main() {
147     BST tree;
148
149     tree.Insert(50);
150     tree.Insert(30);
151     tree.Insert(70);
152     tree.Insert(20);
153     tree.Insert(40);
154     tree.Insert(60);
155     tree.Insert(80);
156
157     cout << "Inorder: ";
158     tree.Inorder();
159
160     cout << "Preorder: ";
```

```
161      tree.Preorder();
162
163      cout << "Postorder: ";
164      tree.Postorder();
165
166      tree.Search(40);
167      tree.Delete(30);
168
169      cout << "After deleting 30, Inorder: ";
170      tree.Inorder();
171
172      return 0;
173  }
174
```

OUTPUT

```
Output                                                    Clear
Inorder: 20 30 40 50 60 70 80
Preorder: 50 30 20 40 70 60 80
Postorder: 20 40 30 60 80 70 50
Found: 40
After deleting 30, Inorder: 20 40 50 60 70 80


=== Code Execution Successful ===
```

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

INPUT

main.cpp

Share Run

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

class Node {
public:
    int data, height;
    Node *left, *right;
    Node(int val) {
        data = val;
        height = 1;
        left = right = NULL;
    }
};

int height(Node* p) {
    return p ? p->height : 0;
}

int balance(Node* p) {
    return height(p->left) - height(p->right);
}

int max(int a, int b) {
    return (a > b) ? a : b;
}

Node* rightRotate(Node* y) {
    Node* x = y->left;
    y->left = x->right;
    x->right = y;
    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;
    x->right = y->left;
    y->left = x;
    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* root, int key) {
    if (!root) return new Node(key);
    if (key < root->data) root->left = insert(root->left, key);
    else if (key > root->data) root->right = insert(root->right, key);
    else return root;

    root->height = max(height(root->left), height(root->right)) + 1;
    int bf = balance(root);

    if (bf > 1 && key < root->left->data) return rightRotate(root);
    if (bf < -1 && key > root->right->data) return leftRotate(root);
    if (bf > 1 && key > root->left->data) {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    if (bf < -1 && key < root->right->data) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

```

```
    return root;
}

Node* search(Node* root, int key) {
    while (root) {
        if (key == root->data) {
            cout << "Found: " << key << endl;
            return root;
        }
        root = (key < root->data) ? root->left : root->right;
    }
    cout << "Not found" << endl;
    return NULL;
}

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

int main() {
    Node* root = NULL;
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 10);
    root = insert(root, 25);

    cout << "Inorder: ";
    inorder(root);
    cout << endl;

    search(root, 25);
    search(root, 10);

    return 0;
}
```

OUTPUT

Output Clear

```
Inorder: 10 20 25 30 40
Found: 25
Found: 10


=== Code Execution Successful ===
```