

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

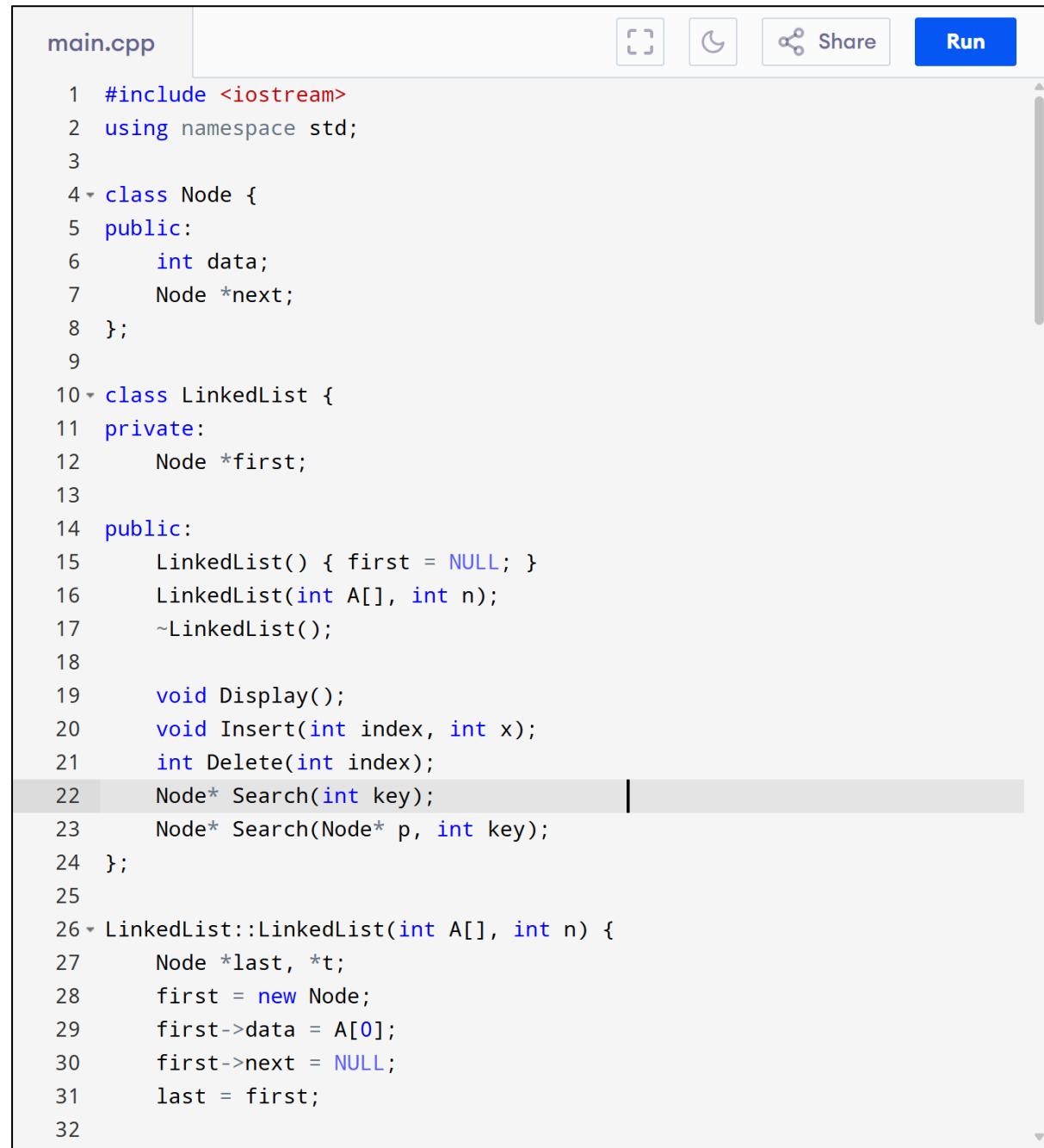
SUBMITTED TO

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



The screenshot shows a code editor window with the file name "main.cpp" at the top left. The code itself is a C++ program for a singly linked list. It includes a Node class with data and a next pointer, and a LinkedList class with methods for construction, insertion, deletion, and search. The code is numbered from 1 to 32. At the top right of the editor, there are several icons: a zoom-in square, a refresh circle, a share icon, and a blue "Run" button. The code is as follows:

```
1 #include <iostream>
2 using namespace std;
3
4 class Node {
5 public:
6     int data;
7     Node *next;
8 };
9
10 class LinkedList {
11 private:
12     Node *first;
13
14 public:
15     LinkedList() { first = NULL; }
16     LinkedList(int A[], int n);
17     ~LinkedList();
18
19     void Display();
20     void Insert(int index, int x);
21     int Delete(int index);
22     Node* Search(int key);
23     Node* Search(Node* p, int key);
24 };
25
26 LinkedList::LinkedList(int A[], int n) {
27     Node *last, *t;
28     first = new Node;
29     first->data = A[0];
30     first->next = NULL;
31     last = first;
32 }
```

```
33+     for (int i = 1; i < n; i++) {
34         t = new Node;
35         t->data = A[i];
36         t->next = NULL;
37         last->next = t;
38         last = t;
39     }
40 }
41
42~ LinkedList::~LinkedList() {
43     Node *p = first;
44~     while (first) {
45         first = first->next;
46         delete p;
47         p = first;
48     }
49 }
50
51~ void LinkedList::Insert(int index, int x) {
52     Node *p = first;
53~     if (index == 0) {
54         Node *t = new Node;
55         t->data = x;
56         t->next = first;
57         first = t;
58~     } else {
59         Node *t = new Node;
60         t->data = x;
61         for (int i = 0; i < index - 1 && p; i++)
62             p = p->next;
63~         if (p) {
64             t->next = p->next;
65             p->next = t;
66         }
67     }
68 }
69
70~ int LinkedList::Delete(int index) {
71     Node *p = first, *q = NULL;
72     int x = -1;
73
74~     if (index < 1) {
75         cout << "Error: Invalid index" << endl;
76         return -1;
77     }
78
79~     if (index == 1) {
80         first = first->next;
81         x = p->data;
82         delete p;
83~     } else {
84~         for (int i = 0; i < index - 1 && p; i++) {
85             q = p;
86             p = p->next;
87         }
88~         if (p) {
89             q->next = p->next;
90             x = p->data;
91             delete p;
92         }
93     }
94
95     return x;
96 }
```

```
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

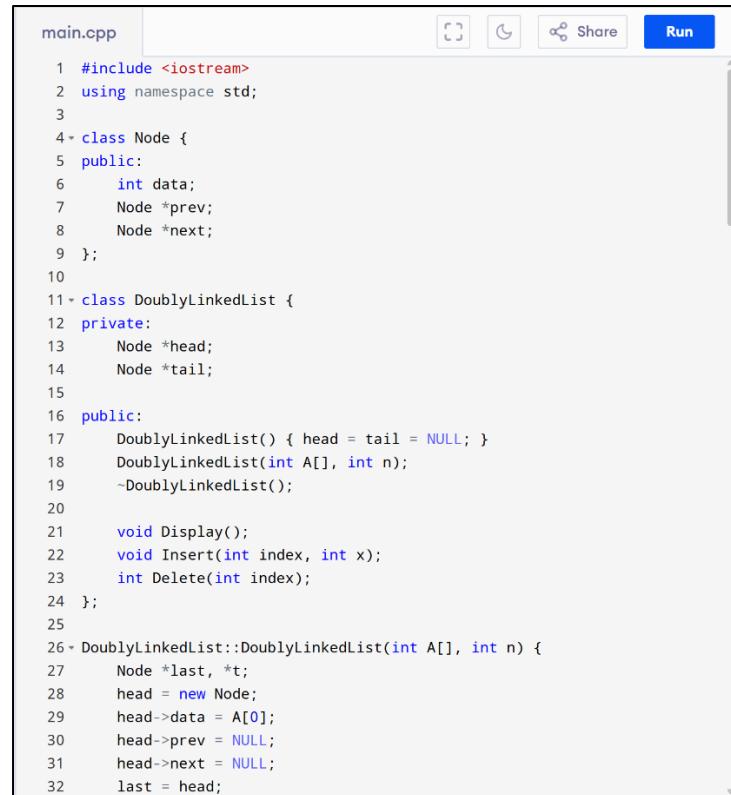
```
Output Clear
20 1 2 3 4 5
20 1 2 10 3 4 5
1 2 10 3 4 5
1 2 3 4 5
Element not found
1 2 3 4 5

==== 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



The screenshot shows a code editor window with the file name "main.cpp" at the top left. The code itself is a C++ program defining a DoublyLinkedList class. The code includes a Node class with data, prev, and next pointers, and a DoublyLinkedList class with head and tail pointers, along with methods for Display, Insert, and Delete. The code is numbered from 1 to 32.

```
main.cpp
1 #include <iostream>
2 using namespace std;
3
4 class Node {
5 public:
6     int data;
7     Node *prev;
8     Node *next;
9 };
10
11 class DoublyLinkedList {
12 private:
13     Node *head;
14     Node *tail;
15
16 public:
17     DoublyLinkedList() { head = tail = NULL; }
18     DoublyLinkedList(int A[], int n);
19     ~DoublyLinkedList();
20
21     void Display();
22     void Insert(int index, int x);
23     int Delete(int index);
24 };
25
26 DoublyLinkedList::DoublyLinkedList(int A[], int n) {
27     Node *last, *t;
28     head = new Node;
29     head->data = A[0];
30     head->prev = NULL;
31     head->next = NULL;
32     last = head;
```

```
33     tail = head;
34
35~   for (int i = 1; i < n; i++) {
36     t = new Node;
37     t->data = A[i];
38     t->prev = last;
39     t->next = NULL;
40     last->next = t;
41     last = t;
42   }
43   tail = last;
44 }
45
46~ DoublyLinkedList::~DoublyLinkedList() {
47   Node *p = head;
48~   while (head) {
49     head = head->next;
50     delete p;
51     p = head;
52   }
53 }
54
55~ void DoublyLinkedList::Insert(int index, int x) {
56   Node *p = head;
57   Node *t = new Node;
58   t->data = x;
59   t->prev = t->next = NULL;
60
61~   if (index == 0) {
62     t->next = head;
63     if (head)
64       head->prev = t;

```

```
65     else
66       tail = t;
67     head = t;
68~   } else {
69     for (int i = 0; i < index - 1 && p; i++)
70       p = p->next;
71~   if (p) {
72     t->next = p->next;
73     t->prev = p;
74     if (p->next)
75       p->next->prev = t;
76     else
77       tail = t;
78     p->next = t;
79   }
80 }
81 }
82
83~ int DoublyLinkedList::Delete(int index) {
84   Node *p = head;
85   int x = -1;
86
87~   if (index < 0) {
88     cout << "Error: Invalid index" << endl;
89     return -1;
90   }
91
92~   if (index == 0 && head) {
93     x = head->data;
94     p = head;
95     head = head->next;
96     if (head)

```

```
97         head->prev = NULL;
98     else
99         tail = NULL;
100    delete p;
101 } else {
102     for (int i = 0; i < index && p; i++)
103         p = p->next;
104 if (p) {
105     x = p->data;
106     if (p->prev)
107         p->prev->next = p->next;
108     if (p->next)
109         p->next->prev = p->prev;
110     if (p == tail)
111         tail = p->prev;
112     delete p;
113 }
114 }
115
116 return x;
117 }
118
119 void DoublyLinkedList::Display() {
120     Node *p = head;
121 while (p) {
122     cout << p->data << " ";
123     p = p->next;
124 }
125 cout << endl;
126 }
127
128 int main() {
```

```
129     int A[] = {4, 6, 7, 8, 9};
130     DoublyLinkedList dl(A, 5);
131
132     dl.Insert(0, 20);
133     dl.Display();
134
135     dl.Insert(3, 10);
136     dl.Display();
137
138     dl.Delete(0);
139     dl.Display();
140
141     dl.Delete(3);
142     dl.Display();
143
144     return 0;
145 }
```

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 | Run Share  
1 #include <iostream>  
2 using namespace std;  
3  
4 class Node {  
5 public:  
6     int data;  
7     Node *next;  
8 };  
9  
10 class CircularLinkedList {  
11 private:  
12     Node *head;  
13  
14 public:  
15     CircularLinkedList() { head = NULL; }  
16     CircularLinkedList(int A[], int n);  
17     ~CircularLinkedList();  
18  
19     void Display();  
20     void Insert(int x);  
21     void Delete(int x);  
22     Node* Search(int x);  
23 };  
24  
25 CircularLinkedList::CircularLinkedList(int A[], int n) {  
26     head = NULL;  
27     for (int i = 0; i < n; i++) {  
28         Insert(A[i]);  
29     }  
30 }  
31  
32 CircularLinkedList::~CircularLinkedList() {
```

```
33     if (!head) return;
34     Node *p = head->next;
35     while (p != head) {
36         Node *temp = p;
37         p = p->next;
38         delete temp;
39     }
40     delete head;
41     head = NULL;
42 }
43
44 void CircularLinkedList::Insert(int x) {
45     Node *t = new Node;
46     t->data = x;
47     t->next = NULL;
48
49     if (!head) {
50         head = t;
51         head->next = head;
52     } else {
53         Node *p = head;
54         while (p->next != head)
55             p = p->next;
56         p->next = t;
57         t->next = head;
58     }
59 }
60
61 void CircularLinkedList::Delete(int x) {
62     if (!head) return;
63
64     Node *p = head, *q = NULL;
```

```
65
66 // Special case: deleting head
67 if (head->data == x) {
68     if (head->next == head) {
69         delete head;
70         head = NULL;
71     } else {
72         while (p->next != head)
73             p = p->next;
74         Node *temp = head;
75         p->next = head->next;
76         head = head->next;
77         delete temp;
78     }
79     return;
80 }
81
82 p = head;
83 do {
84     q = p;
85     p = p->next;
86     if (p->data == x) {
87         q->next = p->next;
88         delete p;
89         return;
90     }
91 } while (p != head);
92 }
93
94 Node* CircularLinkedList::Search(int x) {
95     if (!head) return NULL;
96 }
```

```
97     Node *p = head;
98     do {
99         if (p->data == x) {
100             cout << "Found at node with value: " << p->data << endl;
101             return p;
102         }
103         p = p->next;
104     } while (p != head);
105
106    cout << "Element not found" << endl;
107    return NULL;
108 }
109
110 void CircularLinkedList::Display() {
111     if (!head) return;
112
113     Node *p = head;
114     do {
115         cout << p->data << " ";
116         p = p->next;
117     } while (p != head);
118     cout << endl;
119 }
120
121 int main() {
122     int A[] = {3, 5, 6, 7, 8};
123     CircularLinkedList cl(A, 5);
124
125     cl.Display();
126
127     cl.Insert(10);
128     cl.Display();
```

```
129
130     cl.Delete(3);
131     cl.Display();
132
133     cl.Search(4);
134     cl.Display();
135
136     return 0;
137 }
138
```

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

Run

```
1 #include <iostream>  
2 #include <cstring>  
3 using namespace std;  
4  
5 class Stack {  
6 private:  
7     int top;  
8     int size;  
9     int *S;  
10  
11 public:  
12     Stack(int sz) {  
13         size = sz;  
14         top = -1;  
15         S = new int[size];  
16     }  
17  
18     ~Stack() {  
19         delete[] S;  
20     }  
21  
22     void Push(int x);  
23     int Pop();  
24     bool isEmpty();  
25 };  
26  
27 void Stack::Push(int x) {  
28     if (top == size - 1)  
29         cout << "Stack Overflow" << endl;  
30     else  
31         S[++top] = x;  
32 }
```

```
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 }
```

OUTPUT

Output

Clear

Postfix Expression: 534*+

Postfix Evaluation: 17

Prefix Expression: +5*34

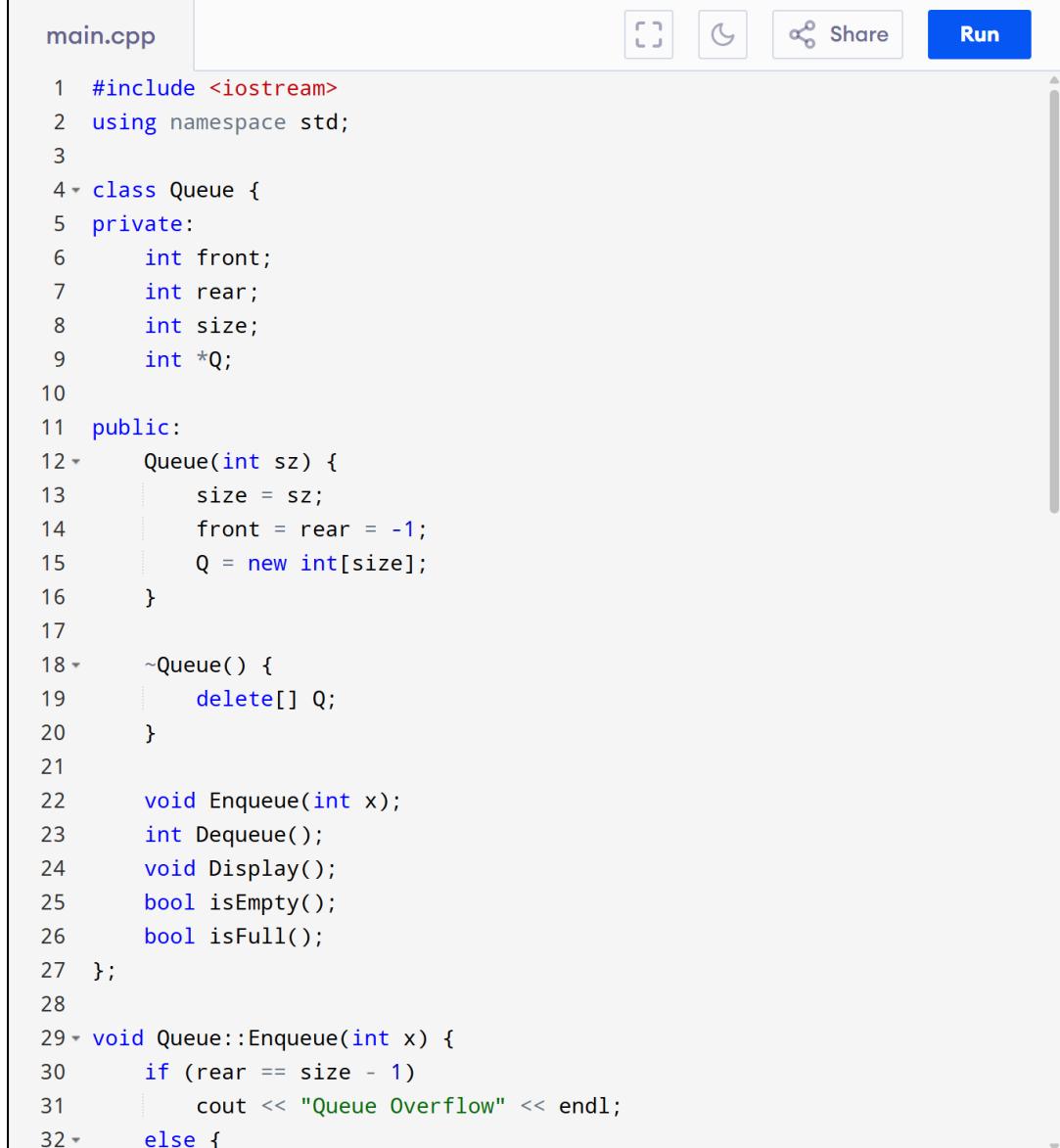
Prefix Evaluation: 17

==== Code Execution Successful ===

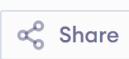
5. Implement Queue as an ADT.

INPUT

```
main.cpp
```



```
1 #include <iostream>
2 using namespace std;
3
4 class Queue {
5 private:
6     int front;
7     int rear;
8     int size;
9     int *Q;
10
11 public:
12     Queue(int sz) {
13         size = sz;
14         front = rear = -1;
15         Q = new int[size];
16     }
17
18     ~Queue() {
19         delete[] Q;
20     }
21
22     void Enqueue(int x);
23     int Dequeue();
24     void Display();
25     bool IsEmpty();
26     bool IsFull();
27 };
28
29 void Queue::Enqueue(int x) {
30     if (rear == size - 1)
31         cout << "Queue Overflow" << endl;
32     else {
```



Run

```
33         rear++;
34         Q[rear] = x;
35     }
36 }
37
38 int Queue::Dequeue() {
39     if (front == rear) {
40         cout << "Queue Underflow" << endl;
41         return -1;
42     } else {
43         front++;
44         return Q[front];
45     }
46 }
47
48 void Queue::Display() {
49     for (int i = front + 1; i <= rear; i++)
50         cout << Q[i] << " ";
51     cout << endl;
52 }
53
54 bool Queue::isEmpty() {
55     return front == rear;
56 }
57
58 bool Queue::isFull() {
59     return rear == size - 1;
60 }
61
62 int main() {
63     Queue q(5);
64 }
```

```
65     q.Enqueue(10);
66     q.Enqueue(20);
67     q.Enqueue(30);
68     q.Display();
69
70     q.Dequeue();
71     q.Display();
72
73     q.Enqueue(40);
74     q.Enqueue(50);
75     q.Display();
76
77     return 0;
78 }
```

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  
1 #include <iostream>  
2 using namespace std;  
3  
4 class Node {  
5 public:  
6     int data;  
7     Node *left;  
8     Node *right;  
9 };  
10  
11 class BST {  
12 private:  
13     Node *root;  
14  
15 public:  
16     BST() { root = NULL; }  
17  
18     void Insert(int x);  
19     void Delete(int x);  
20     Node* Search(int x);  
21     void Inorder();  
22     void Preorder();  
23     void Postorder();  
24  
25 private:  
26     Node* Insert(Node *p, int x);  
27     Node* Delete(Node *p, int x);  
28     Node* Search(Node *p, int x);  
29     void Inorder(Node *p);  
30     void Preorder(Node *p);  
31     void Postorder(Node *p);  
32     Node* FindMin(Node *p);
```

```
33  };
34
35~ void BST::Insert(int x) {
36     root = Insert(root, x);
37 }
38
39~ Node* BST::Insert(Node *p, int x) {
40~     if (p == NULL) {
41         p = new Node;
42         p->data = x;
43         p->left = p->right = NULL;
44     } else if (x < p->data)
45         p->left = Insert(p->left, x);
46     else if (x > p->data)
47         p->right = Insert(p->right, x);
48     return p;
49 }
50
51~ void BST::Delete(int x) {
52     root = Delete(root, x);
53 }
54
55~ Node* BST::Delete(Node *p, int x) {
56     if (p == NULL) return NULL;
57
58     if (x < p->data)
59         p->left = Delete(p->left, x);
60     else if (x > p->data)
61         p->right = Delete(p->right, x);
62~     else {
63~         if (p->left == NULL && p->right == NULL) {
64             delete p;

```

```
65             return NULL;
66~     } else if (p->left == NULL) {
67         Node *temp = p->right;
68         delete p;
69         return temp;
70~     } else if (p->right == NULL) {
71         Node *temp = p->left;
72         delete p;
73         return temp;
74~     } else {
75         Node *temp = FindMin(p->right);
76         p->data = temp->data;
77         p->right = Delete(p->right, temp->data);
78     }
79 }
80 return p;
81 }
82
83~ Node* BST::FindMin(Node *p) {
84     while (p && p->left)
85         p = p->left;
86     return p;
87 }
88
89~ Node* BST::Search(int x) {
90     return Search(root, x);
91 }
92
93~ Node* BST::Search(Node *p, int x) {
94~     while (p) {
95~         if (x == p->data) {
96             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

```
1 #include <iostream>
2 using namespace std;
3
4 class Node {
5 public:
6     int data, height;
7     Node *left, *right;
8     Node(int val) {
9         data = val;
10        height = 1;
11        left = right = NULL;
12    }
13 };
14
15 int height(Node* p) {
16     return p ? p->height : 0;
17 }
18
19 int balance(Node* p) {
20     return height(p->left) - height(p->right);
21 }
22
23 int max(int a, int b) {
24     return (a > b) ? a : b;
25 }
26
27 Node* rightRotate(Node* y) {
28     Node* x = y->left;
29     y->left = x->right;
30     x->right = y;
31     y->height = max(height(y->left), height(y->right)) + 1;
32     x->height = max(height(x->left), height(x->right)) + 1;
```

```
33     return x;
34 }
35
36 Node* leftRotate(Node* x) {
37     Node* y = x->right;
38     x->right = y->left;
39     y->left = x;
40     x->height = max(height(x->left), height(x->right)) + 1;
41     y->height = max(height(y->left), height(y->right)) + 1;
42     return y;
43 }
44
45 Node* insert(Node* root, int key) {
46     if (!root) return new Node(key);
47     if (key < root->data) root->left = insert(root->left, key);
48     else if (key > root->data) root->right = insert(root->right, key);
49     else return root;
50
51     root->height = max(height(root->left), height(root->right)) + 1;
52     int bf = balance(root);
53
54     if (bf > 1 && key < root->left->data) return rightRotate(root);
55     if (bf < -1 && key > root->right->data) return leftRotate(root);
56     if (bf > 1 && key > root->left->data) {
57         root->left = leftRotate(root->left);
58         return rightRotate(root);
59     }
60     if (bf < -1 && key < root->right->data) {
61         root->right = rightRotate(root->right);
62         return leftRotate(root);
63     }
64 }
```

```
65     return root;
66 }
67
68 Node* search(Node* root, int key) {
69     while (root) {
70         if (key == root->data) {
71             cout << "Found: " << key << endl;
72             return root;
73         }
74         root = (key < root->data) ? root->left : root->right;
75     }
76     cout << "Not found" << endl;
77     return NULL;
78 }
79
80 void inorder(Node* root) {
81     if (root) {
82         inorder(root->left);
83         cout << root->data << " ";
84         inorder(root->right);
85     }
86 }
87
88 int main() {
89     Node* root = NULL;
90     root = insert(root, 30);
91     root = insert(root, 20);
92     root = insert(root, 40);
93     root = insert(root, 10);
94     root = insert(root, 25);
95
96     cout << "Inorder: ";
97     inorder(root);
98     cout << endl;
99
100    search(root, 25);
101    search(root, 10);
102
103    return 0;
104 }
105
```

OUTPUT

Output

Clear

Inorder: 10 20 25 30 40

Found: 25

Found: 10

==== Code Execution Successful ===