

```

1  #include <iostream>
2  #include <queue>
3
4  using namespace std;
5
6  struct Node {
7      char info;
8      Node* left;
9      Node* right;
10 };
11
12 Node* createNode(char data) {
13     Node* newNode = new Node();
14     newNode->info = data;
15     newNode->left = NULL;
16     newNode->right = NULL;
17     return newNode;
18 }
19
20 void preOrder(Node* root) {
21     if (root != NULL) {
22         cout << root->info << " ";
23         preOrder(root->left);
24         preOrder(root->right);
25     }
26 }
27
28 void inOrder(Node* root) {
29     if (root != NULL) {
30         inOrder(root->left);
31         cout << root->info << " ";
32         inOrder(root->right);
33     }
34 }
35
36 void postOrder(Node* root) {
37     if (root != NULL) {
38         postOrder(root->left);
39         postOrder(root->right);
40         cout << root->info << " ";
41     }
42 }
43
44 void levelOrder(Node* root) {
45     if (root == NULL) return;
46
47     queue<Node*> q;
48     q.push(root);
49
50     while (!q.empty()) {
51         Node* current = q.front();
52         q.pop();
53
54         cout << current->info << " ";
55
56         if (current->left != NULL) q.push(current->left);
57         if (current->right != NULL) q.push(current->right);
58     }
59 }
60

```

Exercise 1

```

61 int main() {
62     Node* root = createNode('/');
63
64     root->left = createNode('+');
65     root->right = createNode('g');
66
67     root->left->left = createNode('*');
68     root->left->right = createNode('^');
69
70     root->left->left->left = createNode('+');
71     root->left->left->right = createNode('/');
72
73     root->left->right->left = createNode('e');
74     root->left->right->right = createNode('f');
75
76     root->left->left->left->left = createNode('a');
77     root->left->left->left->right = createNode('b');
78
79     root->left->left->right->left = createNode('c');
80     root->left->left->right->right = createNode('d');
81
82     cout << "\n1. Pre-order (Root-Left-Right):" << endl;
83     preOrder(root);
84
85     cout << "\n\n2. In-order (Left-Root-Right):" << endl;
86     inOrder(root);
87
88     cout << "\n\n3. Post-order (Left-Right-Root):" << endl;
89     postOrder(root);
90
91     cout << "\n\n4. Level-order (BFS):" << endl;
92     levelOrder(root);
93
94     cout << endl;
95     return 0;
96 }

```

Output

PS C:\shellyn\kuliah\semester3\strukturData\teori\tugasTree> cd "c:\shellyn\kuliah\semester3\strukturData\teori\tugasTree"; if (\$?) { g++ exercise1.cpp -o exercise1 }; if (\$?) { .\exercise1 }

1. Pre-order (Root-Left-Right):
/ + * + a b / c d ^ e f g

2. In-order (Left-Root-Right):
a + b * c / d + e ^ f / g

3. Post-order (Left-Right-Root):
a b + c d / * e f ^ + g /

4. Level-order (BFS):
/ + g * ^ + / e f a b c d

PS C:\shellyn\kuliah\semester3\strukturData\teori\tugasTree> █

```

1  #include <iostream>
2  #include <queue>
3
4  using namespace std;
5
6  struct Node {
7      char info;
8      Node* left;
9      Node* right;
10 };
11
12 Node* createNode(char data) {
13     Node* newNode = new Node();
14     newNode->info = data;
15     newNode->left = NULL;
16     newNode->right = NULL;
17     return newNode;
18 }
19
20 void preOrder(Node* root) {
21     if (root != NULL) {
22         cout << root->info << " ";
23         preOrder(root->left);
24         preOrder(root->right);
25     }
26 }
27
28 void inOrder(Node* root) {
29     if (root != NULL) {
30         inOrder(root->left);
31         cout << root->info << " ";
32         inOrder(root->right);
33     }
34 }
35
36 void postOrder(Node* root) {
37     if (root != NULL) {
38         postOrder(root->left);
39         postOrder(root->right);
40         cout << root->info << " ";
41     }
42 }
43
44 void levelOrder(Node* root) {
45     if (root == NULL) return;
46
47     queue<Node*> q;
48     q.push(root);
49
50     while (!q.empty()) {
51         Node* current = q.front();
52         q.pop();
53
54         cout << current->info << " ";
55
56         if (current->left != NULL) q.push(current->left);
57         if (current->right != NULL) q.push(current->right);
58     }
59 }
60

```

Exercise 2

```

61 int main() {
62     Node* root = createNode('/');
63
64     root->left = createNode('*');
65     root->right = createNode('H');
66
67     root->left->left = createNode('-');
68     root->left->right = createNode('G');
69
70     root->left->left->left = createNode('-');
71     root->left->left->right = createNode('F');
72
73     root->left->left->left->left = createNode('-');
74     root->left->left->left->right = createNode('E');
75
76     root->left->left->left->left->left = createNode('+');
77     root->left->left->left->left->right = createNode('N');
78
79     root->left->left->left->left->left->left = createNode('+');
80     root->left->left->left->left->left->right = createNode('M');
81
82     root->left->left->left->left->left->left->left = createNode('K');
83     root->left->left->left->left->left->left->right = createNode('L');
84
85     cout << "\n1. Pre-order (Root-Left-Right):" << endl;
86     preOrder(root);
87
88     cout << "\n2. In-order (Left-Root-Right):" << endl;
89     inOrder(root);
90
91     cout << "\n3. Post-order (Left-Right-Root):" << endl;
92     postOrder(root);
93
94     cout << "\n4. Level-order (BFS):" << endl;
95     levelOrder(root);
96
97     cout << endl;
98     return 0;
99 }

```

Output

● PS C:\shellyn\kuliah\semester3\strukturData\teori\tugasTree> cd "c:\shellyn\kuliah\semester3\strukturData\teori\tugasTree"; if (\$?) { g++ exercise2.cpp -o exercise2 } ; if (\$?) { .\exercise2 }

1. Pre-order (Root-Left-Right):
/ * - - + + K L M N E F G H

2. In-order (Left-Root-Right):
K + L + M - N - E - F * G / H

3. Post-order (Left-Right-Root):
K L + M + N - E - F - G * H /

4. Level-order (BFS):
/ * H - G - F - E + N + M K L

○ PS C:\shellyn\kuliah\semester3\strukturData\teori\tugasTree> □

```

1  #include <iostream>
2  #include <queue>
3
4  using namespace std;
5
6  struct Node {
7      char info;
8      Node* left;
9      Node* right;
10 };
11
12 Node* createNode(char data) {
13     Node* newNode = new Node();
14     newNode->info = data;
15     newNode->left = NULL;
16     newNode->right = NULL;
17     return newNode;
18 }
19
20 void preOrder(Node* root) {
21     if (root != NULL) {
22         cout << root->info << " ";
23         preOrder(root->left);
24         preOrder(root->right);
25     }
26 }
27
28 void inOrder(Node* root) {
29     if (root != NULL) {
30         inOrder(root->left);
31         cout << root->info << " ";
32         inOrder(root->right);
33     }
34 }
35
36 void postOrder(Node* root) {
37     if (root != NULL) {
38         postOrder(root->left);
39         postOrder(root->right);
40         cout << root->info << " ";
41     }
42 }
43
44 void levelOrder(Node* root) {
45     if (root == NULL) return;
46
47     queue<Node*> q;
48     q.push(root);
49
50     while (!q.empty()) {
51         Node* current = q.front();
52         q.pop();
53
54         cout << current->info << " ";
55
56         if (current->left != NULL) q.push(current->left);
57         if (current->right != NULL) q.push(current->right);
58     }
59 }
60

```

Exercise 3

```

61 int main() {
62     Node* root = createNode('/');
63
64     root->left = createNode('+');
65     root->right = createNode('+');
66
67     root->left->left = createNode('*');
68     root->left->right = createNode('D');
69
70     root->right->left = createNode('E');
71     root->right->right = createNode('*');
72
73     root->left->left->left = createNode('+');
74     root->left->left->right = createNode('C');
75
76     root->right->right->left = createNode('F');
77     root->right->right->right = createNode('H');
78
79     root->left->left->left->left = createNode('A');
80     root->left->left->left->right = createNode('B');
81
82     cout << "\n1. Pre-order (Root-Left-Right):" << endl;
83     preOrder(root);
84
85     cout << "\n2. In-order (Left-Root-Right):" << endl;
86     inOrder(root);
87
88     cout << "\n3. Post-order (Left-Right-Root):" << endl;
89     postOrder(root);
90
91     cout << "\n4. Level-order (BFS):" << endl;
92     levelOrder(root);
93
94     cout << endl;
95     return 0;
96 }

```

Output

```

PS C:\shellyn\kuliah\semester3\strukturData\teori\tugasTree> cd "c:\shellyn\kuliah\semester3\strukturData\teori\tugasTree\" ; if ($?) { g++ exercise3.cpp -o exercise3 } ; if ($?) { .\exercise3 }

```

```

1. Pre-order (Root-Left-Right):
/ + * + A B C D + E * F H

```

```

2. In-order (Left-Root-Right):
A + B * C + D / E + F * H

```

```

3. Post-order (Left-Right-Root):
A B + C * D + E F H * + /

```

```

4. Level-order (BFS):
/ + + * D E * + C F H A B

```

```

PS C:\shellyn\kuliah\semester3\strukturData\teori\tugasTree>

```

```

1  #include <iostream>
2  #include <queue>
3
4  using namespace std;
5
6  struct Node {
7      char info;
8      Node* left;
9      Node* right;
10 };
11
12 Node* createNode(char data) {
13     Node* newNode = new Node();
14     newNode->info = data;
15     newNode->left = NULL;
16     newNode->right = NULL;
17     return newNode;
18 }
19
20 void preOrder(Node* root) {
21     if (root != NULL) {
22         cout << root->info << " ";
23         preOrder(root->left);
24         preOrder(root->right);
25     }
26 }
27
28 void inOrder(Node* root) {
29     if (root != NULL) {
30         inOrder(root->left);
31         cout << root->info << " ";
32         inOrder(root->right);
33     }
34 }
35
36 void postOrder(Node* root) {
37     if (root != NULL) {
38         postOrder(root->left);
39         postOrder(root->right);
40         cout << root->info << " ";
41     }
42 }
43
44 void levelOrder(Node* root) {
45     if (root == NULL) return;
46
47     queue<Node*> q;
48     q.push(root);
49
50     while (!q.empty()) {
51         Node* current = q.front();
52         q.pop();
53
54         cout << current->info << " ";
55
56         if (current->left != NULL) q.push(current->left);
57         if (current->right != NULL) q.push(current->right);
58     }
59 }
60

```

Exercise 4

```

61 int main() {
62     Node* root = createNode('A');
63
64     root->left = createNode('B');
65     root->right = createNode('C');
66
67     root->left->left = createNode('D');
68     root->left->right = createNode('E');
69
70     root->right->left = createNode('G');
71     root->right->right = createNode('H');
72
73     root->left->left->left = createNode('I');
74
75     root->left->right->left = createNode('J');
76     root->left->right->right = createNode('K');
77
78     root->right->left->left = createNode('L');
79     root->right->left->right = createNode('M');
80
81     root->right->left->right->left = createNode('N');
82     root->right->left->right->right = createNode('O');
83
84     cout << "\n1. Pre-order (Root-Left-Right):" << endl;
85     preOrder(root);
86
87     cout << "\n2. In-order (Left-Root-Right):" << endl;
88     inOrder(root);
89
90     cout << "\n3. Post-order (Left-Right-Root):" << endl;
91     postOrder(root);
92
93     cout << "\n4. Level-order (BFS):" << endl;
94     levelOrder(root);
95
96     cout << endl;
97     return 0;
98 }

```

Output

● PS C:\shellyn\kuliah\semester3\strukturData\teori\tugasTree> cd "c:\shellyn\kuliah\semester3\strukturData\teori\tugasTree"; if (\$?) { g++ exercise4.cpp -o exercise4 }; if (\$?) { .\exercise4 }

1. Pre-order (Root-Left-Right):
A B D I E J K C G L M N O H

2. In-order (Left-Root-Right):
I D B J E K A L G N M O C H

3. Post-order (Left-Right-Root):
I D J K E B L N O M G H C A

4. Level-order (BFS):
A B C D E G H I J K L M N O

○ PS C:\shellyn\kuliah\semester3\strukturData\teori\tugasTree> □