SAURABH KALA SAP ID – 590014968

Lab Experiment: 09 Batch: 1
Subject: Data Structures Lab MCA

Semester: 1st

Objective:

To understand the structure and implementation of binary trees using arrays and linked lists.

- To perform various tree traversal techniques (in-order, pre-order, post-order, and level-order).
- To implement heap sorting using a binary tree structure.

Instructions:

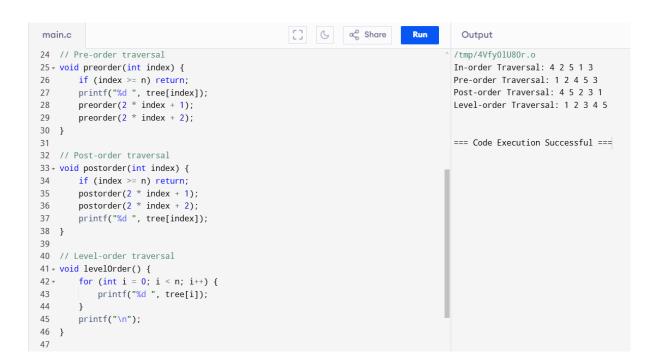
Implement the following tasks in C. Use appropriate data structures (array or linked list) to create the binary tree and demonstrate traversal methods and heap sorting. 1st Assignment:

Binary Tree Creation Using Arrays:

- Represent a complete binary tree using an array.
- Note that for a node at index i:
- The left child is at 2 * i + 1
- The right child is at 2 * i + 2 Using Linked Lists:
- Represent a binary tree where each node contains data and pointers to its left and right children.
- Include functions to create and insert nodes in the binary tree

Binary Tree Using Arrays

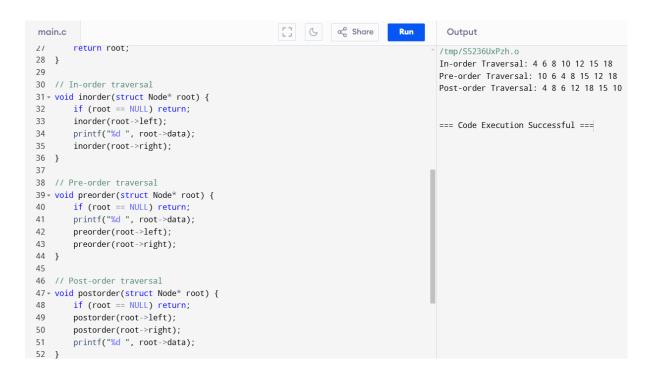
```
Run
                                                                                   Output
main.c
 1 #include <stdio.h>
                                                                                  /tmp/4Vfy0lU80r.o
                                                                                  In-order Traversal: 4 2 5 1 3
 2
 3 #define MAX_SIZE 100
                                                                                  Pre-order Traversal: 1 2 4 5 3
                                                                                 Post-order Traversal: 4 5 2 3 1
 5 int tree[MAX_SIZE];
                                                                                 Level-order Traversal: 1 2 3 4 5
 6 int n = 0; // Current size of the tree
 8 // Function to insert an element into the tree
                                                                                 === Code Execution Successful ===
9 - void insert(int value) {
10 * if (n < MAX_SIZE) {
11
           tree[n] = value;
           n++;
13
       }
14 }
15
16 // In-order traversal
17 - void inorder(int index) {
18
       if (index >= n) return;
       inorder(2 * index + 1); // Left child
19
       printf("%d ", tree[index]);
20
21
       inorder(2 * index + 2); // Right child
22 }
23
24 // Pre-order traversal
25 - void preorder(int index) {
```



```
[] ( c Share
main.c
                                                                          Run
                                                                                     Output
                                                                                   /tmp/4Vfy0lU80r.o
47
                                                                                   In-order Traversal: 4 2 5 1 3
48 - int main() {
                                                                                   Pre-order Traversal: 1 2 4 5 3
        // Inserting elements into the binary tree
49
                                                                                   Post-order Traversal: 4 5 2 3 1
        insert(1);
50
                                                                                   Level-order Traversal: 1 2 3 4 5
51
        insert(2);
        insert(3);
52
53
        insert(4);
                                                                                   === Code Execution Successful ===
        insert(5);
54
55
        printf("In-order Traversal: ");
56
        inorder(0);
57
        printf("\n");
58
59
        printf("Pre-order Traversal: ");
60
        preorder(0);
61
62
        printf("\n");
63
        printf("Post-order Traversal: ");
64
        postorder(0);
65
        printf("\n");
66
67
        printf("Level-order Traversal: ");
68
69
        levelOrder();
70
71
        return 0;
```

Binary Tree Using Linked Lists

```
main.c
                                                                                   Output
 1 #include <stdio.h>
                                                                                  /tmp/S5236UxPzh.o
2 #include <stdlib.h>
                                                                                  In-order Traversal: 4 6 8 10 12 15 18
                                                                                  Pre-order Traversal: 10 6 4 8 15 12 18
4 // Define the structure for a tree node
                                                                                  Post-order Traversal: 4 8 6 12 18 15 10
5 - struct Node {
6
      int data;
       struct Node *left, *right;
                                                                                  === Code Execution Successful ===
8 };
10 // Function to create a new node
11 - struct Node* createNode(int data) {
       struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
13
       newNode->data = data;
       newNode->left = newNode->right = NULL;
14
15
       return newNode;
17
18 // Function to insert nodes in a binary tree (manually for simplicity)
19 * struct Node* insertNode(struct Node* root, int data) {
       if (root == NULL) return createNode(data);
21
22
       if (data < root->data)
23
          root->left = insertNode(root->left, data);
24
25
          root->right = insertNode(root->right, data);
```



```
[] ⟨ □ ∞ Share
                                                                                        Output
main.c
                                                                                       /tmp/S5236UxPzh.o
53
                                                                                       In-order Traversal: 4 6 8 10 12 15 18
54 * int main() {
                                                                                       Pre-order Traversal: 10 6 4 8 15 12 18
Post-order Traversal: 4 8 6 12 18 15 10
      struct Node* root = NULL;
56
       root = insertNode(root, 10);
57
       insertNode(root, 6);
58
                                                                                       === Code Execution Successful ===
59
        insertNode(root, 15);
      insertNode(root, 4);
60
       insertNode(root, 8);
61
62
        insertNode(root, 12);
       insertNode(root, 18);
63
64
       printf("In-order Traversal: ");
65
       inorder(root);
66
       printf("\n");
67
68
69
       printf("Pre-order Traversal: ");
       preorder(root);
70
       printf("\n");
71
72
73
       printf("Post-order Traversal: ");
       postorder(root);
74
75
       printf("\n");
76
       return 0;
77
```

Heap Sort using Binary Tree

```
[] G & Share
                                                                        Run
main.c
                                                                                   Output
1 #include <stdio.h>
                                                                                  /tmp/rg92H5wqn3.o
                                                                                  Heap Sort: 1 2 3 4 5 6
3 #define MAX_SIZE 100
5 int heap[MAX_SIZE];
                                                                                  === Code Execution Successful ===
6 int size = 0;
8 // Function to swap two elements
9 void swap(int *a, int *b) {
      int temp = *a;
10
11
       *a = *b;
      *b = temp;
12
13 }
14
15 // Heapify function to maintain max-heap property
16 - void heapify(int i) {
17
       int largest = i;
18
       int left = 2 * i + 1;
      int right = 2 * i + 2;
19
20
21
       if (left < size && heap[left] > heap[largest])
22
           largest = left;
       if (right < size && heap[right] > heap[largest])
23
24
           largest = right;
25
```

```
[] ( c Share
                                                                            Run
                                                                                      Output
main.c
                                                                                     /tmp/rg92H5wqn3.o
26 +
        if (largest != i) {
                                                                                     Heap Sort: 1 2 3 4 5 6
            swap(&heap[i], &heap[largest]);
27
28
            heapify(largest);
29
                                                                                     === Code Execution Successful ===
30 }
31
32 // Insert an element into the heap
33 - void insertHeap(int value) {
      if (size < MAX_SIZE) {</pre>
35
           heap[size] = value;
           int i = size;
36
           size++;
37
38
           while (i != 0 && heap[(i - 1) / 2] < heap[i]) {</pre>
39 +
               swap(&heap[(i - 1) / 2], &heap[i]);
40
41
               i = (i - 1) / 2;
42
           }
43
       }
44 }
46 // Perform heap sort
47 • void heapSort() {
48 * for (int i = size - 1; i >= 0; i--) {
49
           swap(&heap[0], &heap[i]);
           size--;
50
```

```
Run
                                                                                Output
main.c
                                                                               /tmp/rg92H5wqn3.o
46 // Perform heap sort
                                                                               Heap Sort: 1 2 3 4 5 6
47 void heapSort() {
48 *  for (int i = size - 1; i >= 0; i--) {
       swap(&heap[0], &heap[i]);
49
50
                                                                               === Code Execution Successful ===
          size--;
51
          heapify(<mark>0</mark>);
52
53 }
55 - int main() {
       insertHeap(3);
57
       insertHeap(1);
      insertHeap(6);
58
     insertHeap(5);
59
     insertHeap(2);
60
61
      insertHeap(4);
62
63
       printf("Heap Sort: ");
       heapSort();
64
       for (int i = 0; i < 6; i++) {
65 +
           printf("%d ", heap[i]);
66
67
68
       printf("\n");
69
   return 0;
```

Explanation:

- Array Representation: Simple complete binary tree using an array.
- Linked List Representation: Binary tree using nodes with pointers.
- Heap Sort: Uses a max-heap to perform sorting.

2nd Assignment:

Tree Traversal Methods

Implement the following traversal methods: In-order Traversal:

• Traverse the left subtree, visit the root node, then traverse the right subtree.

Pre-order Traversal:

• Visit the root node, traverse the left subtree, then traverse the right subtree.

Post-order Traversal:

• Traverse the left subtree, traverse the right subtree, then visit the root node.

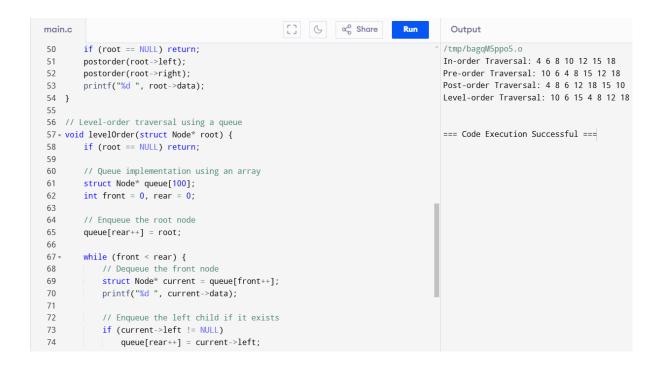
Level-order Traversal:

• Traverse the nodes level by level, starting from the root.

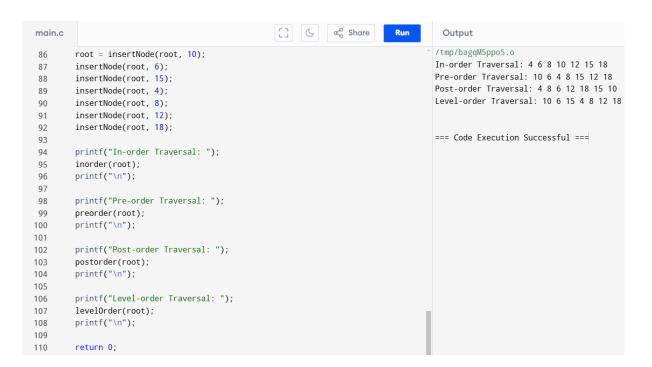
Implement each traversal function and test them with the binary tree created above.

```
[] G & Share
                                                                              Run
main.c
                                                                                          Output
 1 #include <stdio.h>
                                                                                         /tmp/bagqM5ppo5.o
 2 #include <stdlib.h>
                                                                                         In-order Traversal: 4 6 8 10 12 15 18
                                                                                         Pre-order Traversal: 10 6 4 8 15 12 18
 4 // Define the structure for a tree node
                                                                                         Post-order Traversal: 4 8 6 12 18 15 10
 5 struct Node {
                                                                                         Level-order Traversal: 10 6 15 4 8 12 18
 6
       int data:
       struct Node *left, *right;
 8 };
                                                                                         === Code Execution Successful ===
10 // Function to create a new node
11 * struct Node* createNode(int data) {
       struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
13
        newNode->data = data;
14
       newNode->left = newNode->right = NULL;
15
       return newNode;
16 }
17
18 // Function to insert nodes in a binary tree (manual insertion for testing)
19 * struct Node* insertNode(struct Node* root, int data) {
     if (root == NULL)
21
           return createNode(data);
22
23  // Simple binary search tree (BST) insertion for demonstration
24  if (data < root->data)
25  root->left = insertNode(root->left, data);
```

```
[] ⟨ ⟨ oco Share
                                                                         Run
main.c
                                                                                    Output
25
            root->left = insertNode(root->left, data);
                                                                                   /tmp/bagqM5ppo5.o
                                                                                   In-order Traversal: 4 6 8 10 12 15 18
26
        else
                                                                                   Pre-order Traversal: 10 6 4 8 15 12 18
            root->right = insertNode(root->right, data);
27
                                                                                   Post-order Traversal: 4 8 6 12 18 15 10
28
29
        return root;
                                                                                   Level-order Traversal: 10 6 15 4 8 12 18
30 }
31
                                                                                  === Code Execution Successful ===
32 // In-order traversal: Left -> Root -> Right
33 - void inorder(struct Node* root) {
       if (root == NULL) return;
34
35
        inorder(root->left);
36
       printf("%d ", root->data);
37
       inorder(root->right);
38 }
39
40 // Pre-order traversal: Root -> Left -> Right
41 - void preorder(struct Node* root) {
42
       if (root == NULL) return;
        printf("%d ", root->data);
43
44
        preorder(root->left);
45
        preorder(root->right);
46 }
47
48 // Post-order traversal: Left -> Right -> Root
49 - void postorder(struct Node* root) {
```



```
[] G & Share
                                                                          Run
main.c
                                                                                      Output
                                                                                    /tmp/bagqM5ppo5.o
75
                                                                                    In-order Traversal: 4 6 8 10 12 15 18
            // Enqueue the right child if it exists
76
                                                                                    Pre-order Traversal: 10 6 4 8 15 12 18
77
            if (current->right != NULL)
                                                                                    Post-order Traversal: 4 8 6 12 18 15 10
78
                queue[rear++] = current->right;
                                                                                    Level-order Traversal: 10 6 15 4 8 12 18
79
80 }
81
                                                                                    === Code Execution Successful ===
82 - int main() {
83
        struct Node* root = NULL;
84
85
        // Manually creating a binary tree for testing
86
        root = insertNode(root, 10);
        insertNode(root, 6);
87
        insertNode(root, 15);
88
        insertNode(root, 4);
89
90
        insertNode(root, 8);
91
        insertNode(root, 12);
        insertNode(root, 18);
92
93
94
        printf("In-order Traversal: ");
95
        inorder(root);
96
        printf("\n");
97
98
        printf("Pre-order Traversal: ");
99
        preorder(root);
```



Explanation of the Code:

1. Node Structure:

- Each node contains:
 - An integer data
 - A pointer to the left child
 - A pointer to the right child

2. Node Insertion:

• The insertNode function inserts nodes into a Binary Search Tree (BST).

3. Traversal Functions:

- o In-order Traversal (inorder): Recursively traverses the left subtree, visits the root, then traverses the right subtree.
- Pre-order Traversal (preorder): Visits the root, then recursively traverses the left and right subtrees.
- Post-order Traversal (postorder): Recursively traverses the left and right subtrees, then visits the root.
- Level-order Traversal (levelOrder): Uses a queue to traverse the nodes level by level.

4. Level-order Traversal Implementation:

o We use a simple array-based queue to implement level-order traversal