

## Binary Tree creation using Arrays

main.c	Output
<pre>1 #include &lt;stdio.h&gt; 2 3 #define MAX_SIZE 100 4 5 // Structure for the Binary Tree 6 struct BinaryTree { 7     int data; 8 }; 9 10 // Function to print the tree 11 void printTree(struct BinaryTree tree[], int size) { 12     for (int i = 0; i &lt; size; i++) { 13         printf("Node %d: %d\n", i, tree[i].data); 14     } 15 } 16 17 // Main function 18 int main() { 19     struct BinaryTree tree[MAX_SIZE]; // Array representing the tree 20     int size = 7; // Number of nodes in the tree 21 22     // Assigning data to nodes 23     tree[0].data = 1; // Root node 24     tree[1].data = 2; // Left child of node 0 25     tree[2].data = 3; // Right child of node 0 26     tree[3].data = 4; // Left child of node 1 27     tree[4].data = 5; // Right child of node 1 28     tree[5].data = 6; // Left child of node 2 29     tree[6].data = 7; // Right child of node 2 30 31     // Print the tree structure 32     printTree(tree, size); 33 34     return 0; 35 } 36</pre>	<pre>/tmp/kNBHlyMuOM.o Node 0: 1 Node 1: 2 Node 2: 3 Node 3: 4 Node 4: 5 Node 5: 6 Node 6: 7  === Code Execution Successful ===</pre>

## Binary Tree creation using Linked Lists

main.c	Output
<pre>1 #include &lt;stdio.h&gt; 2 #include &lt;stdlib.h&gt; 3 4 // Structure for a Binary Tree Node 5 struct Node { 6     int data; 7     struct Node* left; 8     struct Node* right; 9 }; 10 11 // Function to create a new node 12 struct Node* createNode(int data) { 13     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node)); 14     newNode-&gt;data = data; 15     newNode-&gt;left = newNode-&gt;right = NULL; 16     return newNode; 17 } 18 19 // Function to insert a node in the tree 20 void insert(struct Node** root, int data) { 21     if (*root == NULL) { 22         *root = createNode(data); 23     } else { 24         if (data &lt; (*root)-&gt;data) { 25             insert(&amp;(*root)-&gt;left, data); 26         } else { 27             insert(&amp;(*root)-&gt;right, data); 28         } 29     } 30 } 31 32 // Function to print the tree (In-Order Traversal) 33 void printTree(struct Node* root) { 34     if (root != NULL) { 35         printTree(root-&gt;left); 36         printf("%d ", root-&gt;data); 37         printTree(root-&gt;right); 38     } 39 }</pre>	<pre>/tmp/pGUQdg16EP.o In-Order Traversal: 1 2 3 4 5 6 7  === Code Execution Successful ===</pre>

main.c	Output
<pre>21- if (*root == NULL) { 22-     *root = createNode(data); 23- } else { 24-     if (data &lt; (*root)-&gt;data) { 25-         insert(&amp;(*root)-&gt;left, data); 26-     } else { 27-         insert(&amp;(*root)-&gt;right, data); 28-     } 29- } 30 } 31 32 // Function to print the tree (In-Order Traversal) 33 void printTree(struct Node* root) { 34-     if (root != NULL) { 35-         printTree(root-&gt;left); 36-         printf("%d ", root-&gt;data); 37-         printTree(root-&gt;right); 38-     } 39 } 40 41 // Main function 42 int main() { 43     struct Node* root = NULL; 44 45     // Inserting nodes into the binary tree 46     insert(&amp;root, 4); // Root node 47     insert(&amp;root, 2); // Left child of node 4 48     insert(&amp;root, 6); // Right child of node 4 49     insert(&amp;root, 1); // Left child of node 2 50     insert(&amp;root, 3); // Right child of node 2 51     insert(&amp;root, 5); // Left child of node 6 52     insert(&amp;root, 7); // Right child of node 6 53 54     // Print the tree 55     printf("In-Order Traversal: "); 56     printTree(root); 57     printf("\n"); 58 59     return 0;</pre>	<pre>/tmp/pgUQdg16EP.o In-Order Traversal: 1 2 3 4 5 6 7  === Code Execution Successful ===</pre>

## In-Order, Pre-Order, Post-Order Traversal

main.c	Output
<pre>1 #include &lt;stdio.h&gt; 2 #include &lt;stdlib.h&gt; 3 4 // Definition of a Node in the Binary Tree 5 struct Node { 6     int data; 7     struct Node *left, *right; 8 }; 9 10 // Function to create a new Node 11 struct Node* createNode(int data) { 12     struct Node* node = malloc(sizeof(struct Node)); 13     node-&gt;data = data; 14     node-&gt;left = node-&gt;right = NULL; 15     return node; 16 } 17 18 // In-order Traversal: Left, Root, Right 19 void inOrder(struct Node* root) { 20-     if (root != NULL) { 21-         inOrder(root-&gt;left); 22-         printf("%d ", root-&gt;data); 23-         inOrder(root-&gt;right); 24-     } 25 } 26 27 // Pre-order Traversal: Root, Left, Right 28 void preOrder(struct Node* root) { 29-     if (root != NULL) { 30-         printf("%d ", root-&gt;data); 31-         preOrder(root-&gt;left); 32-         preOrder(root-&gt;right); 33-     } 34 } 35 36 // Post-order Traversal: Left, Right, Root 37 void postOrder(struct Node* root) { 38-     if (root != NULL) { 39-         postOrder(root-&gt;left);</pre>	<pre>/tmp/ES3Ibgtmn0.o In-order Traversal: 4 2 5 1 6 3 7 Pre-order Traversal: 1 2 4 5 3 6 7 Post-order Traversal: 4 5 2 6 7 3 1 Level-order Traversal: 1  === Code Execution Successful ===</pre>

main.c	Output
<pre>39     postOrder(root-&gt;left); 40     postOrder(root-&gt;right); 41     printf("%d ", root-&gt;data); 42 } 43 } 44 45 // Level-order Traversal: Using a queue (Breadth-First Search) 46 struct QueueNode { 47     struct Node* node; 48     struct QueueNode* next; 49 }; 50 51 // Function to enqueue a node 52 void enqueue(struct QueueNode** front, struct QueueNode** rear, struct Node* node) { 53     struct QueueNode* newNode = malloc(sizeof(struct QueueNode)); 54     newNode-&gt;node = node; 55     newNode-&gt;next = NULL; 56     if (*rear == NULL) { 57         *front = *rear = newNode; 58     } else { 59         (*rear)-&gt;next = newNode; 60         *rear = newNode; 61     } 62 } 63 64 // Function to dequeue a node 65 struct Node* dequeue(struct QueueNode** front) { 66     if (*front == NULL) return NULL; 67     struct QueueNode* temp = *front; 68     struct Node* node = temp-&gt;node; 69     *front = (*front)-&gt;next; 70     free(temp); 71     return node; 72 } 73 74 // Level-order Traversal: Using queue for BFS 75 void levelOrder(struct Node* root) { 76     if (root == NULL) return; 77 }</pre>	<pre>/tmp/ES3IbgtmnQ.o In-order Traversal: 4 2 5 1 6 3 7 Pre-order Traversal: 1 2 4 5 3 6 7 Post-order Traversal: 4 5 2 6 7 3 1 Level-order Traversal: 1  === Code Execution Successful ===</pre>

main.c	Output
<pre>72 } 73 74 // Level-order Traversal: Using queue for BFS 75 void levelOrder(struct Node* root) { 76     if (root == NULL) return; 77 78     struct QueueNode* front = NULL; 79     struct QueueNode* rear = NULL; 80 81     enqueue(&amp;front, &amp;rear, root); // Enqueue root 82 83     while (front != NULL) { 84         struct Node* node = dequeue(&amp;front); 85         printf("%d ", node-&gt;data); 86 87         if (node-&gt;left) enqueue(&amp;front, &amp;rear, node-&gt;left); 88         if (node-&gt;right) enqueue(&amp;front, &amp;rear, node-&gt;right); 89     } 90 } 91 92 // Main function to create the tree and test traversal methods 93 int main() { 94     struct Node* root = createNode(1); 95     root-&gt;left = createNode(2); 96     root-&gt;right = createNode(3); 97     root-&gt;left-&gt;left = createNode(4); 98     root-&gt;left-&gt;right = createNode(5); 99     root-&gt;right-&gt;left = createNode(6); 100    root-&gt;right-&gt;right = createNode(7); 101 102    printf("In-order Traversal: "); 103    inOrder(root); 104    printf("\n"); 105 106    printf("Pre-order Traversal: "); 107    preOrder(root); 108    printf("\n");</pre>	<pre>/tmp/ES3IbgtmnQ.o In-order Traversal: 4 2 5 1 6 3 7 Pre-order Traversal: 1 2 4 5 3 6 7 Post-order Traversal: 4 5 2 6 7 3 1 Level-order Traversal: 1  === Code Execution Successful ===</pre>

main.c	Run	Output
<pre>80 81 enqueue(&amp;front, &amp;rear, root); // Enqueue root 82 83 while (front != NULL) { 84     struct Node* node = dequeue(&amp;front); 85     printf("%d ", node-&gt;data); 86 87     if (node-&gt;left) enqueue(&amp;front, &amp;rear, node-&gt;left); 88     if (node-&gt;right) enqueue(&amp;front, &amp;rear, node-&gt;right); 89 } 90 } 91 92 // Main function to create the tree and test traversal methods 93 int main() { 94     struct Node* root = createNode(1); 95     root-&gt;left = createNode(2); 96     root-&gt;right = createNode(3); 97     root-&gt;left-&gt;left = createNode(4); 98     root-&gt;left-&gt;right = createNode(5); 99     root-&gt;right-&gt;left = createNode(6); 100    root-&gt;right-&gt;right = createNode(7); 101 102    printf("In-order Traversal: "); 103    inOrder(root); 104    printf("\n"); 105 106    printf("Pre-order Traversal: "); 107    preOrder(root); 108    printf("\n"); 109 110    printf("Post-order Traversal: "); 111    postOrder(root); 112    printf("\n"); 113 114    printf("Level-order Traversal: "); 115    levelOrder(root); 116    printf("\n"); 117 118    return 0; </pre>	<div>Run</div>	<pre>/tmp/ES3IbgtmnQ.o In-order Traversal: 4 2 5 1 6 3 7 Pre-order Traversal: 1 2 4 5 3 6 7 Post-order Traversal: 4 5 2 6 7 3 1 Level-order Traversal: 1  === Code Execution Successful === </pre>