Binary Tree creation using Arrays

main.c

```c
#include <stdio.h>

#define MAX_SIZE 100

// Structure for the Binary Tree
struct BinaryTree {
    int data;
};

// Function to print the tree
void printTree(struct BinaryTree tree[], int size) {
    for (int i = 0; i < size; i++) {
        printf("Node %d: %d\n", i, tree[i].data);
    }
}

// Main function
int main() {
    struct BinaryTree tree[MAX_SIZE];  // Array representing the tree
    int size = 7;  // Number of nodes in the tree

    // Assigning data to nodes
    tree[0].data = 1;  // Root node
    tree[1].data = 2;  // Left child of node 0
    tree[2].data = 3;  // Right child of node 0
    tree[3].data = 4;  // Left child of node 1
    tree[4].data = 5;  // Right child of node 1
    tree[5].data = 6;  // Left child of node 2
    tree[6].data = 7;  // Right child of node 2

    // Print the tree structure
    printTree(tree, size);

    return 0;
}
```

Output

```
/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 ===
```

Binary Tree creation using Linked Lists

main.c

```c
#include <stdio.h>
#include <stdlib.h>

// Structure for a Binary Tree Node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Function to insert a node in the tree
void insert(struct Node** root, int data) {
    if (*root == NULL) {
        *root = createNode(data);
    } else {
        if (data < (*root)->data) {
            insert(&(*root)->left, data);
        } else {
            insert(&(*root)->right, data);
        }
    }
}

// Function to print the tree (In-Order Traversal)
void printTree(struct Node* root) {
    if (root != NULL) {
        printTree(root->left);
        printf("%d ", root->data);
        printTree(root->right);
    }
}
```

Output

```
/tmp/pGUQdgl6EP.o
In-Order Traversal: 1 2 3 4 5 6 7


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

main.c

```c
    if (*root == NULL) {
        *root = createNode(data);
    } else {
        if (data < (*root)->data) {
            insert(&(*root)->left, data);
        } else {
            insert(&(*root)->right, data);
        }
    }
}

// Function to print the tree (In-Order Traversal)
void printTree(struct Node* root) {
    if (root != NULL) {
        printTree(root->left);
        printf("%d ", root->data);
        printTree(root->right);
    }
}

// Main function
int main() {
    struct Node* root = NULL;

    // Inserting nodes into the binary tree
    insert(&root, 4);  // Root node
    insert(&root, 2);  // Left child of node 4
    insert(&root, 6);  // Right child of node 4
    insert(&root, 1);  // Left child of node 2
    insert(&root, 3);  // Right child of node 2
    insert(&root, 5);  // Left child of node 6
    insert(&root, 7);  // Right child of node 6

    // Print the tree
    printf("In-Order Traversal: ");
    printTree(root);
    printf("\n");

    return 0;
```

Output

```
/tmp/pGUQdgl6EP.o
In-Order Traversal: 1 2 3 4 5 6 7


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

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

main.c

```c
#include <stdio.h>
#include <stdlib.h>

// Definition of a Node in the Binary Tree
struct Node {
    int data;
    struct Node *left, *right;
};

// Function to create a new Node
struct Node* createNode(int data) {
    struct Node* node = malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// In-order Traversal: Left, Root, Right
void inOrder(struct Node* root) {
    if (root != NULL) {
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
    }
}

// Pre-order Traversal: Root, Left, Right
void preOrder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preOrder(root->left);
        preOrder(root->right);
    }
}

// Post-order Traversal: Left, Right, Root
void postOrder(struct Node* root) {
    if (root != NULL) {
        postOrder(root->left);
```

Output

```
/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 ===
```

main.c

Share Run

```c
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ", root->data);
    }
}

// Level-order Traversal: Using a queue (Breadth-First Search)
struct QueueNode {
    struct Node* node;
    struct QueueNode* next;
};

// Function to enqueue a node
void enqueue(struct QueueNode** front, struct QueueNode** rear, struct Node* node) {
    struct QueueNode* newNode = malloc(sizeof(struct QueueNode));
    newNode->node = node;
    newNode->next = NULL;
    if (*rear == NULL) {
        *front = *rear = newNode;
    } else {
        (*rear)->next = newNode;
        *rear = newNode;
    }
}

// Function to dequeue a node
struct Node* dequeue(struct QueueNode** front) {
    if (*front == NULL) return NULL;
    struct QueueNode* temp = *front;
    struct Node* node = temp->node;
    *front = (*front)->next;
    free(temp);
    return node;
}

// Level-order Traversal: Using queue for BFS
void levelOrder(struct Node* root) {
    if (root == NULL) return;
```

Output

```
/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 ===
```

main.c

Share Run

```c
}

// Level-order Traversal: Using queue for BFS
void levelOrder(struct Node* root) {
    if (root == NULL) return;

    struct QueueNode* front = NULL;
    struct QueueNode* rear = NULL;

    enqueue(&front, &rear, root);  // Enqueue root

    while (front != NULL) {
        struct Node* node = dequeue(&front);
        printf("%d ", node->data);

        if (node->left) enqueue(&front, &rear, node->left);
        if (node->right) enqueue(&front, &rear, node->right);
    }
}

// Main function to create the tree and test traversal methods
int main() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);
    root->right->right = createNode(7);

    printf("In-order Traversal: ");
    inOrder(root);
    printf("\n");

    printf("Pre-order Traversal: ");
    preOrder(root);
    printf("\n");
```

Output

```
/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 ===
```

main.c

Share Run

```c

    enqueue(&front, &rear, root);  // Enqueue root

    while (front != NULL) {
        struct Node* node = dequeue(&front);
        printf("%d ", node->data);

        if (node->left) enqueue(&front, &rear, node->left);
        if (node->right) enqueue(&front, &rear, node->right);
    }
}

// Main function to create the tree and test traversal methods
int main() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);
    root->right->right = createNode(7);

    printf("In-order Traversal: ");
    inOrder(root);
    printf("\n");

    printf("Pre-order Traversal: ");
    preOrder(root);
    printf("\n");

    printf("Post-order Traversal: ");
    postOrder(root);
    printf("\n");

    printf("Level-order Traversal: ");
    levelOrder(root);
    printf("\n");

    return 0;
```

Output

```
/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 ===
```