二叉树构建与遍历代码文档

本文档旨在详细说明上述C语言代码的功能和执行过程，包括二叉树的构建以及前序、中序和后序遍历。

1. 代码简介

上述C代码实现了以下功能：

- 定义了二叉树结构 `struct TreeNode`，包括数据成员 `data` 以及左右子树指针 `left` 和 `right`。
- 提供了创建新二叉树节点的函数 `createNode`，用于分配内存并初始化新节点的数据和指针。
- 提供了构造二叉树的函数 `buildTree`，该函数根据扩展前序遍历序列构建二叉树。
- 提供了前序遍历 `preorderTraversal`、中序遍历 `inorderTraversal` 和后序遍历 `postorderTraversal` 的函数，用于遍历二叉树并输出遍历结果。
- 主函数 `main` 中演示了如何使用这些函数来构建二叉树并执行不同遍历方式。

2. 二叉树构建过程

2.1. `struct TreeNode` 结构

在代码中，定义了二叉树节点的数据结构如下：

```
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
```

每个节点包含一个整数 `data` 作为节点的数据，以及两个指针 `left` 和 `right`，分别指向左子树和右子树。

2.2. `createNode` 函数

`createNode` 函数用于创建一个新的二叉树节点。它分配了内存以存储节点，初始化数据为提供的值，以及将左右子树指针设置为 `NULL`。代码如下：

```
struct TreeNode* createNode(int data) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
```

2.3. `buildTree` 函数

`buildTree` 函数用于构建二叉树。它使用扩展前序遍历序列作为输入。扩展前序遍历是一种遍历方式，其中 `-1` 表示空节点，非 `-1` 的值表示节点的数据。这个函数通过递归方式构建二叉树，代码如下：

```
struct TreeNode* buildTree(int* preorder, int* index) {
    if (preorder[*index] == -1) {
        (*index)++;
        return NULL;
    }

    struct TreeNode* root = createNode(preorder[(*index)++]);
    root->left = buildTree(preorder, index);
    root->right = buildTree(preorder, index);

    return root;
}
```

2.4. 二叉树构建示例

在主函数中，演示了如何使用`buildTree`函数构建二叉树，其中示例扩展前序遍历序列为：

```
int preorder[] = {1, 2, 4, -1, -1, 5, -1, -1, 3, 6, -1, -1, 7, -1, -1};
```

函数会递归地创建二叉树节点，并返回根节点的指针。

3. 二叉树遍历过程

3.1. 前序遍历

前序遍历是一种遍历方式，首先访问根节点，然后按左子树、右子树的顺序递归遍历。前序遍历函数`preorderTraversal`如下：

```
void preorderTraversal(struct TreeNode* root) {
    if (root == NULL) return;
    printf("%d ", root->data);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}
```

3.2. 中序遍历

中序遍历是一种遍历方式，首先遍历左子树，然后访问根节点，最后遍历右子树。中序遍历函数`inorderTraversal`如下：

```
void inorderTraversal(struct TreeNode* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}
```

3.3. 后序遍历

后序遍历是一种遍历方式，首先遍历左子树，然后遍历右子树，最后访问根节点。后序遍历函数 `postorderTraversal` 如下：

```
void postorderTraversal(struct TreeNode* root) {
    if (root == NULL) return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d ", root->data);
}
```

4. 主函数

在主函数中，扩展前序遍历序列被用于构建二叉树，并示范了如何执行前序、中序和后序遍历，以及输出遍历结果。

5. 结论

以上是该C代码的功能和执行过程的详细解释。该代码演示了如何构建一个二叉树，以及如何对该二叉树进行前序、中序和后序遍历。这对于理解二叉树的构建和遍历过程非常有帮助。