109.Convert Sorted List to Binary Search Tree

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

/* Definition for singly-linked list */
struct ListNode {
    int val;
    struct ListNode *next;
};

/* Definition for binary tree node */
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};

/* Create a new list node */
struct ListNode* createListNode(int val) {
    struct ListNode* node = (struct ListNode*)malloc(sizeof(struct ListNode));
    node->val = val;
    node->next = NULL;
    return node;
}

/* Create a new tree node */
struct TreeNode* createTreeNode(int val) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = node->right = NULL;
    return node;
```

```
}

/* Function to display linked list */
void displayList(struct ListNode* head) {
    struct ListNode* temp = head;
    while (temp != NULL) {
        printf("%d-> ", temp->val);
        temp = temp->next;
    }
    printf("NULL\n");
}

/* Function to create sorted linked list from user input */
struct ListNode* createSortedList(int n) {
    int val;
    struct ListNode *head = NULL, *tail = NULL;
    for (int i = 0; i < n; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &val);
        struct ListNode* node = createListNode(val);
        if (head == NULL) {
            head = tail = node;
        } else {
            tail->next = node;
            tail = node;
        }
    }
    return head;
}

/* Convert sorted linked list to height-balanced BST */
```

```
struct TreeNode* sortedListToBST(struct ListNode* head) {
    if (head == NULL) return NULL;

    if (head->next == NULL) {
        struct TreeNode* root = createTreeNode(head->val);
        return root;
    }

    /* Find middle using slow-fast pointer */
    struct ListNode *slow = head, *fast = head, *prev = NULL;

    while (fast != NULL && fast->next != NULL) {
        prev = slow;
        slow = slow->next;
        fast = fast->next->next;
    }

    /* Disconnect left half */
    prev->next = NULL;

    /* Middle element becomes root */
    struct TreeNode* root = createTreeNode(slow->val);

    /* Recursively build left and right subtrees */
    root->left = sortedListToBST(head);
    root->right = sortedListToBST(slow->next);

    return root;
}

/* Tree traversal functions */
```

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

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

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

int main() {
    int n;
    printf("Enter number of elements in sorted linked list: ");
    scanf("%d", &n);

    printf("Enter sorted elements:\n");
    struct ListNode* head = createSortedList(n);

    printf("Linked list:\n");
    displayList(head);
```

```
    struct TreeNode* bstRoot = sortedListToBST(head);

    printf("\nBST In-order Traversal: ");
    inorder(bstRoot);
    printf("\nBST Pre-order Traversal: ");
    preorder(bstRoot);
    printf("\nBST Post-order Traversal: ");
    postorder(bstRoot);
    printf("\n");

    return 0;
}
```

OUTPUT: