Copilot

Yes, you’re right. It can be beneficial to define a `typedef` for the linked list as well. This can make the code more readable and easier to manage, especially when working with complex data structures. Here’s how you might do it:

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

// Define the Node data structure
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Define the LinkedList data structure
typedef struct LinkedList {
    Node* head;
} LinkedList;

// Function to insert a node at the end of the linked list
void insert(LinkedList* list, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;

    if (list->head == NULL) {
        list->head = newNode;
    } else {
        Node* last = list->head;
        while (last->next != NULL) {
            last = last->next;
        }
        last->next = newNode;
    }
}

// Function to print the linked list
void printList(LinkedList* list) {
    Node* node = list->head;
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
}

int main() {
    LinkedList list;
    list.head = NULL;

    insert(&list, 1);
    insert(&list, 2);
    insert(&list, 3);
    insert(&list, 4);
    insert(&list, 5);

    printList(&list);

    return 0;
}
```

In this version of the code, a `LinkedList` structure is defined, which contains a `head` pointer to the first node in the list. The `insert` and `printList` functions are updated to take a pointer to a `LinkedList` instead of a pointer to a `Node`. This makes it clear that these functions operate on an entire list rather than on individual nodes. The `main` function creates a `LinkedList`, adds nodes to it, and then prints it. This approach can make the code more intuitive and easier to understand. It also provides a clear structure for expanding the linked list functionality in the future. For example, you could easily add a `size` field to the `LinkedList` structure to keep track of the number of nodes in the list.