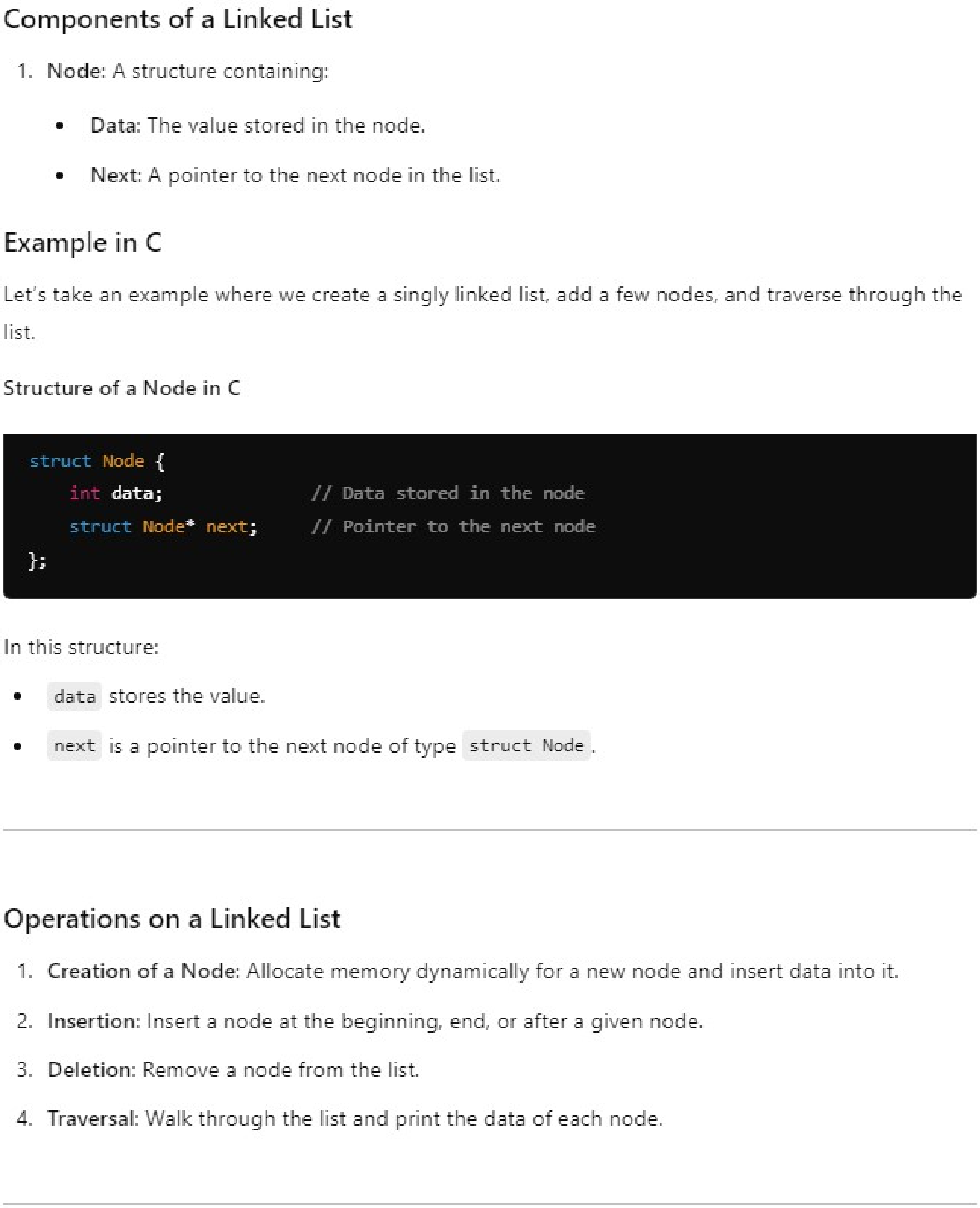
SINGLY LINKED LIST



Components of a Linked List

1. **Node**: A structure containing:

- **Data**: The value stored in the node.
- **Next**: A pointer to the next node in the list.

Example in C

Let's take an example where we create a singly linked list, add a few nodes, and traverse through the list.

Structure of a Node in C

```c
struct Node {
    int data;              // Data stored in the node
    struct Node* next;     // Pointer to the next node
};
```

In this structure:

- `data` stores the value.
- `next` is a pointer to the next node of type `struct Node`.

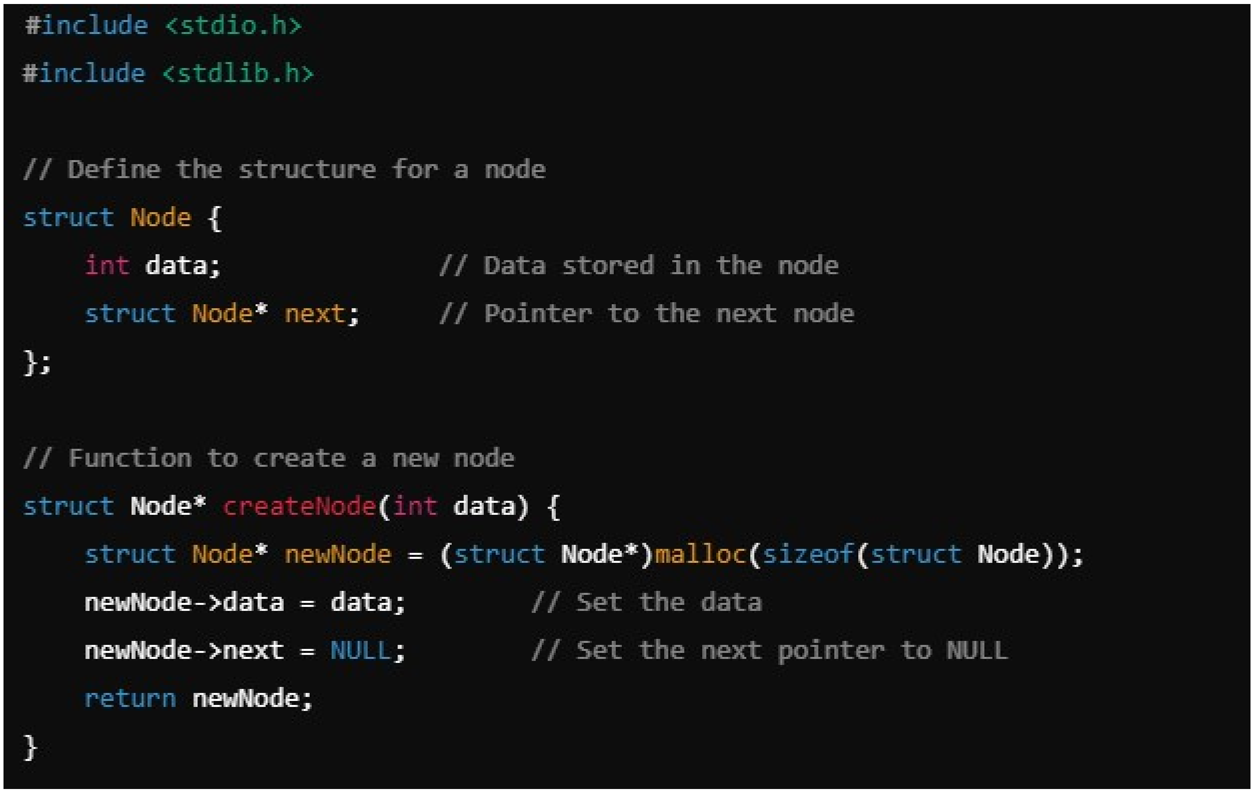
Operations on a Linked List

1. **Creation of a Node**: Allocate memory dynamically for a new node and insert data into it.
2. **Insertion**: Insert a node at the beginning, end, or after a given node.
3. **Deletion**: Remove a node from the list.
4. **Traversal**: Walk through the list and print the data of each node.

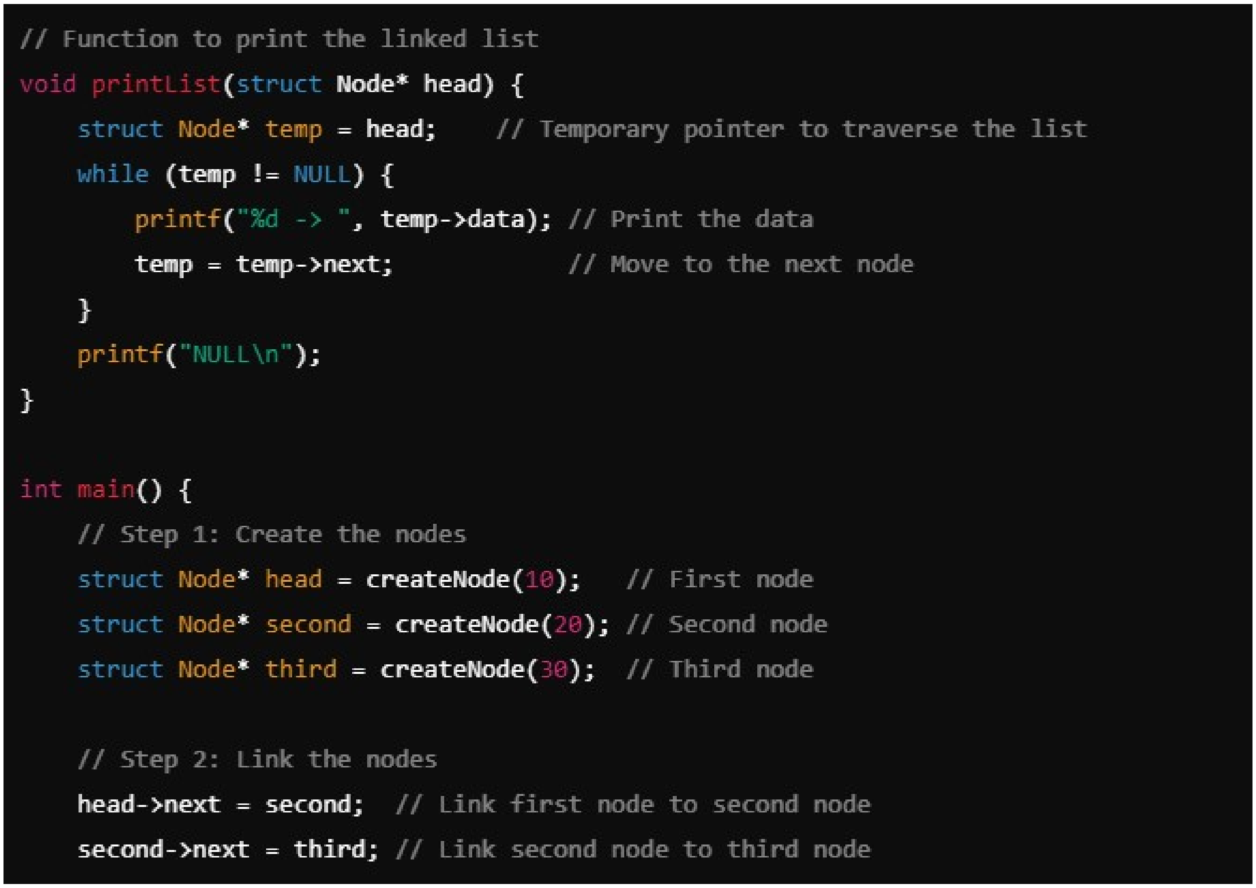
Step-by-Step Example

Let's create a linked list with 3 nodes, insert data, and print the linked list.

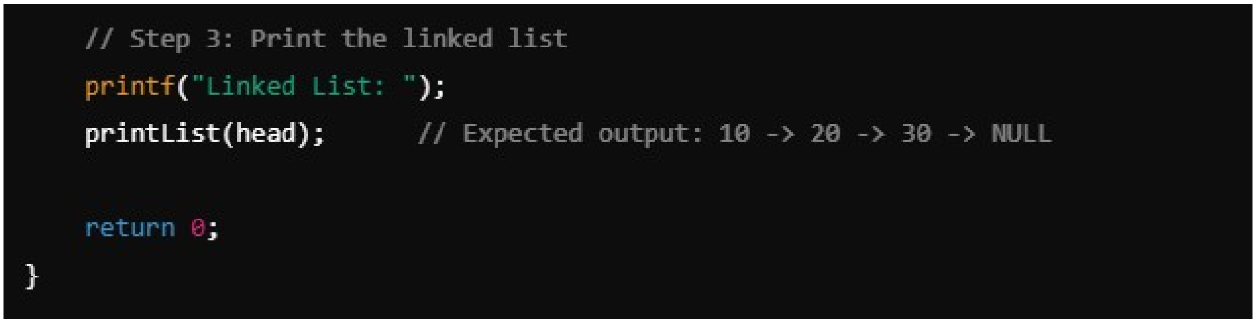
C Code Example



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

// Define the structure for a node
struct Node {
    int data;              // Data stored in the node
    struct Node* next;     // Pointer to the next node
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;        // Set the data
    newNode->next = NULL;        // Set the next pointer to NULL
    return newNode;
}
```



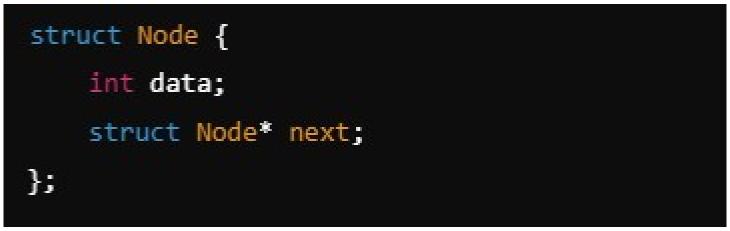
```c
// Function to print the linked list
void printList(struct Node* head) {
    struct Node* temp = head;    // Temporary pointer to traverse the list
    while (temp != NULL) {
        printf("%d -> ", temp->data); // Print the data
        temp = temp->next;            // Move to the next node
    }
    printf("NULL\n");
}

int main() {
    // Step 1: Create the nodes
    struct Node* head = createNode(10);   // First node
    struct Node* second = createNode(20); // Second node
    struct Node* third = createNode(30);  // Third node

    // Step 2: Link the nodes
    head->next = second;  // Link first node to second node
    second->next = third; // Link second node to third node
```



```c
    // Step 3: Print the linked list
    printf("Linked List: ");
    printList(head);      // Expected output: 10 -> 20 -> 30 -> NULL

    return 0;
}
```

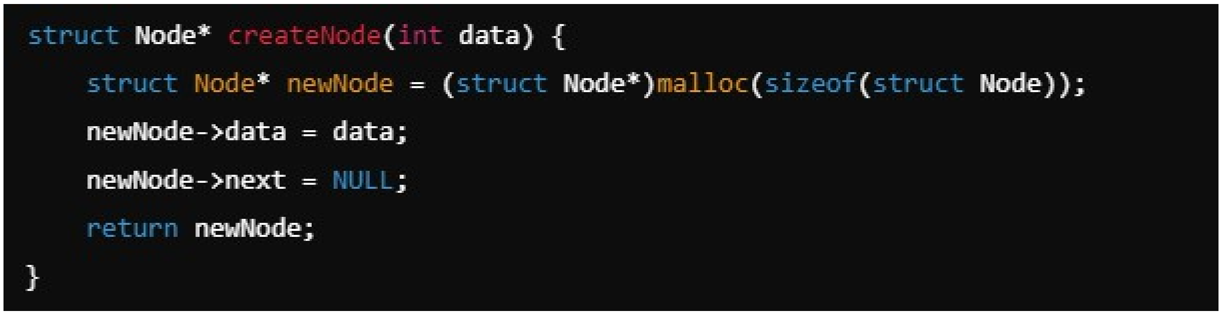
Explanation of the Code

1. **Structure Definition**: We define a `Node` structure with two members: `data` and `next`.



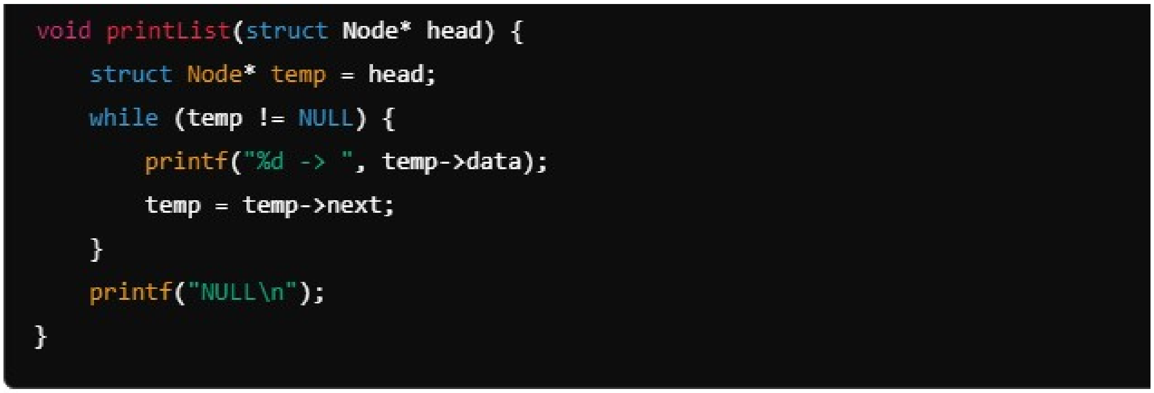
```c
struct Node {
    int data;
    struct Node* next;
};
```

2. **createNode Function**: This function dynamically allocates memory for a new node, assigns a value to the `data` field, and initializes the `next` pointer to `NULL`.



```c
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

3. **printList Function**: This function takes the head of the list as input and traverses the list, printing each node's data.

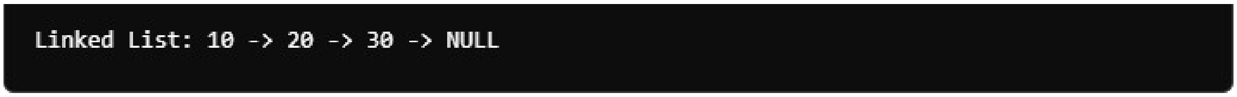


```c
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

4. **Main Program**:

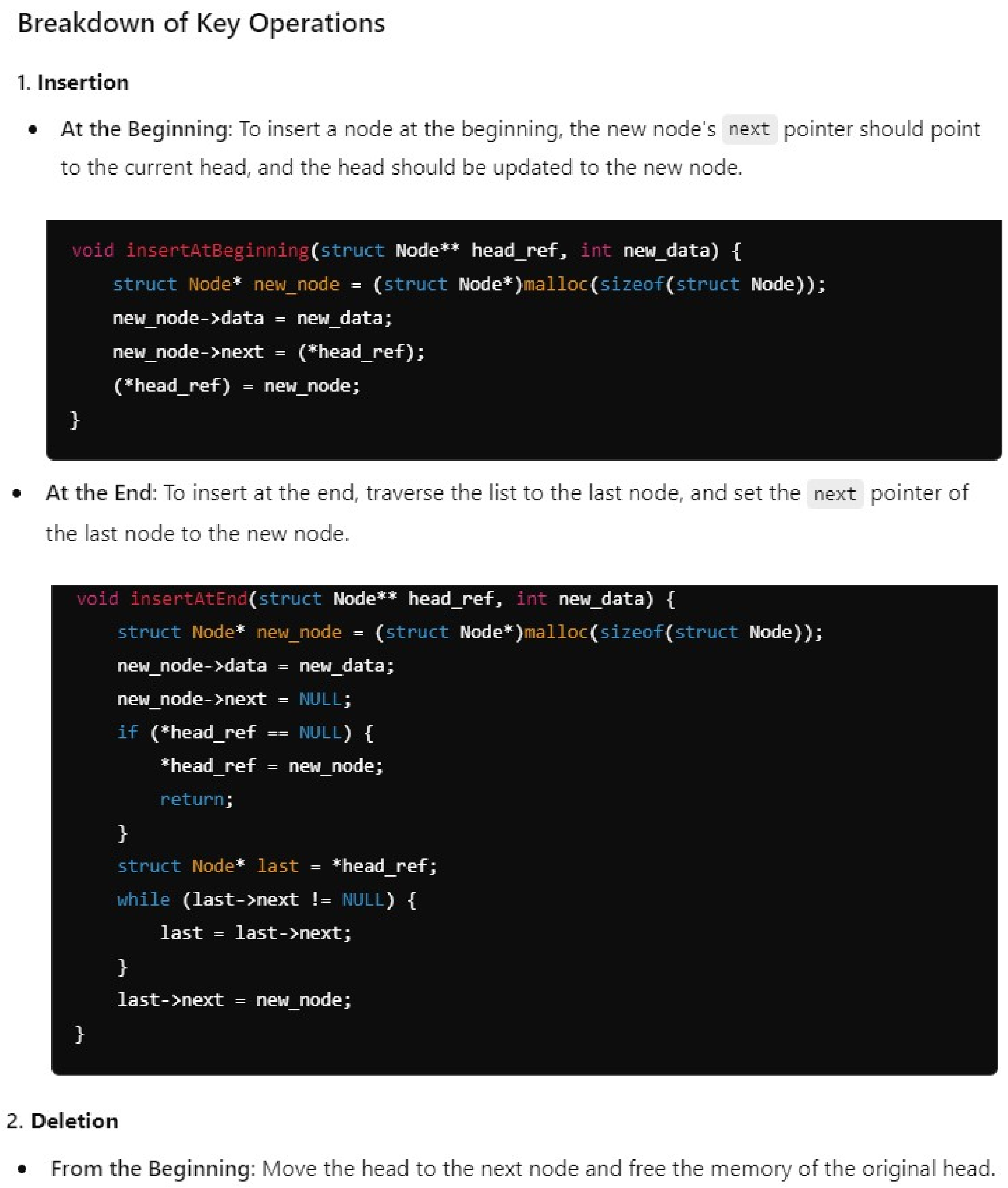
   - We create three nodes: `head`, `second`, and `third`, using the `createNode` function.
   - The `next` pointer of each node is linked to form a sequence.
   - Finally, the linked list is printed using the `printList` function.

Output of the Program



```
Linked List: 10 -> 20 -> 30 -> NULL
```

The list is successfully created with three nodes, and the data in each node is printed as we traverse through the list.



Breakdown of Key Operations

1. **Insertion**

- **At the Beginning**: To insert a node at the beginning, the new node's `next` pointer should point to the current head, and the head should be updated to the new node.

```
void insertAtBeginning(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
```

- **At the End**: To insert at the end, traverse the list to the last node, and set the `next` pointer of the last node to the new node.

```
void insertAtEnd(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = NULL;
    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }
    struct Node* last = *head_ref;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = new_node;
}
```

2. **Deletion**

- **From the Beginning**: Move the head to the next node and free the memory of the original head.

```
void deleteFromBeginning(struct Node** head_ref) {
    if (*head_ref == NULL) return;
    struct Node* temp = *head_ref;
    *head_ref = (*head_ref)->next;
    free(temp);
}
```

3. **Traversal**

To traverse through the linked list, we start from the head and keep moving to the next node until we reach `NULL`.