Doubly Linked List in c with proper explanation

A **doubly linked list** is a data structure where each node has three components:

1. A pointer to the previous node.
2. The data itself.
3. A pointer to the next node.

Unlike a singly linked list, where each node points only to the next node, a doubly linked list allows traversal in both directions — forward and backward.

Structure of a Doubly Linked List Node

In C, we define a doubly linked list node as:

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

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

Here, each node stores:

- data: the value of the node.
- next: a pointer to the next node in the list.
- prev: a pointer to the previous node in the list.

Basic Operations

1. Insertion at the Beginning

To insert a new node at the beginning of the list, we update the prev of the current head and next of the new node to point to the current head.

```
void insertAtBeginning(struct Node** head, int new_data) {
   // Allocate memory for new node
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

   // Put data in the new node
   new_node->data = new_data;

   // Make the new node's next point to the current head
   new_node->next = (*head);
   new_node->prev = NULL;

   // Update previous head node's prev to point to the new node
   if ((*head) != NULL)
      (*head)->prev = new_node;

   // Move the head pointer to the new node
   (*head) = new_node;
}
```

2. Insertion at the End

To insert a node at the end of the list, we traverse to the last node, then update its next pointer and the prev pointer of the new node.

```
void insertAtEnd(struct Node** head, int new_data) {
    // Allocate memory for new node
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

    struct Node* last = *head;  // Used for traversal

    // Put data in the new node
    new_node->data = new_data;
    new_node->next = NULL;  // As it will be the last node

    // If the list is empty, make the new node the head
    if (*head == NULL) {
        new_node->prev = NULL;
        *head = new_node;
        return;
    }

    // Traverse to the last node
    while (last->next != NULL)
        last = last->next;

    // Change the next of the last node to point to the new node
    last->next = new_node;

    // Make the new node's prev point to the last node
    new_node->prev = last;
}
```

3. Traversal

To traverse the doubly linked list in forward direction:

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

To traverse the list in backward direction, we first move to the last node, then traverse backward using the prev pointer:

```
void traverseBackward(struct Node* node) {
    struct Node* last = NULL;

    // Move to the last node
    while (node != NULL) {
        last = node;
        node = node->next;
    }

    // Traverse backward
    while (last != NULL) {
        printf("%d -> ", last->data);
        last = last->prev;
    }
    printf("NULL\n");
}
```

Full Example Program

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

// Doubly linked list node structure
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

// Function to insert at the beginning
void insertAtBeginning(struct Node** head, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head);
    new_node->prev = NULL;

    if ((*head) != NULL)
        (*head)->prev = new_node;

    (*head) = new_node;
}

// Function to insert at the end
void insertAtEnd(struct Node** head, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head;
    new_node->data = new_data;
    new_node->next = NULL;

    if (*head == NULL) {
        new_node->prev = NULL;
        *head = new_node;
        return;
    }

    while (last->next != NULL)
        last = last->next;

    last->next = new_node;
    new_node->prev = last;
}

// Function to traverse forward
void traverseForward(struct Node* node) {
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

// Function to traverse backward
void traverseBackward(struct Node* node) {
    struct Node* last = NULL;

    while (node != NULL) {
        last = node;
        node = node->next;
    }

    while (last != NULL) {
        printf("%d -> ", last->data);
        last = last->prev;
    }
    printf("NULL\n");
}

// Driver program to test above functions
int main() {
    struct Node* head = NULL;

    insertAtEnd(&head, 10);
    insertAtEnd(&head, 20);
    insertAtBeginning(&head, 5);
    insertAtEnd(&head, 30);

    printf("Forward traversal: ");
    traverseForward(head);

    printf("Backward traversal: ");
    traverseBackward(head);

    return 0;
}
```

Explanation of the Program:

- insertAtBeginning adds nodes to the start of the list.
- insertAtEnd adds nodes to the end of the list.
- traverseForward prints the list from the head to the last node.
- traverseBackward prints the list from the last node to the head.

This basic structure allows easy insertion and traversal in both directions.