## Ex 6: singly linked list

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

// Define the structure for a node in the linked list
struct Node {
   int data;
   struct Node* next;
};

// Function to insert a new node at the beginning of the linked list
struct Node* insertNode(struct Node* head, int newData)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = newData;
   newNode->next = head;
   return newNode;
}

// Function to display the linked list
void displayList(struct Node* head)
 {
   printf("Linked List: ");
   while (head != NULL)
  {
      printf("%d -> ", head->data);
      head = head->next;
   }
   printf("NULL\n");
}

// Function to free the memory allocated for the linked list
void freeList(struct Node* head)
{
   struct Node* current = head;
   struct Node* nextNode;

   while (current != NULL)
  {
      nextNode = current->next;
      free(current);
      current = nextNode;
   }
}
```

```
int main()
{
    // Initialize an empty linked list
    struct Node* head = NULL;

    // Insert nodes into the linked list
    head = insertNode(head, 10);
    head = insertNode(head, 20);
    head = insertNode(head, 30);

    // Display the linked list
    displayList(head);

    // Free the memory allocated for the linked list
    freeList(head);

    return 0;
}
```

**Code Explanation:**

- **Insert node:**

1. **Function Signature:**

```
struct Node* insertNode(struct Node* head, int newData)
```

- **struct Node***: The function returns a pointer to a **struct Node**, indicating that it returns the head of the modified linked list.
- **insertNode**: The name of the function.
- **(struct Node* head)**: The parameter **head** is a pointer to the current head of the linked list.
- **int newData**: The data value to be added to the new node.

2. **Memory Allocation:**

```
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

- **struct Node* newNode**: Declares a new pointer to a **struct Node**, representing the new node to be added.
- **(struct Node*)malloc(sizeof(struct Node))**: Allocates memory for a new node using the **malloc** function. The **sizeof(struct Node)** ensures that enough memory is allocated to store a node.

3. **Node Initialization:**

```
newNode->data = newData; newNode->next = head;
```

- **newNode->data = newData**: Assigns the provided **newData** value to the **data** field of the new node.
- **newNode->next = head**: Sets the **next** pointer of the new node to point to the current head of the linked list.
-

4. **Return Statement:**

```
return newNode;
```

- Returns the pointer to the new node, effectively making the new node the new head of the linked list.

## ➢ delete node/ free memory allocated to node

1. Function Signature:

```
void freeList(struct Node* head)
```

void: The function returns no value.

freeList: The name of the function.

(struct Node* head): The parameter head is a pointer to the head of the linked list, indicating the starting point for freeing memory.

2. Initialization of Pointers:

```
struct Node* current = head;
struct Node* nextNode;
```

struct Node* current = head;: Initializes a pointer current to the head of the linked list, allowing traversal through the list.

struct Node* nextNode;: Declares a pointer nextNode to store the next node in the list before freeing the current node.

3. Memory Deallocation Loop:

```
while (current != NULL)
{
    nextNode = current->next;
    free(current);
    current = nextNode;
}
```

- **nextNode = current->next;**: This line stores the reference to the next node in the list before the current node is freed. It essentially keeps track of where the next node is located.
- **free(current);**: This line frees the memory allocated for the current node. This is the step where the node is "deleted" from the list by releasing the memory associated with it.
- **current = nextNode;**: This line moves the **current** pointer to the next node in the list, effectively progressing through the linked list.

The loop continues until the **current** pointer becomes **NULL**, indicating that the end of the linked list has been reached. In each iteration, the function frees the memory of the current node and moves on to the next one, effectively deleting each node from the linked list.

## ➢ Display list

The displayList function takes the head of a linked list as input and iterates through the entire list, printing the data values of each node. The function uses a while loop to traverse the list, printing each node's data and updating the head pointer to the next node. The result is a clear display of the linked list in the format "data1 -> data2 -> ... -> NULL".

1. Function Signature:

```
void displayList(struct Node* head)
```

void: The function returns no value.
displayList: The name of the function.
(struct Node* head): The parameter head is a pointer to the head of the linked list, indicating the starting point for displaying the list.

2. Print Statement:

```
printf("Linked List: ");
```

Prints a label indicating that the output will be the linked list.

3. Traversal Loop:

```
while (head != NULL)
{
    printf("%d -> ", head->data);
    head = head->next;
}
```

- while (head != NULL): Initiates a loop that continues until the end of the linked list is reached.
- printf("%d -> ", head->data);: Prints the data value of the current node, followed by an arrow (->), indicating the link to the next node.
- head = head->next;: Moves the head pointer to the next node in the list, continuing the loop.

4. End of List Print Statement:

```
printf("NULL\n");
```

Prints "NULL" to indicate the end of the linked list.