PROGRAM NO - 6

GIVEN A LINKED LIST AND A NUMBER N, WRITE A FUNCTION THAT RETURNS THE VALUE AT THE NTH NODE FROM THE END OF THE LINKED LIST

Algorithm to Find Nth Node from the End of a Linked List:

1. **Start with Two Pointers**: Initialize two pointers, fast and slow, both pointing to the head of the linked list.

2. **Move the Fast Pointer Ahead:** Move the **fast** pointer N positions ahead in the linked list. This step creates a gap of N nodes between the **fast** and **slow** pointers.

3. **Move Both Pointers Together**: Move both the **fast** and **slow** pointers one node at a time in the same direction. Continue this process until the fast pointer reaches the end of the list.

4. **Identify the Nth Node**: When the **fast** pointer reaches the end of the list, the **slow** pointer will be at the Nth node from the end.

5. **Return the Value:** Retrieve and return the value stored in the **slow** pointer, which represents the Nth node from the end of the linked list.

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

These lines include some standard C libraries that we'll use in our program.

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

Here, we define a structure called Node that represents a single element (node) in our linked list. It has two parts: data, which stores some information, and next, which points to the next node in the list.

```
Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}
```

This function createNode is used to create a new node. It takes an integer value as input, allocates memory for the node, sets the data to the given value, and sets the next pointer to NULL. If memory allocation fails, it shows an error message and exits the program.

```
void insertNode(Node** head, int value) {
    Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

The insertNode function adds a new node to the end of the linked list. It takes a pointer to the head of the list and an integer value to insert. If the list is empty (head is NULL), it sets the head to the new node. Otherwise, it traverses the list until it finds the last node and connects the new node to it.

```
int getNthFromEnd(Node* head, int n) {
    if (head == NULL || n <= 0) {
        printf("Invalid input.\n");
        exit(EXIT_FAILURE);
    }

    Node* fast = head;
    Node* slow = head;

    int count = 0;
    while (count < n) {
        if (fast == NULL) {
            printf("The list does not have %d nodes.\n", n);
            exit(EXIT_FAILURE);
        }
        fast = fast->next;
        count++;
    }

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

    return slow->data;
}
```

Now, here's the core of the problem. The getNthFromEnd function takes the head of the list and an integer n. It returns the value of the node that is n positions from the end of the list.

First, it checks for invalid input (e.g., an empty list or a non-positive n).
It uses two pointers, fast and slow, initially pointing to the head.
The fast pointer is moved n positions ahead. This creates a gap of n nodes between fast and slow.
Then, both fast and slow are moved one step at a time until fast reaches the end of the list. When this happens, slow will be at the n-th node from the end.

```
void freeList(Node* head) {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}
```

The freeList function is used to free the memory allocated for the linked list. It iterates through the list, freeing each node one by one.

```
int main() {
    Node* head = NULL;
    int n, value;

    // Insert nodes into the linked list
    insertNode(&head, 1);
    insertNode(&head, 2);
    insertNode(&head, 3);
    insertNode(&head, 4);
    insertNode(&head, 5);

    // Get the value at the Nth node from the end
    n = 3;
    int result = getNthFromEnd(head, n);
    printf("Value at the %dth node from the end: %d\n", n, result);

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

    return 0;
}
```

Finally, in the main function, we create a linked list with some values, find the value at the Nth node from the end (in this case, n = 3), print the result, and then free the memory to avoid memory leaks.