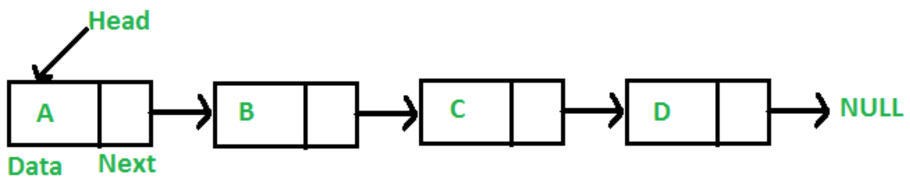
Find the n'th node from the end of a Linked List

Given a Linked List and a number n, write a function that returns the value at the n'th node from the end of the Linked List.

For example, if the input is below list and n = 3, then output is "B"



Method 1 (Use length of linked list)

1. Calculate the length of Linked List. Let the length be len.
2. Print the (len - n + 1)th node from the beginning of the Linked List.

Double pointer concept : First pointer is used to store the address of the variable and second pointer used to store the address of the first pointer. If we wish to change the value of a variable by a function, we pass pointer to it. And if we wish to change value of a pointer (i. e., it should start pointing to something else), we pass pointer to a pointer.

Below is the implementation of the above approach:

```
// Simple C++ program to find n'th node from end
#include <bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node {
    int data;
    struct Node* next;
```

```
};

/* Function to get the nth node from the last of a linked lis
void printNthFromLast(struct Node* head, int n)
{
    int len = 0, i;
    struct Node* temp = head;

    // count the number of nodes in Linked List
    while (temp != NULL) {
        temp = temp->next;
        len++;
    }

    // check if value of n is not
    // more than length of the linked list
    if (len < n)
        return;

    temp = head;

    // get the (len-n+1)th node from the beginning
    for (i = 1; i < len - n + 1; i++)
        temp = temp->next;

    cout << temp->data;

    return;
}

void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node = new Node();

    /* put in the data */
    new_node->data = new_data;
    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

// Driver Codeint main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    // create linked 35->15->4->20
    push(&head, 20);
    push(&head, 4);
    push(&head, 15);
    push(&head, 35);

    printNthFromLast(head, 4);
    return 0;
}
```

Output

```
35
```

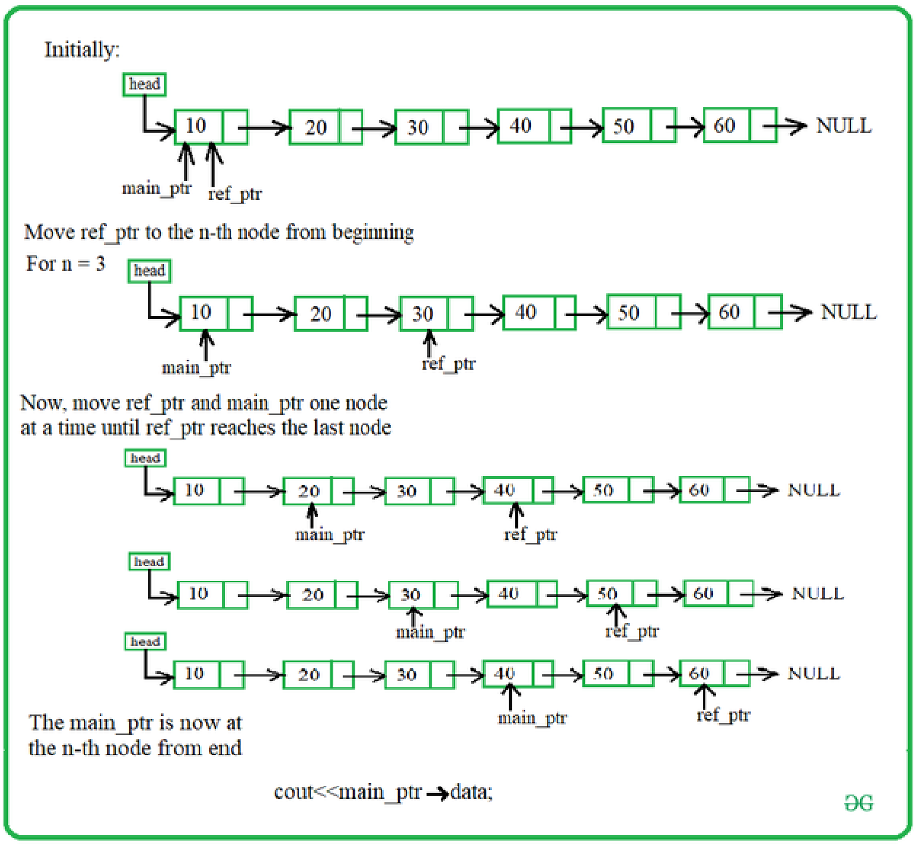
```
void printNthFromLast(struct Node* head, int n)
{
    static int i = 0;
    if (head == NULL)
        return;
    printNthFromLast(head->next, n);
    if (++i == n)
        cout<<head->data;
}
```

Time Complexity: O(n) where n is the length of linked list.

Method 2 (Use two pointers)

Maintain two pointers - reference pointer and main pointer. Initialize both reference and main pointers to head. First, move the reference pointer to n nodes from head. Now move both pointers one by one until the reference pointer reaches the end. Now the main pointer will point to nth node from the end. Return the main pointer.

Below image is a dry run of the above approach:


Initially:
head
10
20
30
40
50
60
NULL
main_ptr
ref_ptr
Move ref_ptr to the n-th node from beginning
For n = 3
Now, move ref_ptr and main_ptr one node at a time until ref_ptr reaches the last node
The main_ptr is now at the n-th node from end
cout<<main_ptr→data;

Below is the implementation of the above approach:

```
// C++ program to find n-th node
// from the end of the linked list.
#include <bits/stdc++.h>
using namespace std;

struct node {
    int data;
    node* next;
    node(int val)
    {
        data = val;
        next = NULL;
    }
};

struct llist {

    node* head;
    llist() { head = NULL; }

    // insert operation at the beginning of the list.
    void insertAtBegin(int val)
    {
        node* newNode = new node(val);
        newNode->next = head;
        head = newNode;
    }

    // finding n-th node from the end.
    void nthFromEnd(int n)
    {
        // create two pointers main_ptr and ref_ptr
        // initially pointing to head.
        node* main_ptr = head;
        node* ref_ptr = head;

        // if list is empty, return
        if (head == NULL) {
            cout << "List is empty" << endl;
            return;
        }

        // move ref_ptr to the n-th node from beginning.
        for (int i = 1; i < n; i++) {
            ref_ptr = ref_ptr->next;
            if (ref_ptr == NULL) {
                cout << n
                     << " is greater than no. of nodes in "
                        "the list"
                     << endl;
                return;
            }
        }

        // move ref_ptr and main_ptr by one node until
        // ref_ptr reaches end of the list.
        while (ref_ptr != NULL && ref_ptr->next != NULL) {
            ref_ptr = ref_ptr->next;
            main_ptr = main_ptr->next;
        }
        cout << "Node no. " << n
             << " from end is: " << main_ptr->data << endl;
    }

    void displaylist()
    {
        node* temp = head;
        while (temp != NULL) {
            cout << temp->data << "->";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
};

int main()
{
    llist ll;

    for (int i = 60; i >= 10; i -= 10)
        ll.insertAtBegin(i);

    ll.displaylist();
    for (int i = 1; i <= 7; i++)
        ll.nthFromEnd(i);

    return 0;
}
```

Output

```
10->20->30->40->50->60->NULL
Node no. 1 from end is: 60
Node no. 2 from end is: 50
Node no. 3 from end is: 40
Node no. 4 from end is: 30
Node no. 5 from end is: 20
Node no. 6 from end is: 10
7 is greater than no. of nodes in the list
```

Time Complexity: O(n) where n is the length of linked list.