Find the middle of a given linked list

Given a singly linked list, find the middle of the linked list. For example, if the given linked list is 1->2->3->4->5 then the output should be 3.

If there are even nodes, then there would be two middle nodes, we need to print the second middle element. For example, if given linked list is 1->2->3->4->5->6 then the output should be 4.

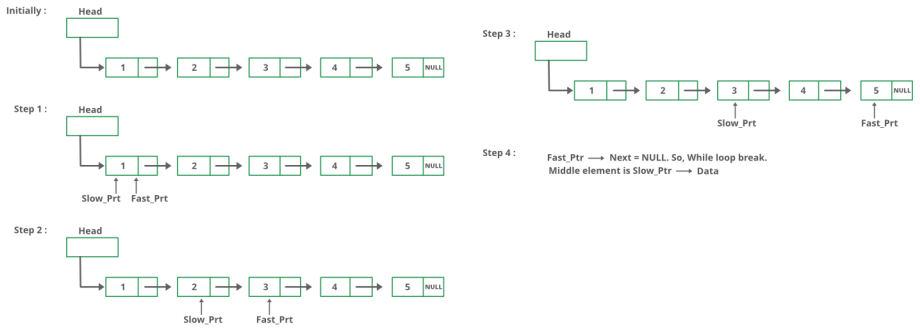
Method 1:

Traverse the whole linked list and count the no. of nodes. Now traverse the list again till count/2 and return the node at count/2.

Method 2:

Traverse linked list using two pointers. Move one pointer by one and the other pointers by two. When the fast pointer reaches the end slow pointer will reach the middle of the linked list.

Below image shows how printMiddle function works in the code :




```
// C++ program for the above approach
#include <iostream>
using namespace std;
```

```
class Node{
    public:
        int data;
        Node *next;
};

class NodeOperation{
  public:

    // Function to add a new node
    void pushNode(class Node** head_ref,int data_val){

        // Allocate node
        class Node *new_node = new Node();

         // Put in the data
         new_node->data = data_val;

        // 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;
    }

    // A utility function to print a given linked list

    void printNode(class Node *head){
        while(head != NULL){
            cout <<head->data << "->";
            head = head->next;
        }
        cout << "NULL" << endl;
    }

    void printMiddle(class Node *head){
        struct Node *slow_ptr = head;
```

```
        struct Node *fast_ptr = head;

        if (head!=NULL)
        {
            while (fast_ptr != NULL && fast_ptr->next != NULL
            {
                fast_ptr = fast_ptr->next->next;
                slow_ptr = slow_ptr->next;
            }
            cout << "The middle element is [" << slow_ptr->da
        }
    }
};

// Driver Code
int main(){
    class Node *head = NULL;
    class NodeOperation *temp = new NodeOperation();
    for(int i=5; i>0; i--){
        temp->pushNode(&head, i);
        temp->printNode(head);
        temp->printMiddle(head);
    }
    return 0;
}
```

Output

```
5->NULL
The middle element is [5]

4->5->NULL
The middle element is [5]

3->4->5->NULL
The middle element is [4]

2->3->4->5->NULL
The middle element is [4]

1->2->3->4->5->NULL
The middle element is [3]
```

Method 3:

Initialize mid element as head and initialize a counter as 0. Traverse the list from head, while traversing increment the counter and change mid to mid->next whenever the counter is odd. So the mid will move only half of the total length of the list.

```
#include <bits/stdc++.h>
using namespace std;

// Link list node

struct node{
    int data;
    struct node* next;
};

// Function to get the middle of
// the linked list
void printMiddle(struct node* head)
{
    int count = 0;
    struct node* mid = head;

    while (head != NULL)
    {

        // Update mid, when 'count'
        // is odd number
        if (count & 1)
            mid = mid->next;

        ++count;
        head = head->next;
    }

    // If empty list is provided
    if (mid != NULL)
        printf("The middle element is [%d]\n\n",
```

```
                mid->data);
}

void push(struct node** head_ref, int new_data)
{

    // Allocate node
    struct node* new_node = (struct node*)malloc(
        sizeof(struct 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;
}

// A utility function to print
// a given linked list
void printList(struct node* ptr)
{
    while (ptr != NULL)
    {
        printf("%d->", ptr->data);
        ptr = ptr->next;
    }
    printf("NULL\n");
}

// Driver code
int main()
{

    // Start with the empty list
```

```
    struct node* head = NULL;
    int i;

    for(i = 5; i > 0; i--)
    {
        push(&head, i);
        printList(head);
        printMiddle(head);
    }
    return 0;
}
```

Output

```
5->NULL
The middle element is [5]

4->5->NULL
The middle element is [5]

3->4->5->NULL
The middle element is [4]

2->3->4->5->NULL
The middle element is [4]

1->2->3->4->5->NULL
The middle element is [3]
```