Find the middle of a given linked list in C and Java

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

If there are even nodes, then there would be two middle nodes, we need to print second middle element. For example, if given linked list is 1->2->3->4->5->6 then 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 other pointer by two. When the fast pointer reaches end slow pointer will reach middle of the linked list.

C

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

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

/* Function to get the middle of the linked list*/
void printMiddle(struct 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;
        }
        printf("The middle element is [%d]\n\n", slow_ptr->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");
}

/* Drier program to test above function*/
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;
}
```

Java

```java
// Java program to find middle of linked list
class LinkedList
{
    Node head; // head of linked list

    /* Linked list node */
    class Node
    {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    /* Function to print middle of linked list */
    void printMiddle()
    {
        Node slow_ptr = head;
        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;
            }
            System.out.println("The middle element is [" +
                                slow_ptr.data + "] \n");
        }
    }

    /* Inserts a new Node at front of the list. */
    public void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    /* This function prints contents of linked list
       starting from  the given node */
    public void printList()
    {
        Node tnode = head;
        while (tnode != null)
        {
            System.out.print(tnode.data+"->");
            tnode = tnode.next;
        }
        System.out.println("NULL");
    }

    public static void main(String [] args)
    {
        LinkedList llist = new LinkedList();
        for (int i=5; i>0; --i)
        {
            llist.push(i);
            llist.printList();
            llist.printMiddle();
```

```
        }
    }
}
// This code is contributed by Rajat Mishra
```

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.

Thanks to Narendra Kangralkar for suggesting this method.

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

/* 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);
```

```c
    /* 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");
}

/* Drier program to test above function*/
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]
```