# Linked List Nth to Last Node - SOLUTION

## Problem Statement

Write a function that takes a head node and an integer value **n** and then returns the nth to last node in the linked list. For example, given:

In [4]:
```
class Node:

    def __init__(self, value):
        self.value = value
        self.nextnode  = None
```

In [5]:
```
a = Node(1)
b = Node(2)
c = Node(3)
d = Node(4)
e = Node(5)

a.nextnode = b
b.nextnode = c
c.nextnode = d
d.nextnode = e

# This would return the node d with a value of 4, because its the 2nd to last node.
target_node = nth_to_last_node(2, a)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-a1e2bc5209c8> in <module>
     11
     12 # This would return the node d with a value of 4, because its the 2nd to last node.
---> 13 target_node = nth_to_last_node(2, a)

NameError: name 'nth_to_last_node' is not defined
```

In [6]:
```
target_node.value
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-867ea52f7d5a> in <module>
----> 1 target_node.value

NameError: name 'target_node' is not defined
```

## Solution

One approach to this problem is this:

Imagine you have a bunch of nodes and a "block" which is n-nodes wide. We could walk this "block" all the way down the list, and once the front of the block reached the end, then the other end of the block would be a the Nth node!

So to implement this "block" we would just have two pointers a left and right pair of pointers. Let's mark out the steps we will need to take:

- Walk one pointer **n** nodes from the head, this will be the right_point
- Put the other pointer at the head, this will be the left_point
- Walk/traverse the block (both pointers) towards the tail, one node at a time, keeping a distance **n** between them.
- Once the right_point has hit the tail, we know that the left point is at the target.

Let's see the code for this!

In [9]:
```
def nth_to_last_node(n, head):

    left_pointer  = head
    right_pointer = head

    # Set right pointer at n nodes away from head
    for i in range(n-1):

        # Check for edge case of not having enough nodes!
        if not right_pointer.nextnode:
            raise LookupError('Error: n is larger than the linked list.')

        # Otherwise, we can set the block
        right_pointer = right_pointer.nextnode

    # Move the block down the linked list
    while right_pointer.nextnode:
        left_pointer  = left_pointer.nextnode
        right_pointer = right_pointer.nextnode

    # Now return left pointer, its at the nth to last element!
    return left_pointer
```

# Test Your Solution

In [10]:
```
"""
RUN THIS CELL TO TEST YOUR SOLUTION AGAINST A TEST CASE

PLEASE NOTE THIS IS JUST ONE CASE
"""

from nose.tools import assert_equal

a = Node(1)
b = Node(2)
c = Node(3)
d = Node(4)
e = Node(5)

a.nextnode = b
b.nextnode = c
c.nextnode = d
d.nextnode = e

####
```

```
class TestNLast(object):

    def test(self,sol):

        assert_equal(sol(2,a),d)
        print('ALL TEST CASES PASSED')

# Run tests
t = TestNLast()
t.test(nth_to_last_node)
```

```
ALL TEST CASES PASSED
```

# Good Job!