Header file

```
#ifndef LINKED_INCLUDED
#define LINKED_INCLUDED


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

typedef struct node Node;

Node* createNode (int);
Node* connect    (Node*, Node*);


// --------------------
//  Linked List related
//
// --------------------
typedef Node        LinkedList;
extern  LinkedList *l_head;

LinkedList* initialize (Node*);
Node*       insertLast (Node*);
Node*       traverseToTail (Node *);
Node*       kth_element (int);
Node*       kth_element_from_last (int);


// helper functions
void        print_list();

#endif
```

Source File

```
#include "linked.hpp"
#include <stdlib.h>

Node* createNode (int val) {
    Node* n = (Node *) malloc (sizeof (Node));
    n->data = val;
    n->next = 0;
    return n;
}

Node* connect (Node* n, Node* following) {
    if (n != 0) {
        n->next = following;
        return n;
    }
    return 0;
}

//-------------
//Linked List related
//-------------

LinkedList* l_head;   // global variable

LinkedList *initialize (Node* n) {
    l_head = n;
    return l_head;
}

Node* traverseToTail (Node *n) {
    Node *prev = 0;
    for (; n != 0; n = n-> next) {
        prev = n;
    }
    return prev;
}

Node* insertLast (Node *node_to_add) {
    Node* tail = traverseToTail (l_head);
    if (tail != 0)
        return connect (tail, node_to_add);
    return 0;
}

Node* kth_element (int count) {
```

```
    Node* k = 0;
    for (Node* n = l_head; n != 0 && count--; n = n->next)
        k = n;
    return k;
}

int size_of_list (LinkedList* ll ) {
    int count =0;
    for (Node* n = ll; n !=0; n = n->next, count++);
    return count;
}

Node* kth_element_from_last (int count) {
    int size = size_of_list(l_head);
    int k    = size - count;
    return kth_element (k);
}

#include <iostream>
using namespace std;

void print_list () {
    int count = 0;
    for (Node* n = l_head; n != 0; n = n->next, count++)
        cout << count << ". " << n << " :: " << n->data << endl;

}
```

Test file

```
#include "linked.hpp"
#include <gtest/gtest.h>

using namespace ::testing;

class LList : public ::testing::Test {
protected:
    virtual void SetUp(){
        initialize (createNode (1100) );
        second = createNode(1200); insertLast (second);
```

```
        third  = createNode (1300); insertLast (third);

        insertLast (createNode (1400) );
        insertLast (createNode (1500) );
        insertLast (createNode (1600) );
        insertLast (createNode (1700) );
    }
    Node *second, *third;
};


TEST_F(LList, create_a_node_using_dynamic_memory)
{

    Node* first = createNode (1001);  // with value 1001
    ASSERT_NE ((Node*)(0), first);
}

TEST_F(LList, create_a_node_with_data_in_it)
{
    Node* first = createNode (1001);  // with value 1001
    ASSERT_EQ (1001, first->data);
}

TEST_F(LList, connect_two_nodes_up)
{

    Node* first  =  createNode (1001);
    Node* second =  createNode (1002);
    connect (first, second);

    ASSERT_EQ (first->next, second);
    ASSERT_EQ (1002, first->next->data);
}

TEST_F(LList, initialize_linked_list)
{
    Node* first = createNode (1001);
    initialize (first);
    ASSERT_EQ (l_head, first);

}

TEST_F(LList, create_list_with_three_nodes)
{
```

```
    initialize (createNode(1100) );

    Node *second = createNode (1200);
    insertLast (second);

    Node *third  = createNode (1300);
    insertLast (third);

    ASSERT_EQ (l_head->next->next, third);

}

TEST_F (LList, find_3rd_element_in_list)
{

    print_list();

    ASSERT_EQ (third, kth_element (3) );
    ASSERT_EQ (1300,  kth_element (3)->data );

    ASSERT_EQ (1700, kth_element (7)->data );

}


TEST_F (LList, find_5th_element_from_last)
{

    print_list(); // containing 1100, 1200, 1300, 1400, 1500,
1600, 1700

    ASSERT_EQ (second,  kth_element_from_last (5) );
    ASSERT_EQ (1200,    kth_element_from_last (5)->data );

    insertLast (createNode (1800) );
    ASSERT_EQ  (third,  kth_element_from_last (5) );

}
```