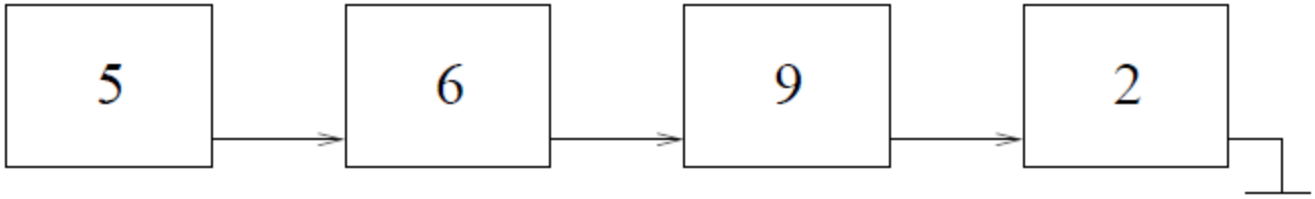
# Linked List

Dr. Na Li
CSE @ UTA
March 28, 2013

# Linked List

- Linked lists are a type of data structure, which is a way to represent data in memory. Memory is requested from the operating system as needed, with each additional piece of allocated memory added to our list.

- For example, if we had the numbers 5, 6, 9, 2, our linked list might look like this:



# Creating Linked List

- In order for us to navigate our list, we must keep track of the address of each piece of allocated memory. We do this by creating a structure that contains a variable for storing our data as well as a pointer variable for storing the address of the next node in the list.
- In the simplest case, a node in the linked list will have this form:
  - struct node
  - {
    - int value; /* or any other type */
    - struct node* next;
  - };

# Arrays vs. Linked Lists

- Linked lists are an alternative to using arrays to store multiple, related values. Why not just use arrays?
- Arrays
  - pros
    - contiguous memory allows easy navigation and the use of row offsets
    - easy on programmer
  - cons
    - can't easily insert/delete items (time complexity?)
    - can't free after use if statically allocated arrays?

# Arrays vs. Linked Lists

- Linked List
  - pros
    - don't need to know the number of elements in advance
    - can insert new elements without moving other elements
    - add/delete new elements
    - can release unneeded memory
  - cons
    - not as easy to understand and manage
    - not easy to access a specific node

# Creating Linked List (1)

- To build a list with num nodes, we might do the following:

```
#include<stdio.h>
struct node{
int data;
struct node * next;
};
int main()
{
        int num, i;
        struct node * first = NULL; /* null is used to know where the list ends */
        struct node * temp;
    printf("Please input a positive integer:");
        scanf("%d", &num);
        for(i = 0; i< num; i++)
        {
                temp = (struct node *) malloc(sizeof(struct node));
                temp->data = i;
                temp->next = first;
                first = temp;
        }

        while(first != NULL)
        {
                printf("%d\n", first->data);
                first = first->next;
        }
        return 0;
}
```

- first points to the first node in the list.

# Creating Linked List (2)

```
#include<stdio.h>
struct node{
int data;
struct node * next;
};
int main()
{
        int num, i;
        struct node * first = NULL; /* null is used to know where the list ends */
        struct node * temp, *p, *pre;
   printf("Please input a positive integer:");
        scanf("%d", &num);
        for(i = 0; i< num; i++)
        {
                temp = (struct node *) malloc(sizeof(struct node));
                temp->data = i;
                temp->next = NULL;
                pre = p = first;
                while(p != NULL)
                {
                        pre = p;
                        p=p->next;
                }
                if(pre != NULL)
                        pre->next = temp;
                else
                        first = temp;

        }

        while(first != NULL)
        {
                printf("%d\n", first->data);
                first = first->next;
        }
        return 0;
}
```

- What's the difference between the two implementations of creating a linked list?
- What we can do to make (2) more efficient is to have a pointer pointing to the tail of the list.
  - temp = (struct node *) malloc(sizeof(struct node));
  - temp->data = i;
  - temp->next = first;
  - tail->next = temp;//Note whenever you use tail->, tail cannot be NULL
  - tail = temp;

# Printing the list

```
void printlist(struct node * head)
{
    printf("The list includes ");
    while(head != NULL)
    {
        printf("%d   ", head->data);
        head = head->next;
    }
    printf("\n");
}
```

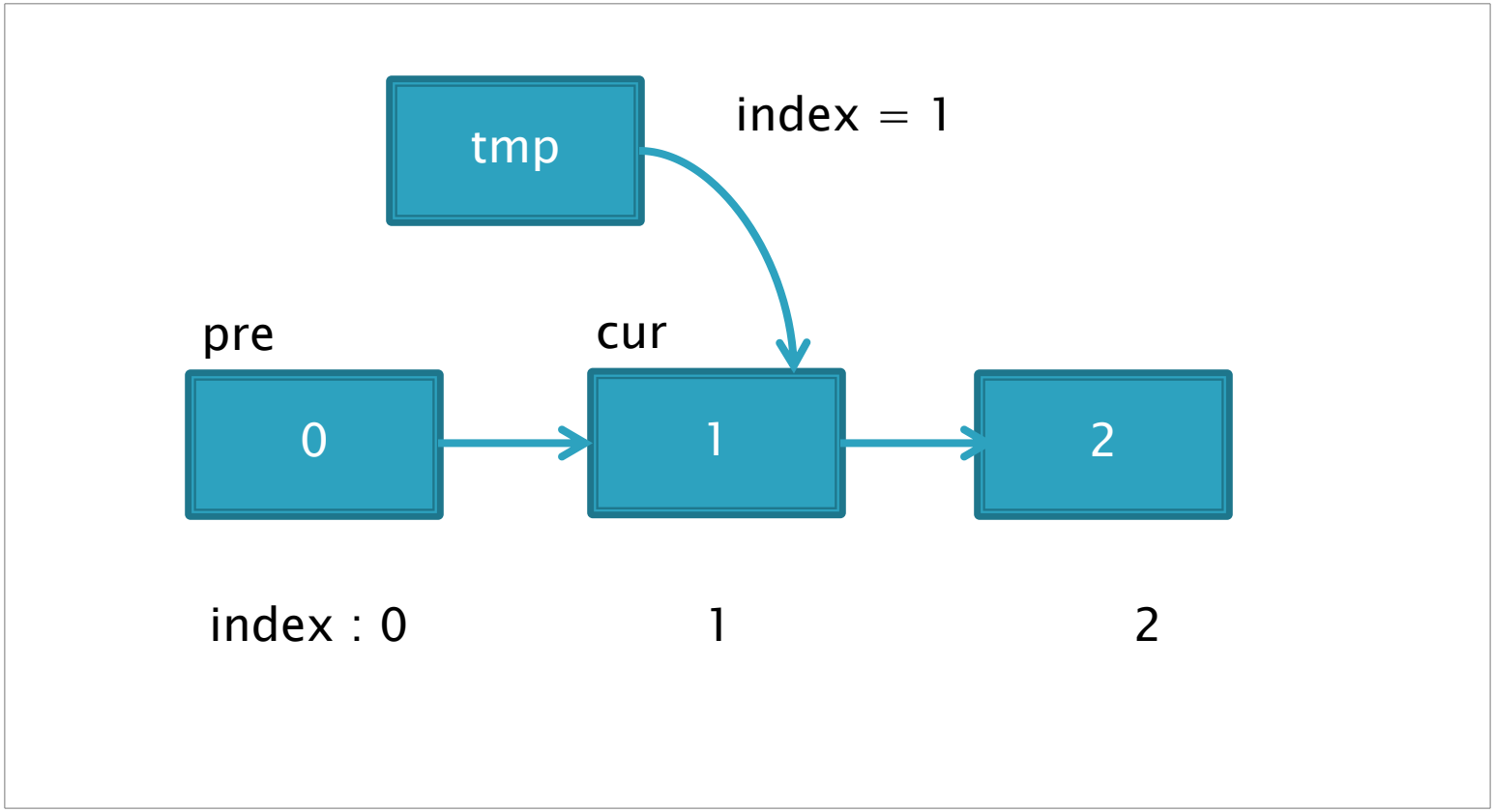
# Searching Linked List

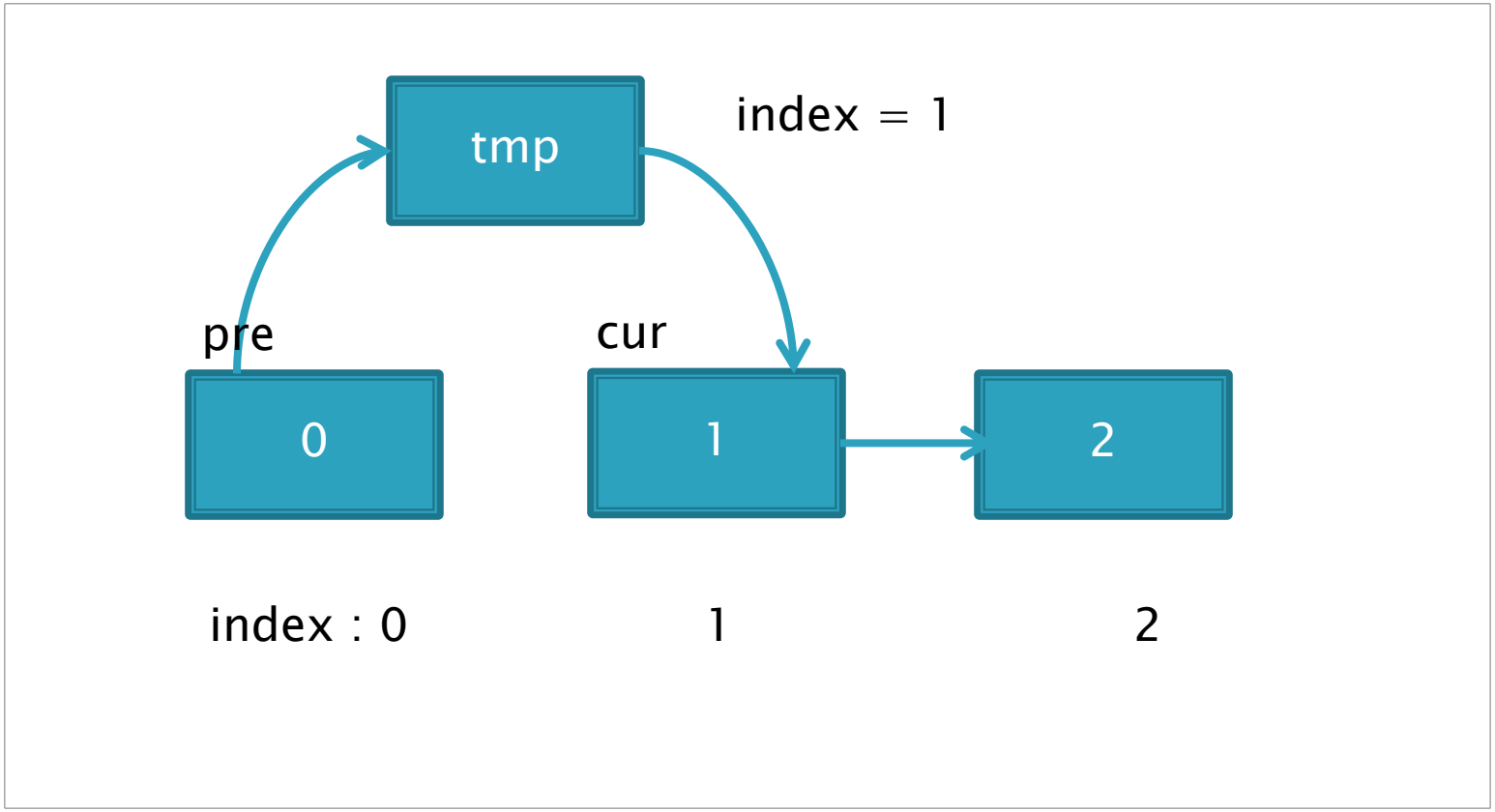
- In this example, we search the list for each node containing a value of d.

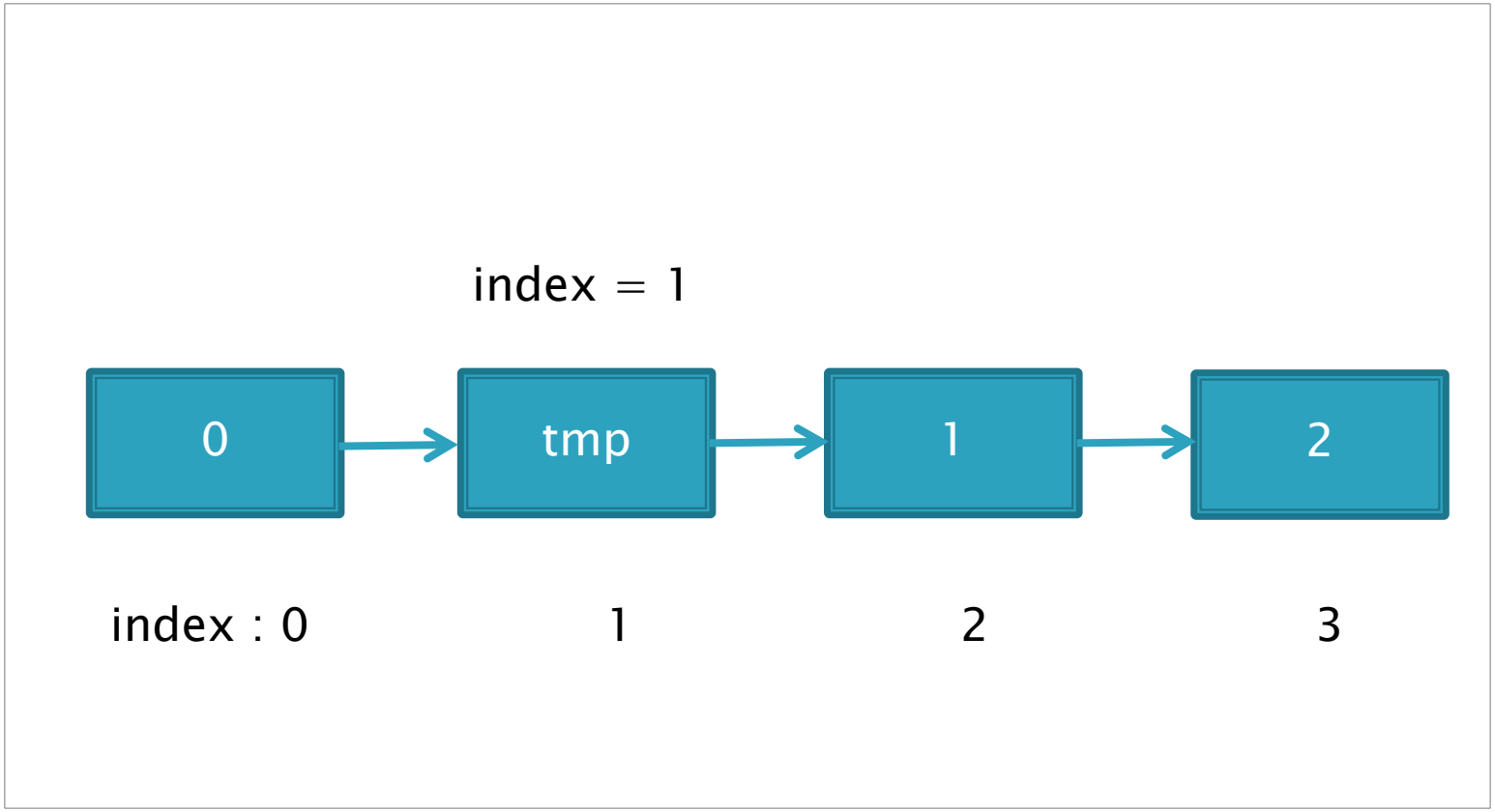
```
void searchNode(struct node * head, int d)
{
        struct node *p;
        int i = 0;
        p = head;
        while(p!= NULL)
        {
                if(p->data == d)
                        printf("found %d at %d\n", d, i);
                i++;
                p = p->next;
        }
}
```

# Inserting a Node in a Singly Linked List

```
struct node * insertNode(struct node *head, int t, int index)
{
        struct node *tmp, *pre, *cur;
        int i = 0;
        tmp = (struct node *) malloc(sizeof(struct node));
        tmp->data = t;
        if(index == 0)
        {
                tmp->next = head;
                head = tmp;
        }
        else
        {
                cur = head;
                while(i < index)
                {
                        pre = cur;
                        cur = cur->next;
                        i++;
                }//find the place to insert
                tmp->next = pre->next;//point to the next node
                pre->next = tmp;//disconnect and reconnect
        }
        return head;
}
```


tmp
index = 1
pre
cur
0
1
2
index : 0
1
2


tmp
index = 1
pre
cur
0
1
2
index : 0
1
2

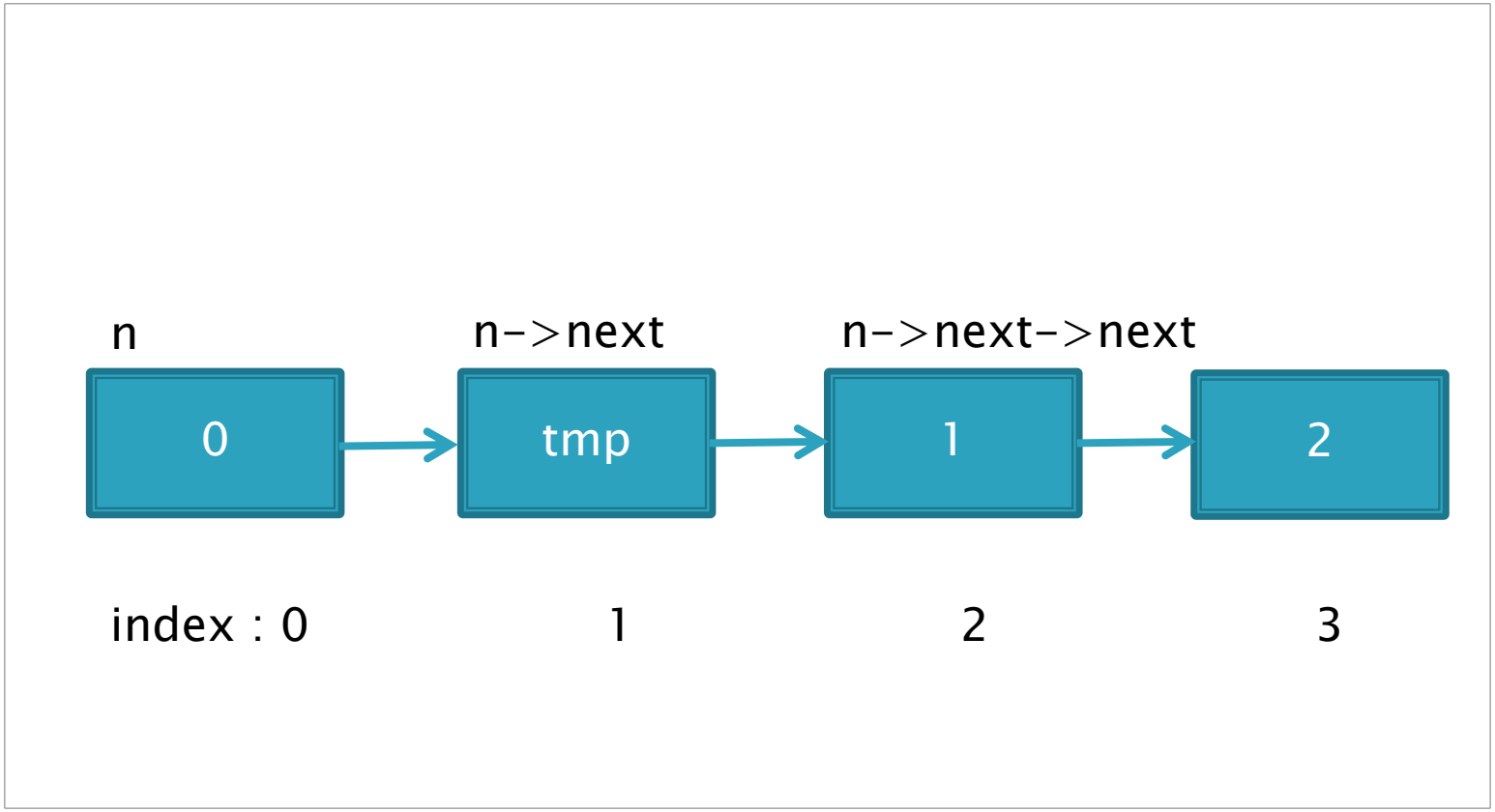

index = 1
0
tmp
1
2
index : 0
1
2
3

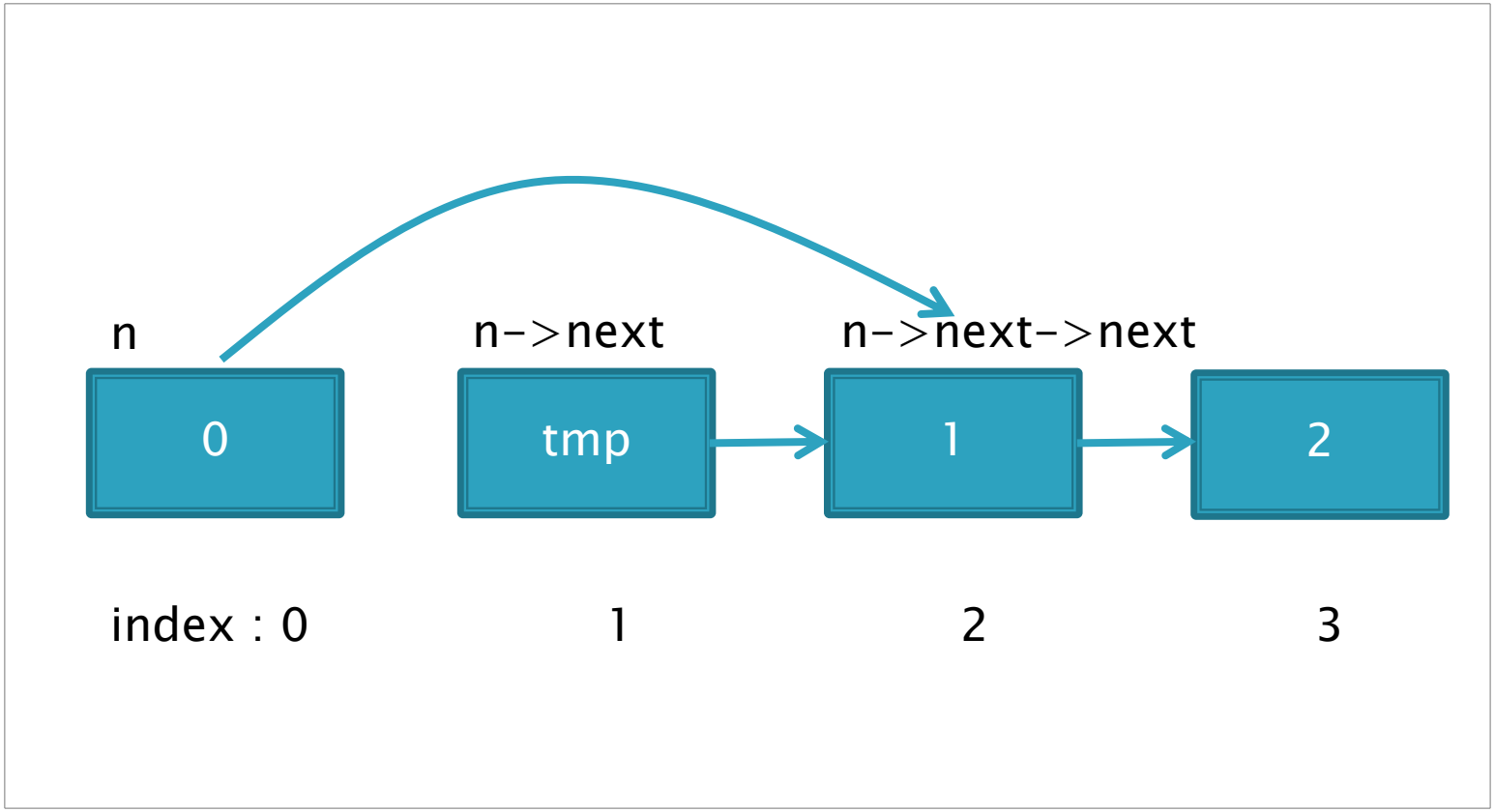
- For inserting a node in the linked list, what matters is to find the node which should point to the node to be inserted.
- There are variations of inserting a node into a linked list.
  - Insert it at a particular place based on the index
  - Insert it before/after a node which has value equal to the input value

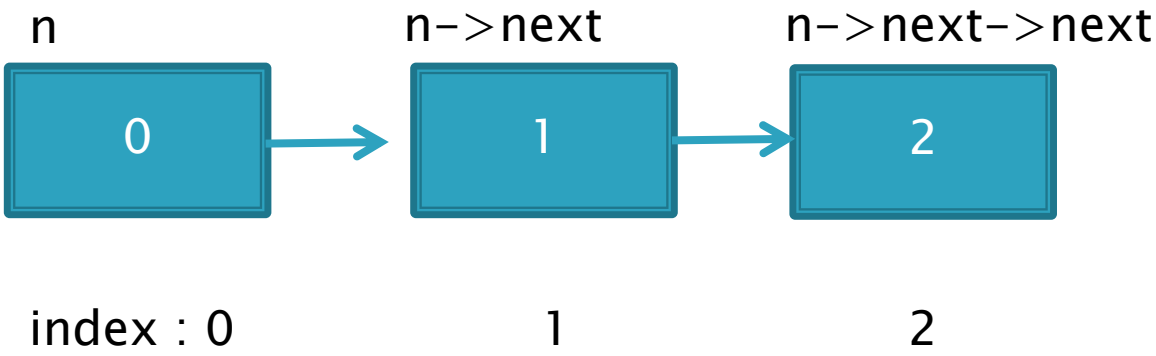
# Deleting a Node from a Singly Linked List

```
struct node * deleteNode(struct node * head, int d) {
        struct node * n = head, *tmp;
        if (n->data == d) {
                return head->next; /* moved head */
        }
        while (n->next != NULL) {
                if (n->next->data == d) {
                        tmp = n->next;
                        n->next = n->next->next;
                        free(tmp);/*if the node is created by malloc()*/
                        break;
        }
        n = n->next;
        }
        return head;/* head didn't change */
}
```

The returned value is the head pointer of the list.


n
n->next
n->next->next
0
tmp
1
2
index : 0
1
2
3


n
n->next
n->next->next
0
tmp
1
2
index : 0
1
2
3


n
n->next
n->next->next
0
1
2
index : 0
1
2

# Other Considerations

- In our examples, the purpose of our linked list was to store a single int in each node. We could store other types of data or even multiple data items per node.

- One potential problem with the simple linked list in our examples is that we can only go in one direction. It is possible to create a doubly linked lists that contain two pointers in each structure, one for each direction.

# Find the nth to the end

- Assumption: The minimum number of nodes in list is *n*. Algorithm:
  - Create two pointers, p1 and p2, that point to the beginning of the node.
  - Increment p2 by n-1 positions, to make it point to the nth node from the beginning (to make the distance of n between p1 and p2(including p1 and p2)).
  - Check for p2->next == null if yes return value of *p1*, otherwise increment *p1* and *p2*.If next of *p2* is null it means *p1* points to the nth node from the last as the distance between the two is *n*.
  - Repeat Step 3.

```
LinkedListNode nthToEnd(LinkedListNode *head, int n) {
        if (head == null || n < 1) {
                return null;
        }
        LinkedListNode * p1, * p2;
        p1 = p2 = head;
        for (int j = 0; j < n - 1; j++) { // skip n-1 steps ahead
                if (p2 == null) {
                        return null; // not found since list size < n
                }
                p2 = p2->next;
        }
        while (p2->next != null) {
                p1 = p1->next;
                p2 = p2->next;
        }
        return p1;
}
```

# Tutorial for Doubly Linked List

- Can you create a doubly linked-list? Insert a node in or delete a node from the list? Search a node?
- http://www.thelearningpoint.net/computer-science/data-structures-doubly-linked-list-with-c-program-source-code

# Some interview questions

- Here are some questions from interviews for you to practice
- **1.** Write code to remove duplicates from an unsorted linked list.
- FOLLOW UP
- How would you solve this problem if a temporary buffer is not allowed?
- ---
- **2.** You have two numbers represented by a linked list, where each node contains a single digit.The digits are stored in reverse order, such that the 1's digit is at the head of the list.Write a function that adds the two numbers and returns the sum as a linked list.
- EXAMPLE Input: (3 -> 1 -> 5) + (5 -> 9 -> 2)
- Output: 8 -> 0 -> 8
- ---
- **3.** Given a circular linked list, implement an algorithm which returns node at the beginning of the loop.
- DEFINITION Circular linked list: A (corrupt) linked list in which a node's next pointer points to an earlier node, so as to make a loop in the linked list.
- EXAMPLE
- input: A -> B -> C -> D -> E -> C [the same C as earlier]
- output: C
- ---

# Study reference

- Stack and Queue
- http://www.thelearningpoint.net/computer-science/data-structures-stacks--with-c-program-source-code
- http://www.thelearningpoint.net/computer-science/data-structures-queues--with-c-program-source-code

- We will learn about stack and queue this Wednesday.