Data organization…

The List Data Struture

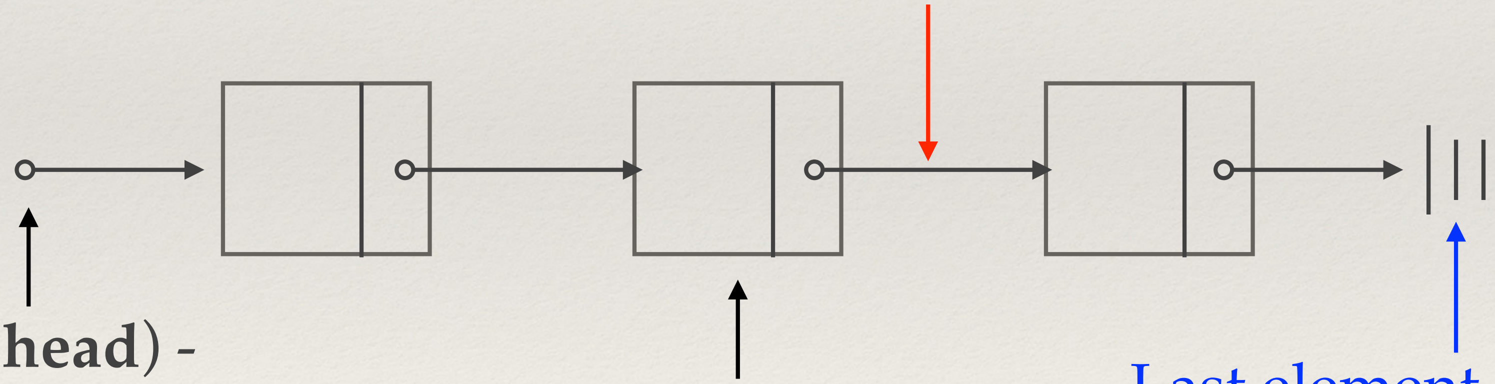
The List Data Structure

- A data structure is used to **organize** (structure) data so that it is easy to manipulate (**search, order, retrieve**).
- A **list** or **linked list** is a common structure with many applications.
    - It is also one of the easiest dynamic structures to build and use.

Linked List

- **Pointers** are used to *attach* **elements** (aka **nodes**) in the **list** into a **chain**.
- **Elements** in the list are usually **structures**.

Each element of the list points to the next one



Root (head) - start of the list is a pointer

A structure/element of the list

Last element of the list points to NULL

Building a Linked List

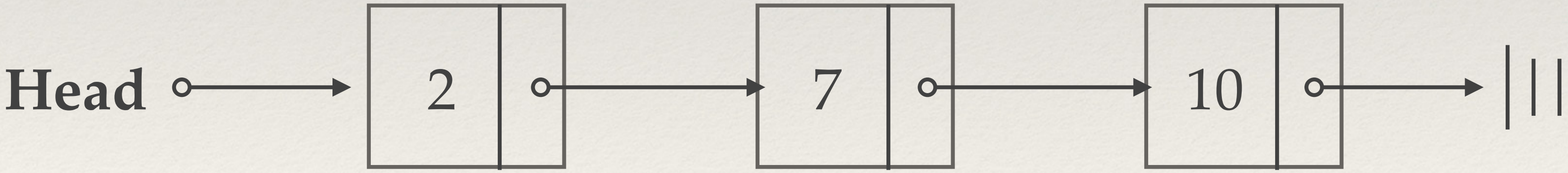
- To build an element of the list, we must declare a pointer inside each list element.

```
struct element {
   int num;
   struct element *next;
};
typedef struct element Node;
```

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Inserting into a List

- Adding an element into a list involves changing the pointers to possibly many elements.
- Example:



Head

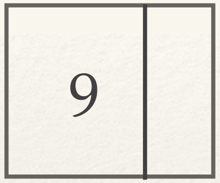
2

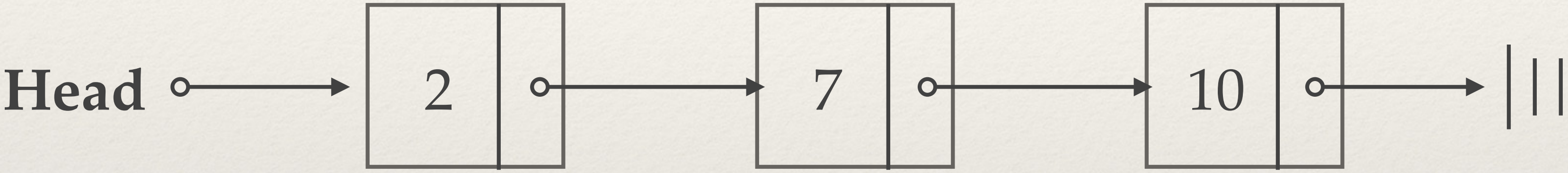
7

10

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Inserting into a List

- Add  to the list in-between 7 and 10.



- Create/`malloc()` a structure/element with `num` = 9 in it and store it using a temporary pointer.

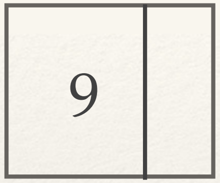
```
struct element *tmpptr;
```

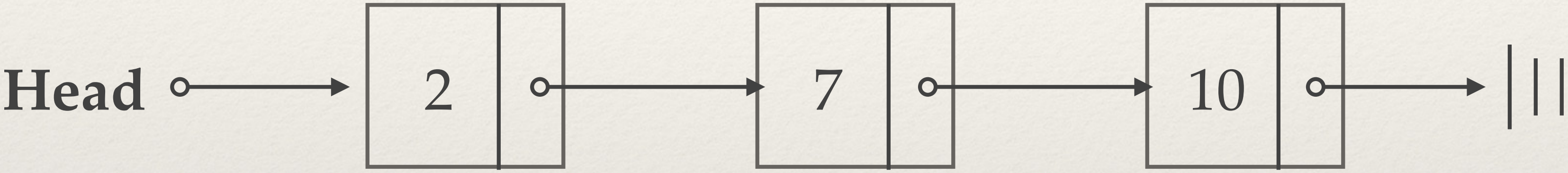
same as

```
Node *tmpptr;
```

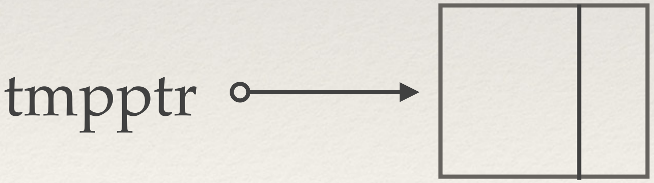
```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Inserting into a List

- Add  to the list in-between 2 and 7.



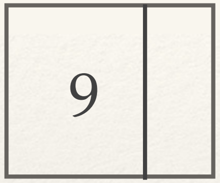
- Create/`malloc()` a structure/element with `num` = 9 in it and store it using a temporary pointer.

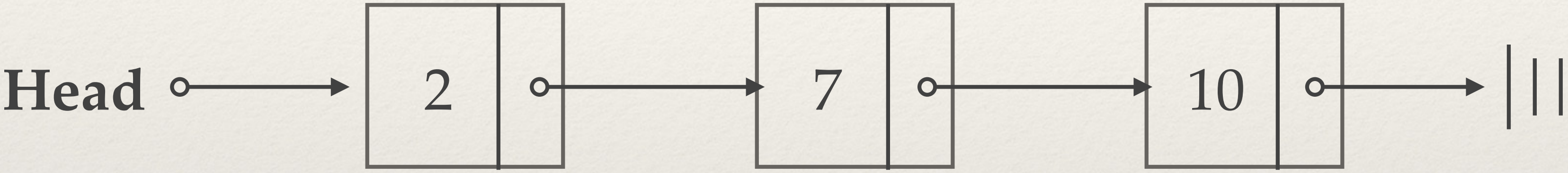


```
Node *tmpptr = malloc(sizeof(Node));
```

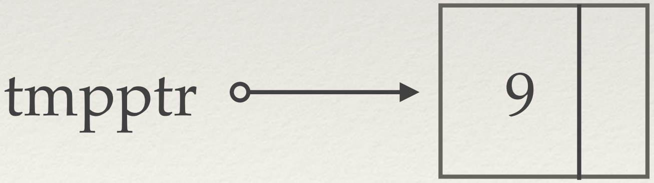
```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Inserting into a List

- Add  to the list in-between 2 and 7.



- Create/`malloc()` a structure/element with `num` = 9 in it and store it using a temporary pointer.

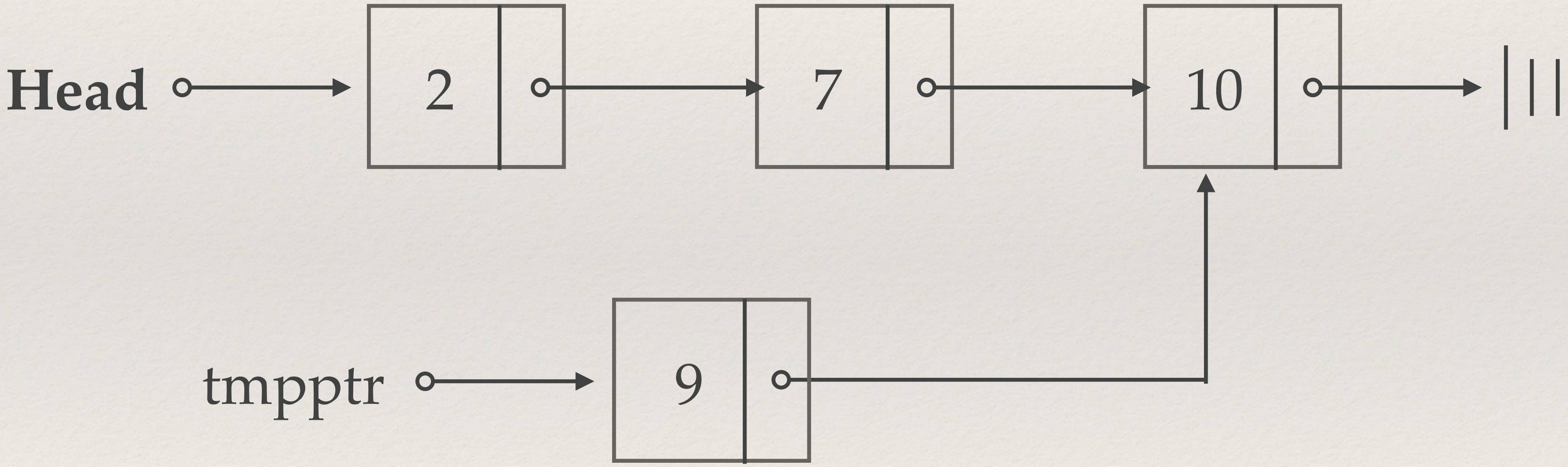


```
tmpptr->num = 9;
```

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Inserting into a List

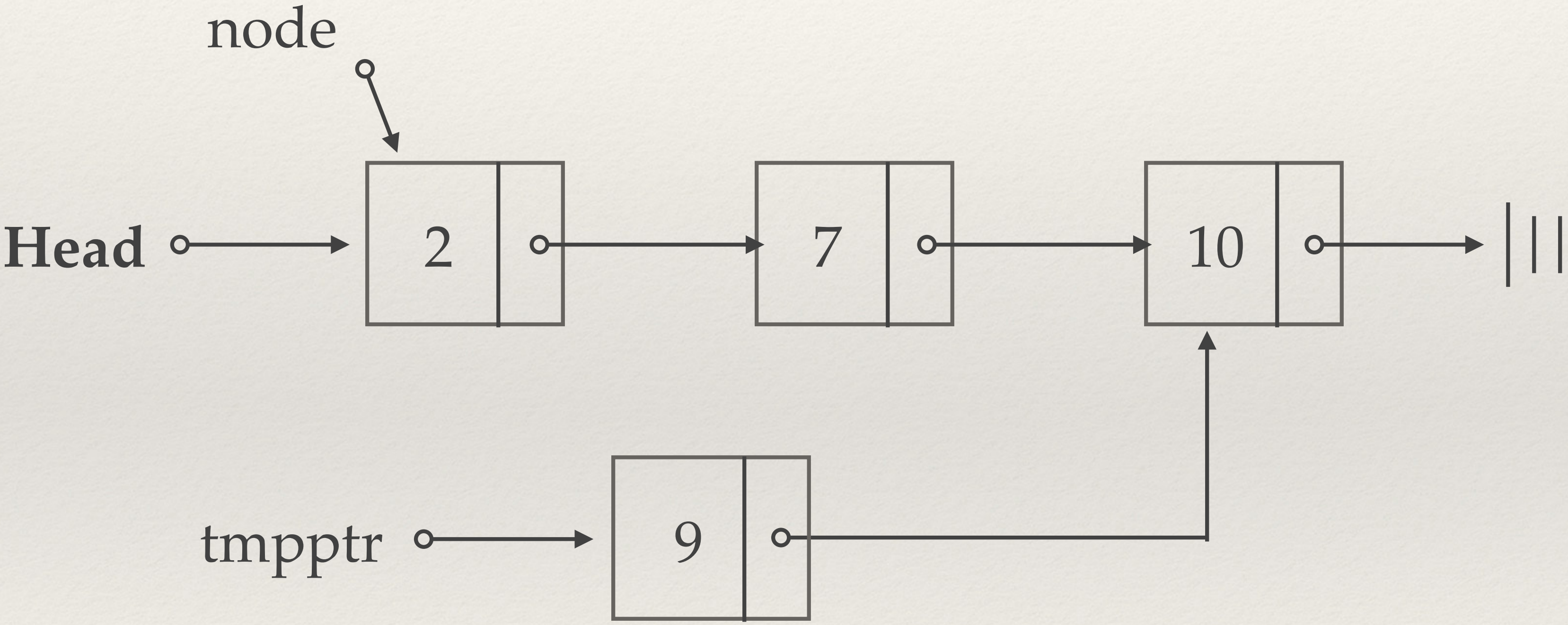
- Set the pointer from [7 |] to [9 |] instead of [10 |].



```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Inserting into a List

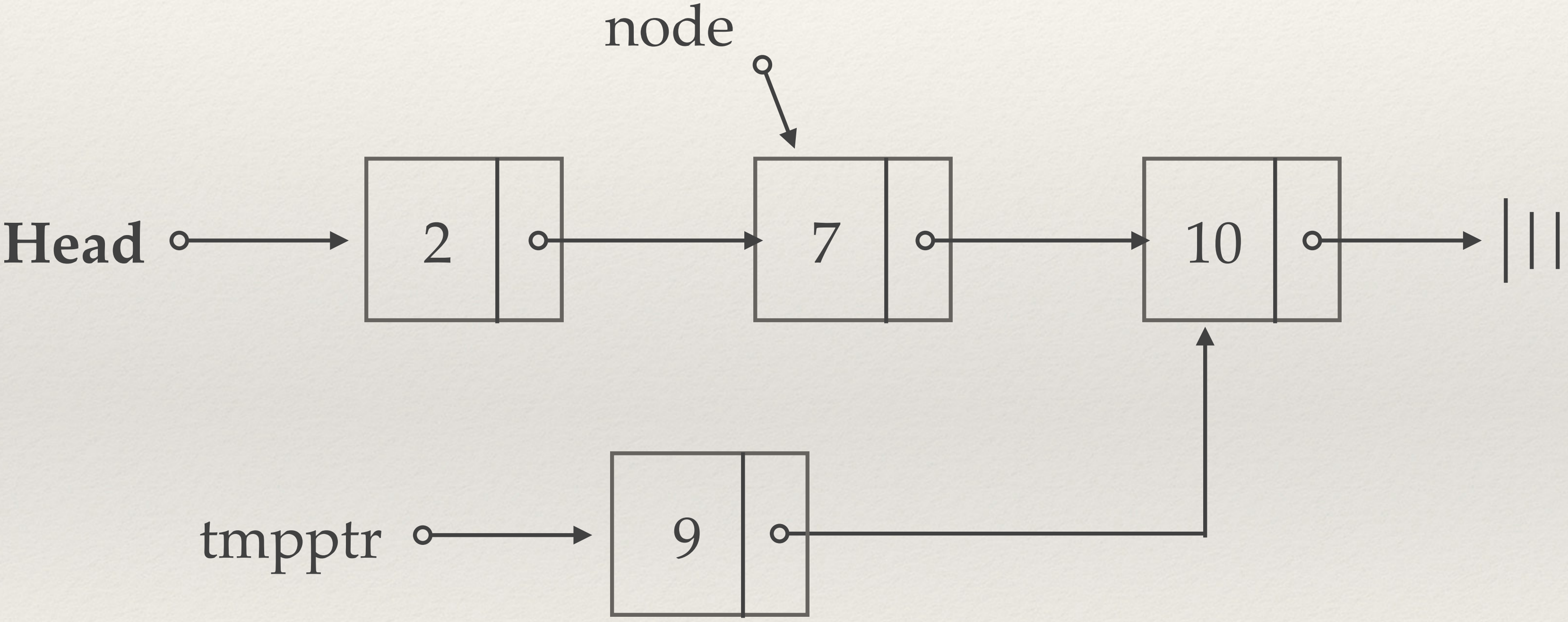
- Set the pointer from [7 |] to [9 |] instead of [10 |].



```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Inserting into a List

- Set the pointer from [7 |] to [9 |] instead of [10 |].

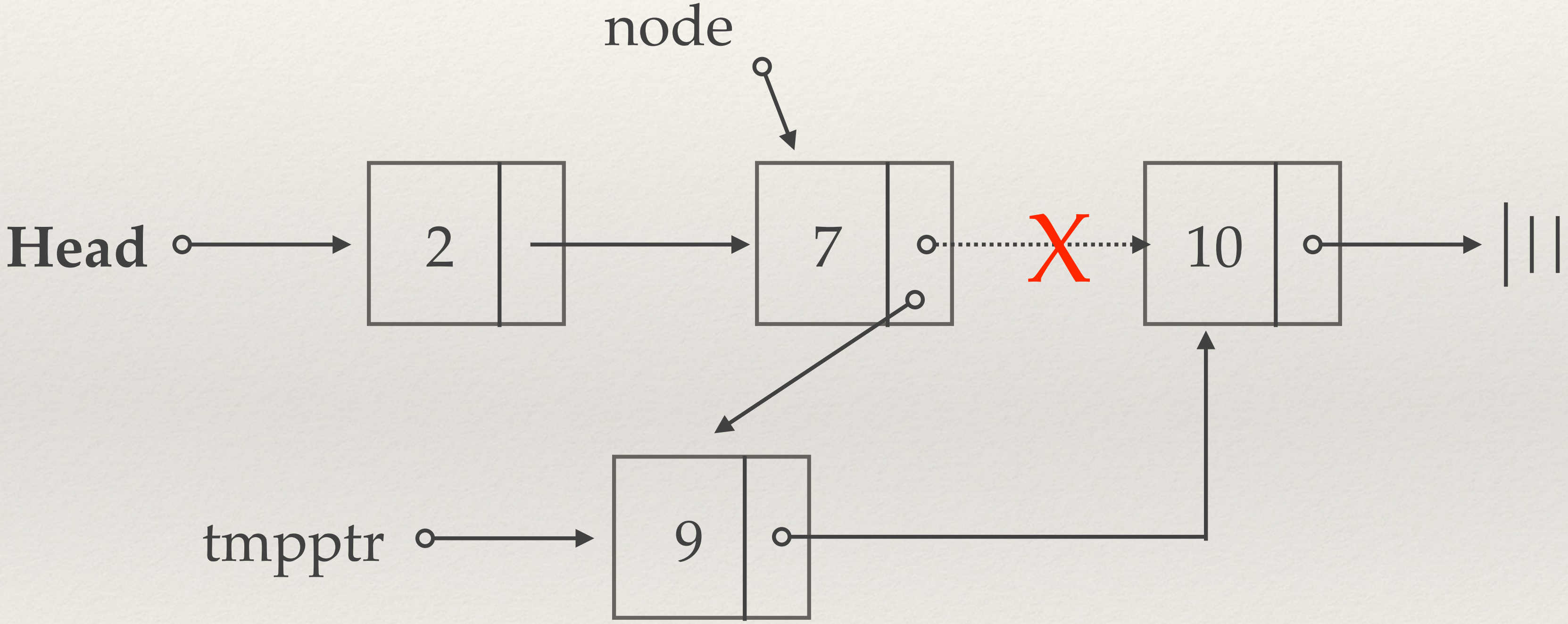


```
tmpptr->next = node->next;
```

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Inserting into a List

- Set the pointer from [2|] to [5|] instead of [7|].

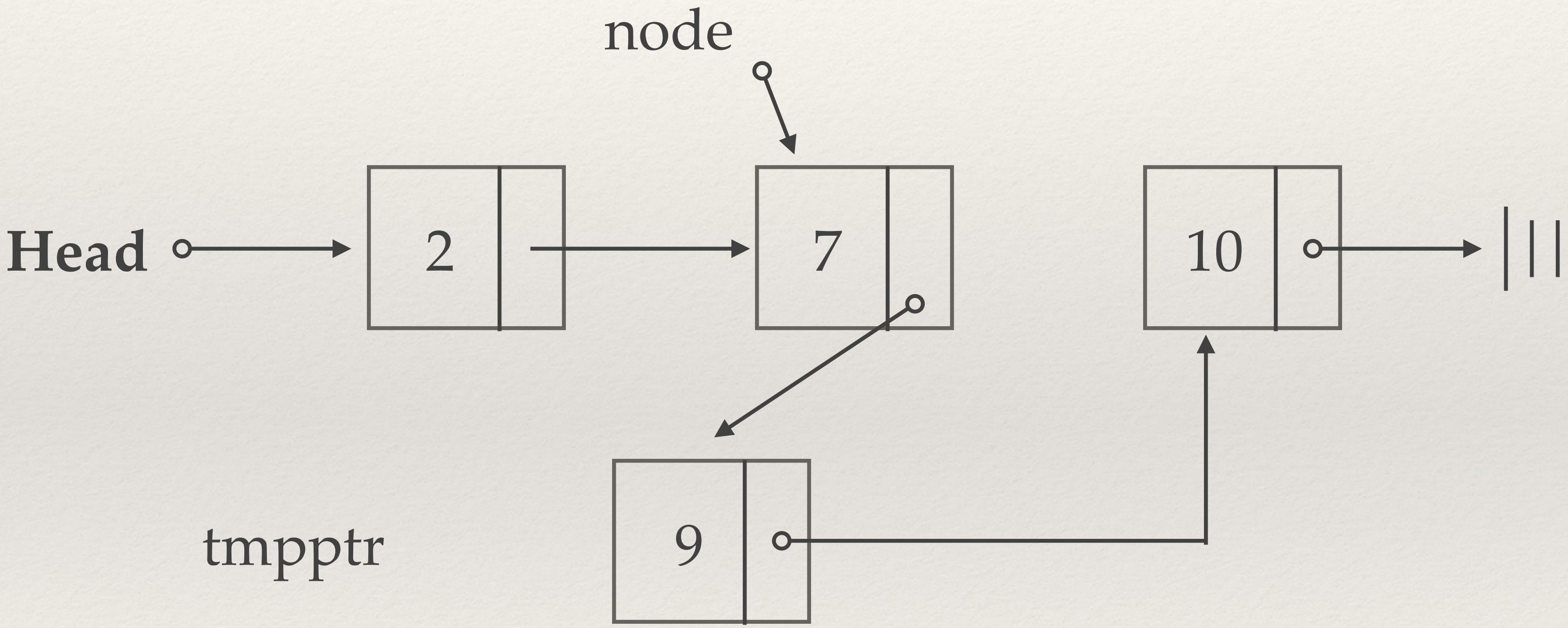


```
node->next = tmpptr;
```

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Inserting into a List

- Set the pointer from [2 |] to [5 |] instead of [7 |].



```
tmpptr = NULL;
```

Inserting into a List

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

- Putting it together

```
void insert_value(Node * before, int value){
    /* create new_node */
    Node * new_node = malloc(sizeof(Node));
    /* set new node's value */
    new_node->num = value;
    /* set new_node to point to what before is pointing to */
    new_node->next = before->next;
    /* set before to now point to the new_node */
    before->next = new_node;

    new_node = NULL;
}
```

not needed

new_node is local and will not be accessible when function exits

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Inserting into a List

- Putting it together

```
void insert_value(Node * before, int value){
    /* create new_node */
    Node * new_node = malloc(sizeof(node));
    /* set new node’s value */
    new_node->num = value;
    /* set new_node to point to what before is pointing to */
    new_node->next = before->next;
    /* set before to now point to the new_node */
    before->next = new_node;
}
```

Adding to the front of the List

- **It is also possible to add an element to front of a list**

```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
   ptr = malloc(sizeof(Node));
   ptr->num = i;
   ptr->next = head;
   head = ptr;
}
```

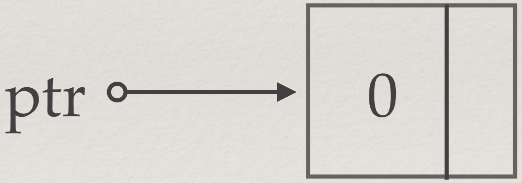
usually how list
are created

one node at a time

Adding to the front of the List

- **Create struct and place value in num.**

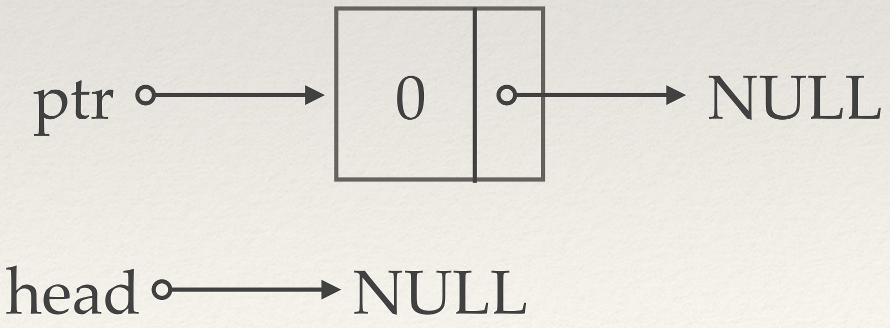
```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
    ptr = malloc(sizeof(node));
    ptr->num = i;
    ptr->next = head;
    head = ptr;
}
```



Adding to the front of the List

- **Set head to NULL and next to NULL.**

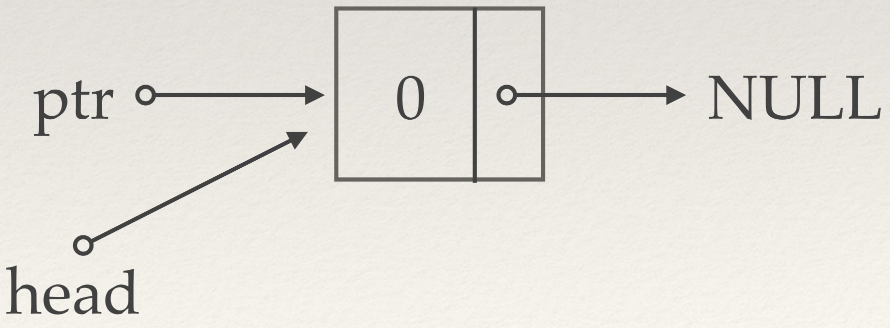
```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
    ptr = malloc(sizeof(datatype));
    ptr->num = i;
    ptr->next = head;
    head = ptr;
}
```



Adding to the front of the List

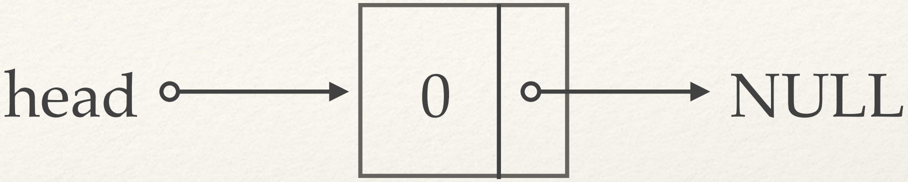
- **Set head to NULL and next to NULL.**

```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
    ptr = malloc(sizeof(datatype));
    ptr->num = i;
    ptr->next = head;
    head = ptr;
}
```



Adding to the front of the List

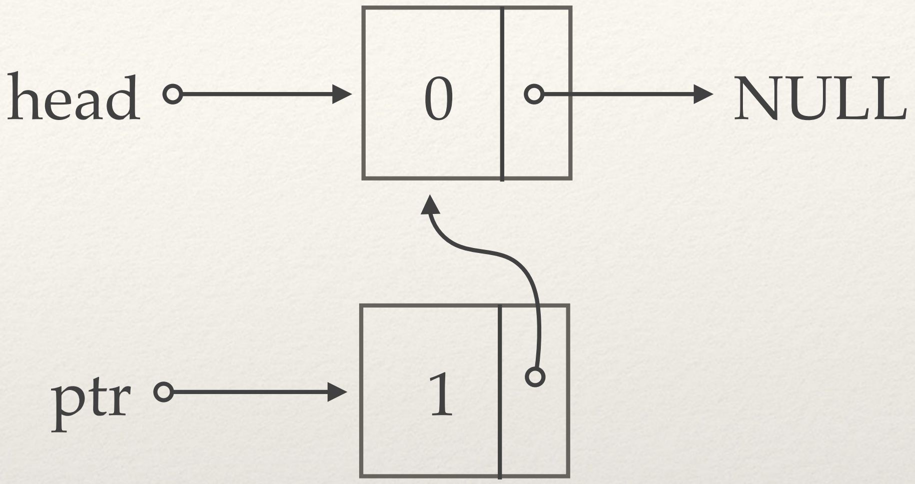
Next iteration: i = 1



```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
   ptr = malloc(sizeof(node));
   ptr->num = i;
   ptr->next = head;
   head = ptr;
}
```

Adding to the front of the List

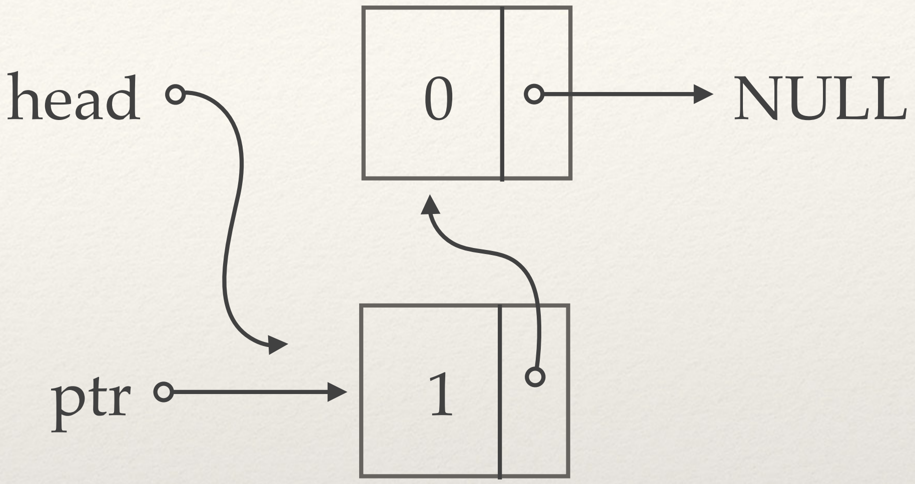
Next iteration: i = 1



```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
   ptr = malloc(sizeof(node));
   ptr->num = i;
   ptr->next = head;
   head = ptr;
}
```

Adding to the front of the List

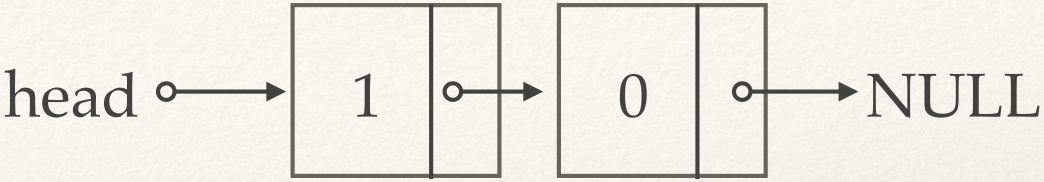
Next iteration: i = 1



```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
    ptr = malloc(sizeof(node));
    ptr->num = i;
    ptr->next = head;
    head = ptr;
}
```

Adding to the front of the List

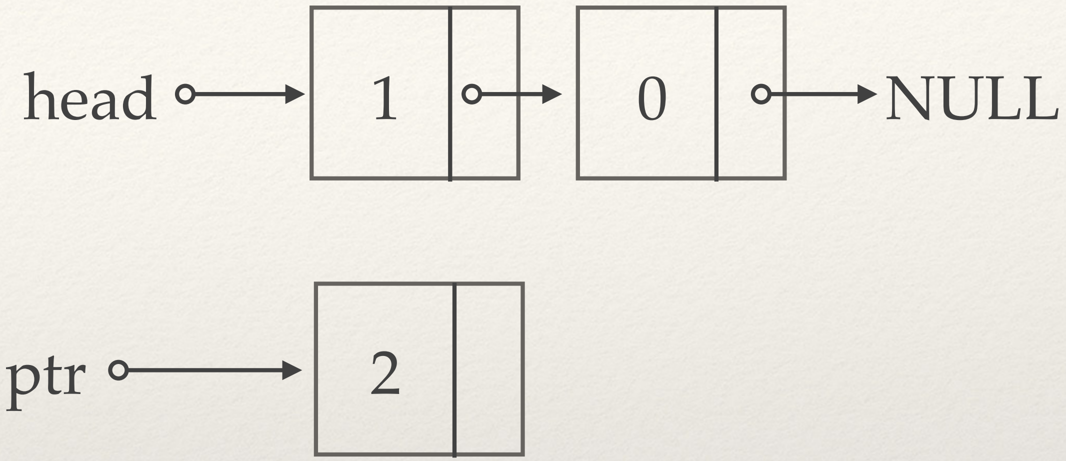
Next iteration: i = 2



```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
    ptr = malloc(sizeof(node));
    ptr->num = i;
    ptr->next = head;
    head = ptr;
}
```

Adding to the front of the List

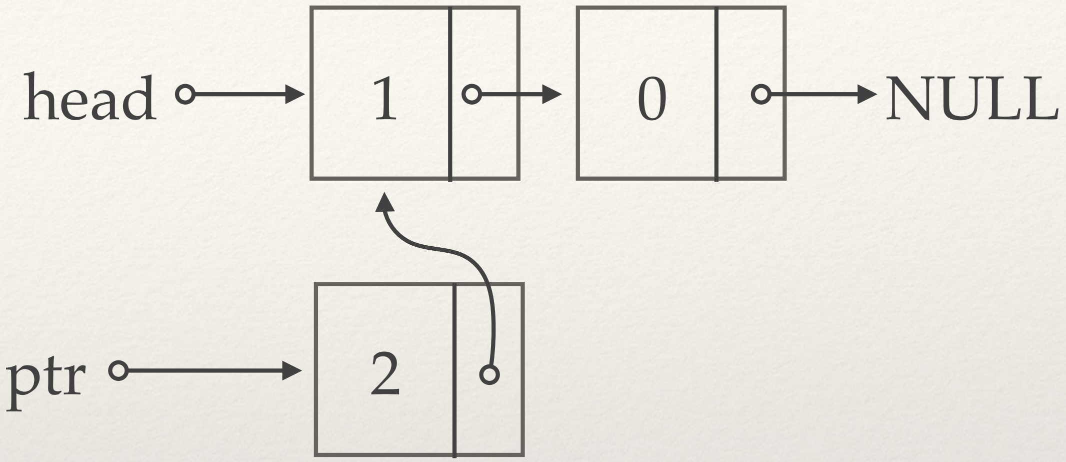
Next iteration: i = 2



```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
   ptr = malloc(sizeof(node));
   ptr->num = i;
   ptr->next = head;
   head = ptr;
}
```

Adding to the front of the List

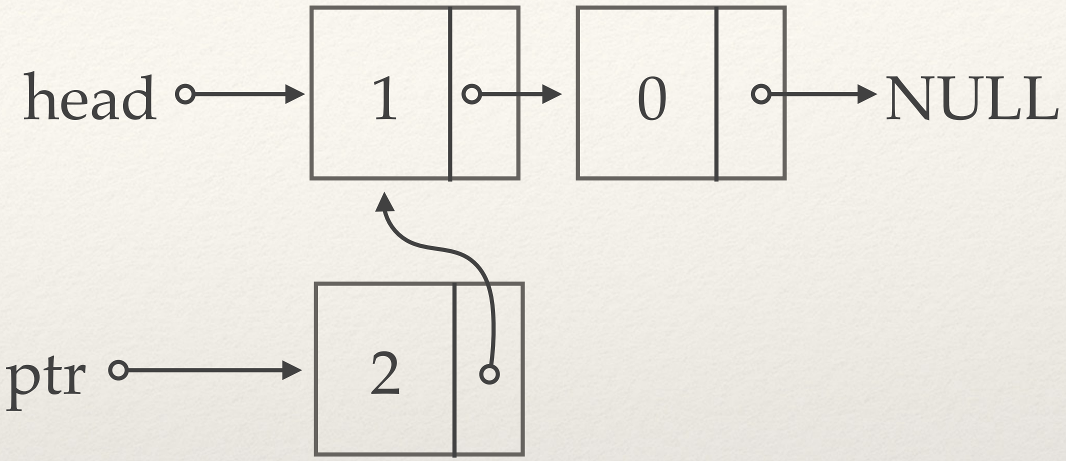
Next iteration: i = 2



```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
   ptr = malloc(sizeof(node));
   ptr->num = i;
   ptr->next = head;
   head = ptr;
}
```

Adding to the front of the List

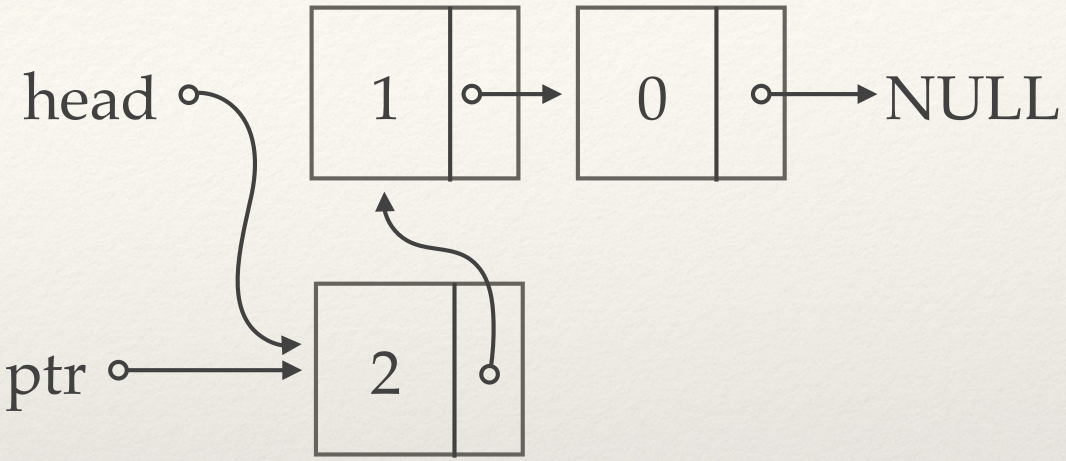
Next iteration: i = 2



```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
    ptr = malloc(sizeof(node));
    ptr->num = i;
    ptr->next = head;
    head = ptr;
}
```

Adding to the front of the List

Next iteration: i = 2



```
Node *head, *ptr;
int i;
head = ptr = NULL;
for ( i=0; i<3; i++ ) {
    ptr = malloc(sizeof(node));
    ptr->num = i;
    ptr->next = head;
    head = ptr;
}
```

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Adding to the front

- Might think this is almost the same as `insert_value(Node *, int)`

```
void add_front(Node * head, int value){
    /* create new_node */
    Node * new_node = malloc(sizeof(Node));
    /* set new node’s value */
    new_node->num = value;
    /* set new_node to point to what the head is pointing to */
    new_node->next = head;
    /* set the head to now point to the new_node */
    head = new_node;
}
```

Code has a severe bug!

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Adding to the front

- The bug becomes obvious when looking at this example:

```
Node * head = NULL;

/* a lot of code (somehow) creating the list  */
/* the first element currently set to 5       */

printf(“%d”, head->num);      → 5

add_front(head, 10);

printf(“%d”, head->num);      → 5
```

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Adding to the front

- Need to change the content of the address of the head
  (*i.e. need the address of a Node pointer, which is a double pointer*)

```
void add_front(Node ** head, int value){
     /* create new_node */
     Node *new_node = malloc(sizeof(Node));
     /* set new node’s value */
     new_node->num = value;
     /* set new_node to point to what the head is pointing to */
     new_node->next = *head;
     /* set the head to now point to the new_node */
     *head = new_node;
}
```

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Adding to the front

- Now it works

```
Node * head = NULL;

/* a lot of code (somehow) creating the list  */
/* the first element currently set to 5       */

printf(“%d”, head->num);      → 5

add_front(&head, 10);

printf(“%d”, head->num);      → 10
```

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Adding to the front

alternative approach

- Need to return the new ‘head’

  (*which can then be set outside the function*)

```
Node * add_front(Node * head, int value){
     /* create new_node */
     Node * new_node = malloc(sizeof(Node));
     /* set new node’s value */
     new_node->num = value;
     /* set new_node to point to what the head is pointing to */
     new_node->next = head;
     /* return the new_node as the new head */
     return new_node;
}
```

```
struct element {
   int num;
   struct element *next;
};

typedef struct element Node;
```

Adding to the front

alternative approach

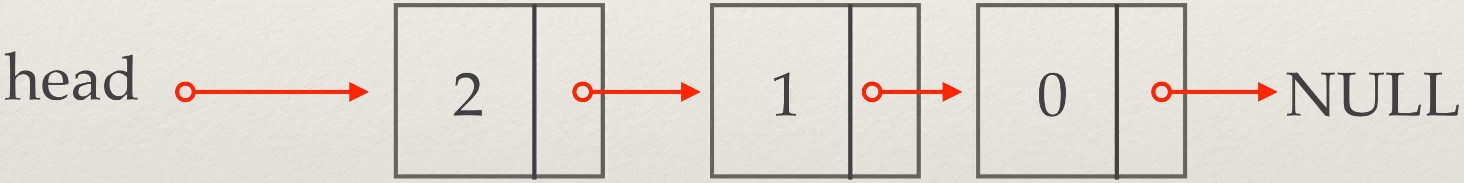
- Now it works

```
Node * head = NULL;

/* a lot of code (somehow) creating the list  */
/* the first element currently set to 5       */

printf(“%d”, head->num);      →  5

head = add_front(head, 10);

printf(“%d”, head->num);      →  10
```

```
typedef struct {
   int num;
   struct element *next;
} Node;
```

Stepping Through a List

- Start at the **head** and follow the pointers / **links**!

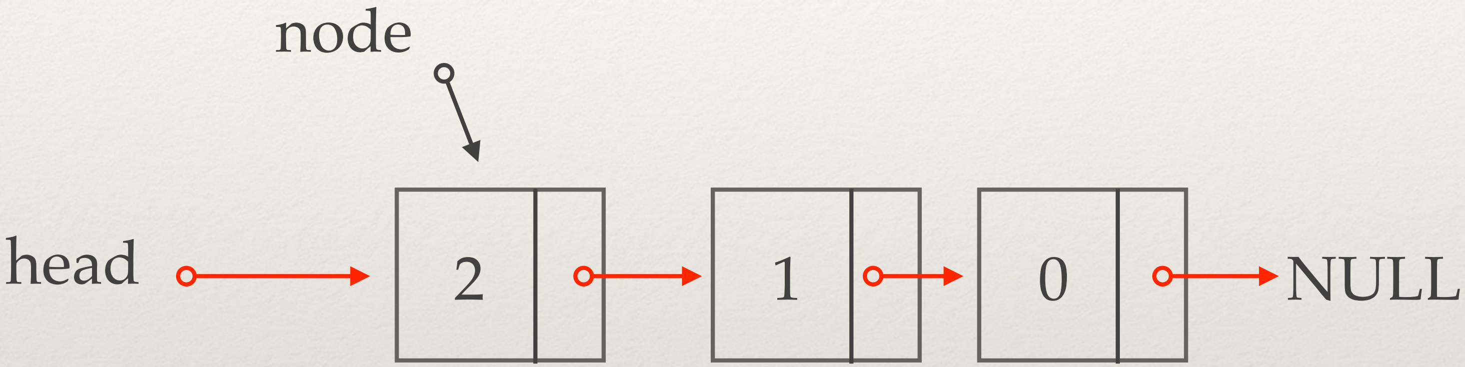


```
node = head;

while ( node != NULL ) {

   /*  code using the node goes here */

   node = node->next;

}
```

```
typedef struct {
   int num;
   struct element *next;
} Node;
```

Stepping Through a List

- Start at the **head** and follow the pointers / **links**!

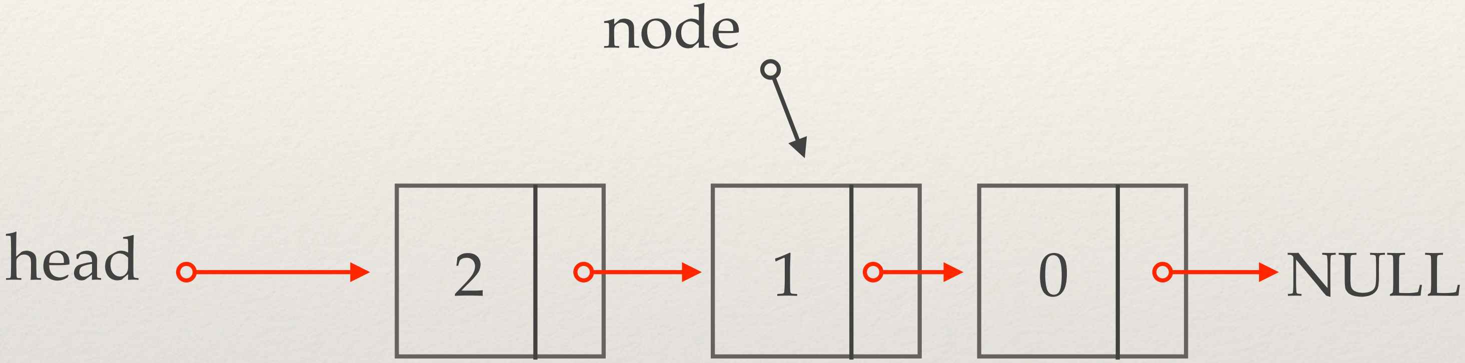


```
node = head;

while ( node != NULL ) {

   /*  code using the node goes here */

   node = node->next;

}
```

```
typedef struct {
   int num;
   struct element *next;
} Node;
```

Stepping Through a List

- Start at the **head** and follow the pointers / **links**!

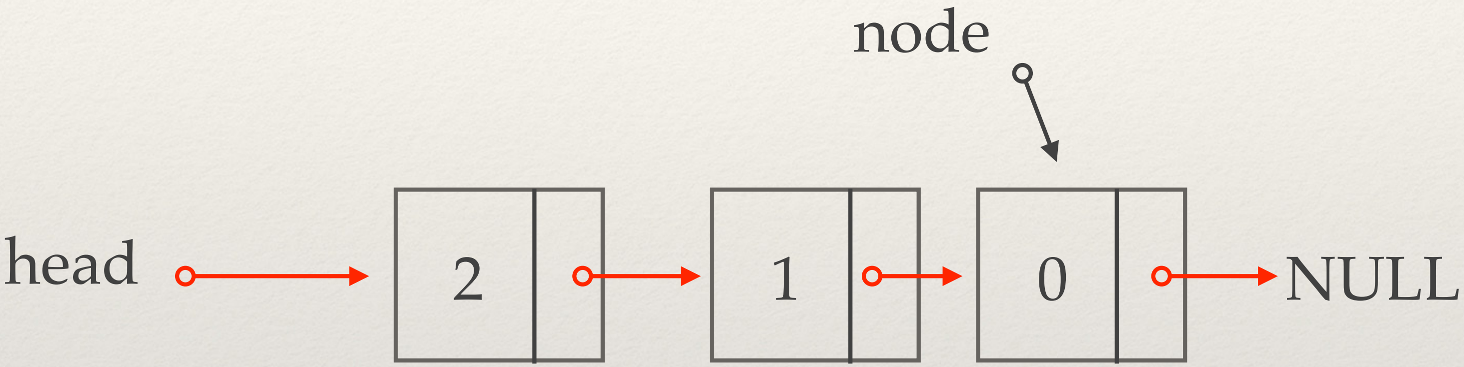


```
node = head;

while ( node != NULL ) {

    /*  code using the node goes here */

    node = node->next;

}
```

```
typedef struct {
   int num;
   struct element *next;
} Node;
```

Stepping Through a List

- Start at the **head** and follow the pointers / **links**!

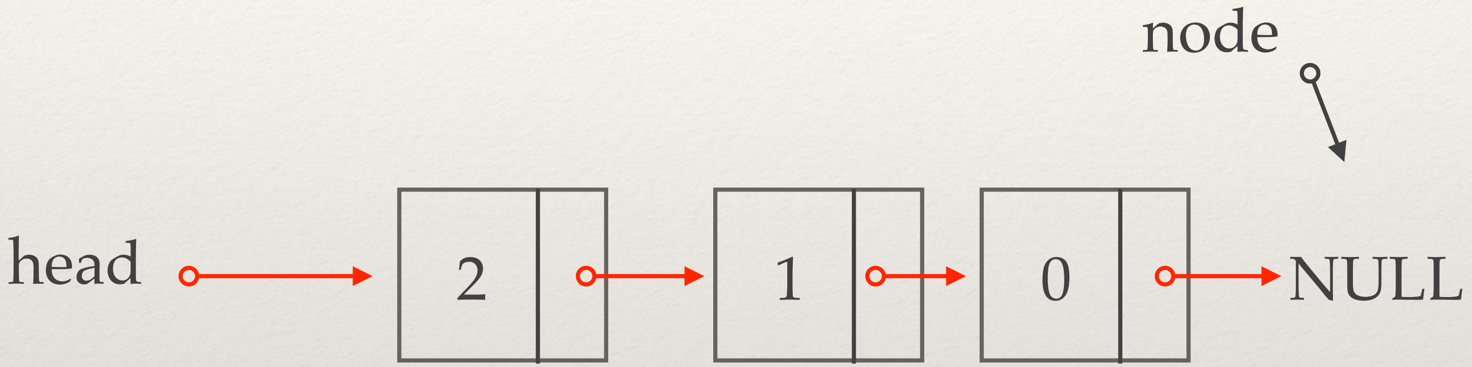


```
node = head;

while ( node != NULL ) {

    /*  code using the node goes here */

    node = node->next;

}
```

```
typedef struct {
   int num;
   struct element *next;
} Node;
```

Stepping Through a List

- Start at the **head** and follow the pointers / **links**!



```
node = head;
while ( node != NULL ) {
   /*  code using the node goes here */
   node = node->next;
}
```

```
typedef struct {
   int num;
   struct element *next;
} Node;
```

Printing a List

- Start at the **node** and follow the pointers / **links**!

  usually the head of the list is passed in

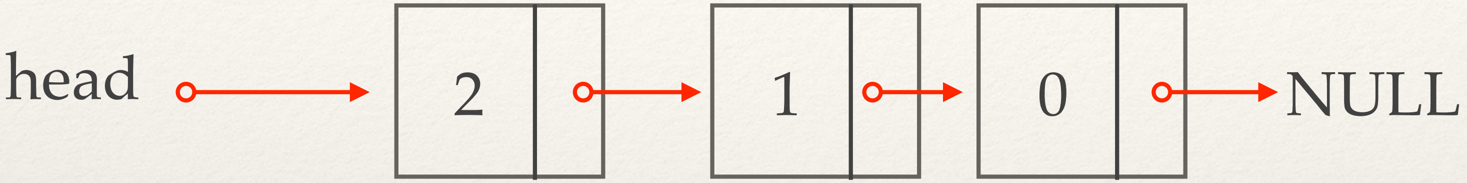
```
void print_list( Node * node ) {
    printf("%s", "< ")
    while ( node != NULL ) {
        print_node( node );
        node = node->next;
    }
    printf("%s", " >\n ");
}
```

```
void print_node( Node * node ) {
    if (node->next != NULL)
        printf ( "%d, ", node->num );
    else
        printf ( "d", node->num );
}
```

Print formatting: **< 12, 3, 47 >**

```
typedef struct {
   int num;
   struct element *next;
} Node;
```

Printing a List



```
void print_list( Node * node ) {
    printf("%s", "< ")
    while ( node != NULL ) {
        print_node( node );
        node = node->next;
    }
    printf("%s", " >\n ");
}
```

```
void print_node( Node * node ) {
    if (node->next != NULL)
        printf ( "%d, ", node->num );
    else
        printf ( "d", node->num );
}
```

```
print_list( head );          →  < 2, 1, 0 >
print_list( head->next );    →  < 1, 0 >
```

```
typedef struct {
   int num;
   struct element *next;
} Node;
```

Finding an Element

- Step through the list, looking for a value
- Keep looking if data not equal to value looked for or still haven’t hit the end of the list
- Return the node who equals the value (or NULL if not found)

```
Node * find ( Node * node, int value ){
   while ( node != NULL && node -> num != value )
         node = node -> next;
   return node;
}
```

Finding an Element

float/double version

```
typedef struct {
   double num;
   struct element *next;
} Node;
```

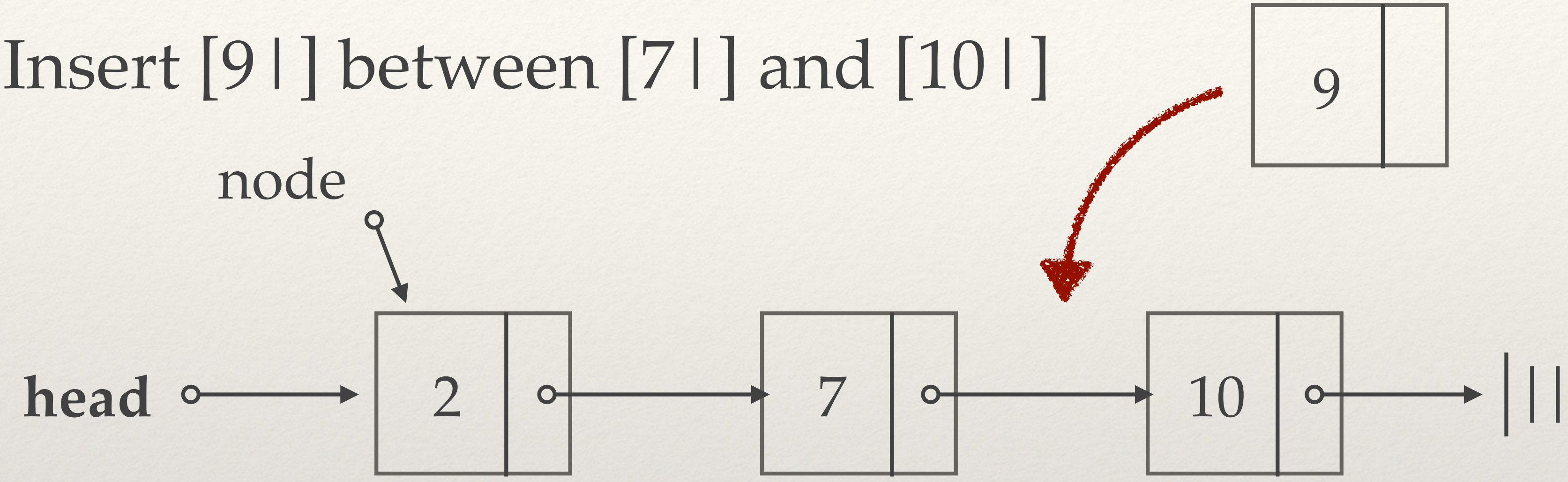
```
#include <math.h>
#define EPSILON = 0.001

int is_approx(double x, double y){
    return fabs(x - y) < EPSILON;
}

 Node * ffind ( Node * node, double value ){
    while ( node != NULL && !is_approx(node -> num, value) )
          node = node -> next;
    return node;
 }
```

```
typedef struct {
   int num;
   struct element *next;
} Node;
```

Inserting into a List

using our list functions

- Insert [9|] between [7|] and [10|]



```
print_list(head);
```

→ < 2, 7, 10 >

```
ptr = find(head, 7);
insert_value(ptr, 9);
```

```
print_list(head);
```

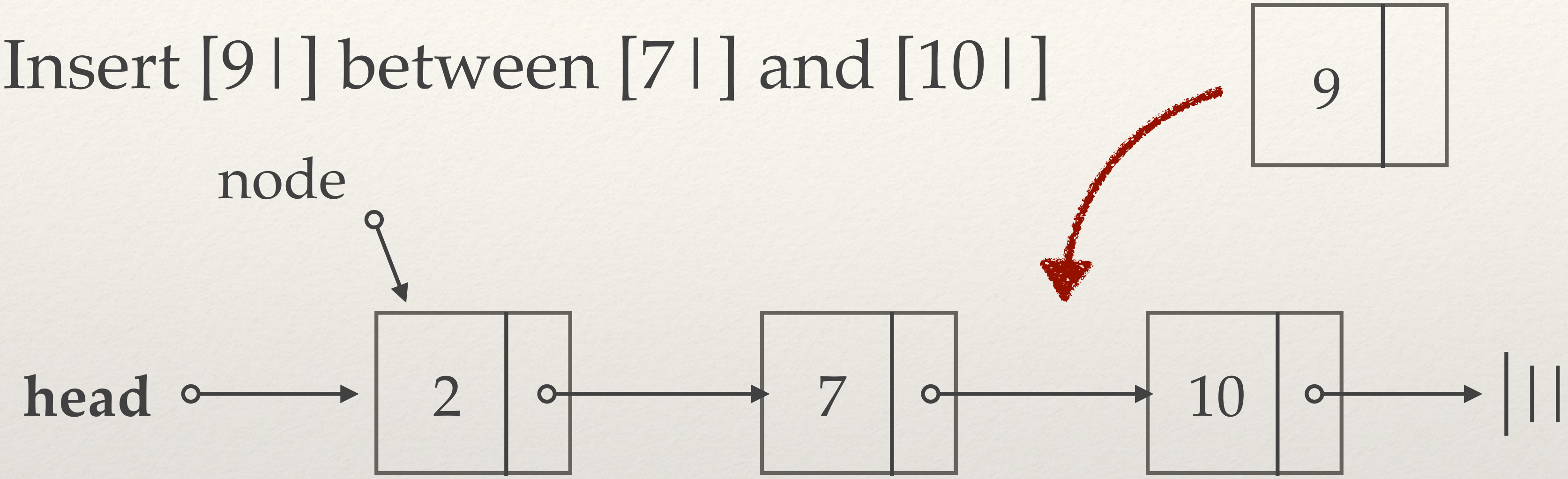
→ < 2, 7, 9, 10 >

```
typedef struct {
   int num;
   struct element *next;
} Node;
```

Inserting into a List

using our list functions

- Insert [9 |] between [7 |] and [10 |]



```
print_list(head);
```
→ **< 2, 7, 10 >**

```
insert_value(find(head, 7), 9);
```

```
print_list(head);
```
→ **< 2, 7, 9, 10 >**