

ECS 36A, May 31, 2023

Announcements

1. Homework 4 is out, as is the Gradescope.
2. Friday's lecture will be recorded and posted to Canvas
 - It will also be posted to AggieVideo
3. I will be out of town on Thursday and Friday, so office hours on Friday are cancelled.
4. Would extending the due date for homework 3 be helpful?

Breaking a Line into Alphanumeric Words

- You can use `strtok()`
 - You have to *exclude* everything except alphanumerics
 - So the end of token characters has to be *everything* but alphanumerics!
- You can use `fscanf()` and a pattern
 - If you do this, the pattern should be only alphanumerics
 - You also have to go between alphanumeric patterns as `fscanf()` will not
- There's a better way!

Breaking a Line into Alphanumeric Words

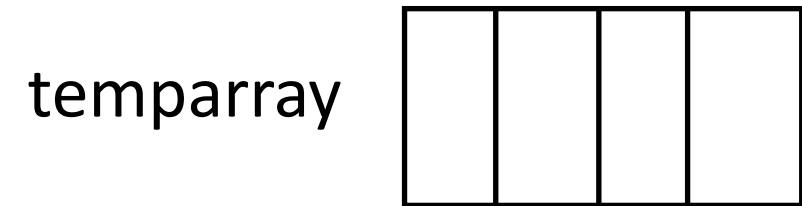
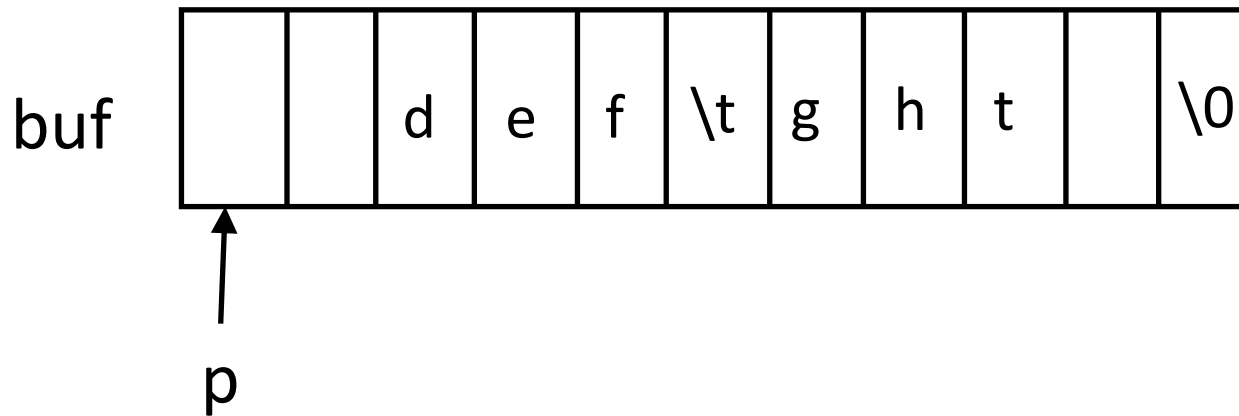
- Read one line at a time; for each line . . .
- If the first character is an alphanumeric:
 1. Advance until you do not see an alphanumeric, copying each character into an array; add a '\0' after it
 2. Insert the word into your list
 3. Advance until you see an alphanumeric or '\0'
 4. If '\0', go back up and read the next line
 5. If alphanumeric, go to 1
 6. If not, you will get to a '\0'; when you do, read the next line and go to 1

Rough Outline in Pseudocode

```
while (get there's another line)
    p points to beginning
    while(p does not point to '\0')
        while (p is not an alphanumeric and not
'\0') skip character, advance p
        if (p points to '\0')
            break
        while (p is an alphanumeric)
            copy *p into temp array, advance p
        put '\0' at end of temparray
        insert word in temparray into the list
```

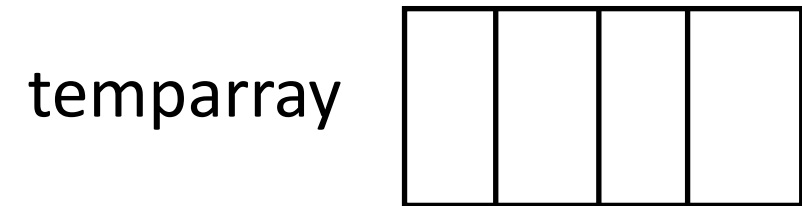
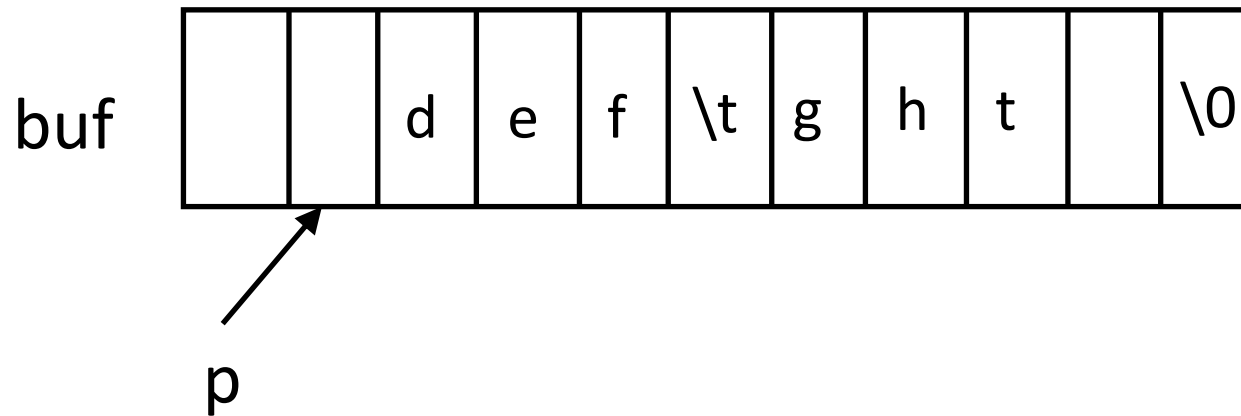
In Pictures

- Input line " def ght " (not including quotes)



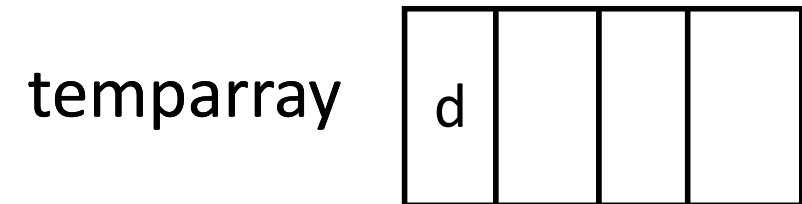
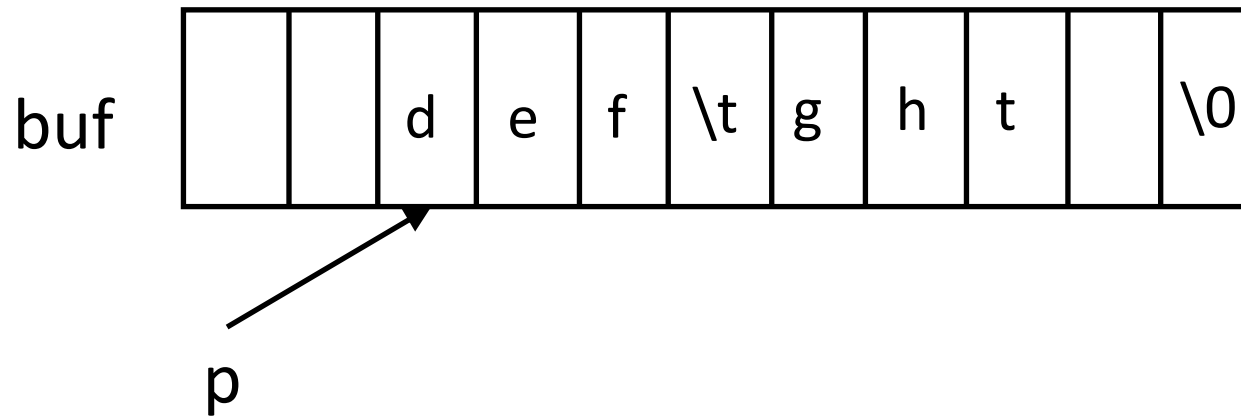
In Pictures

- Input line " def ght " (not including quotes)



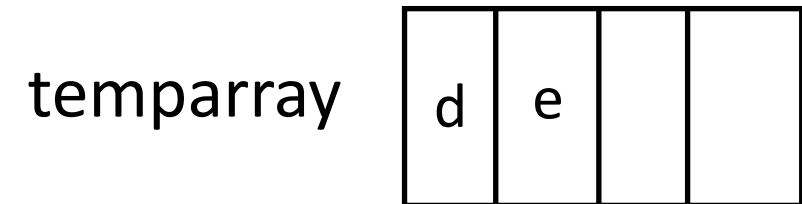
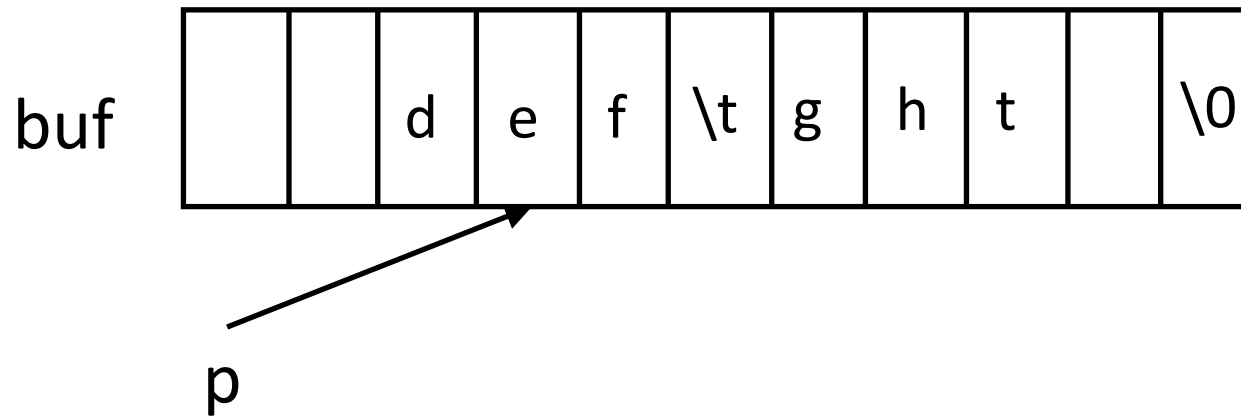
In Pictures

- Input line " def ght " (not including quotes)



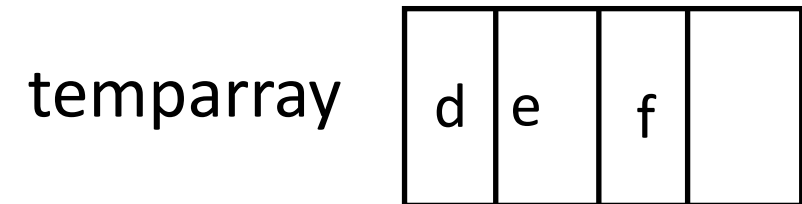
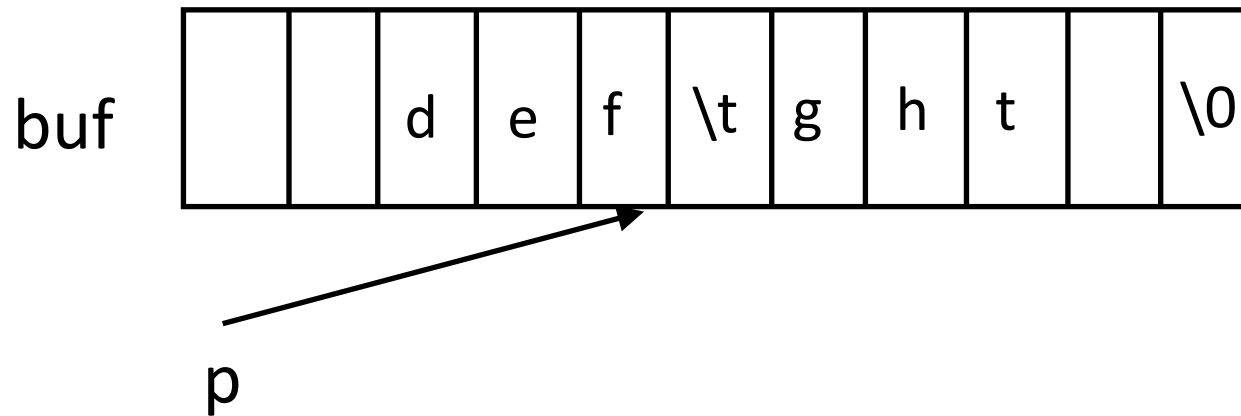
In Pictures

- Input line " def ght " (not including quotes)



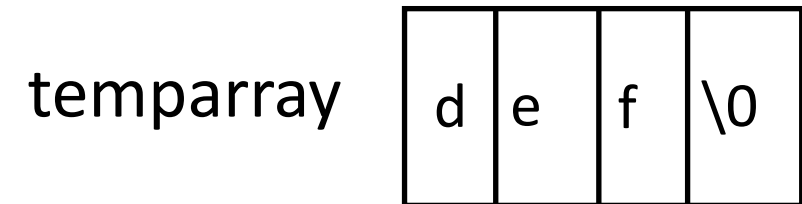
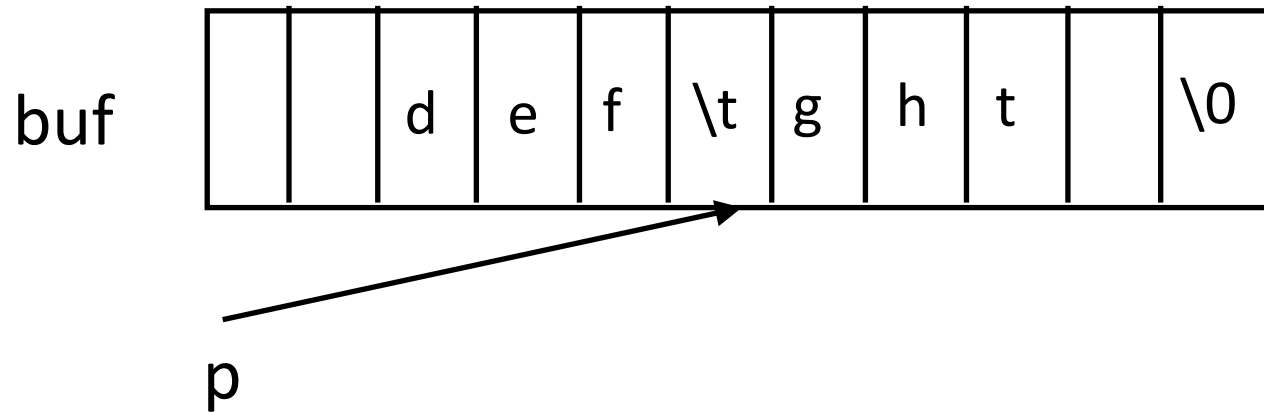
In Pictures

- Input line " def ght " (not including quotes)



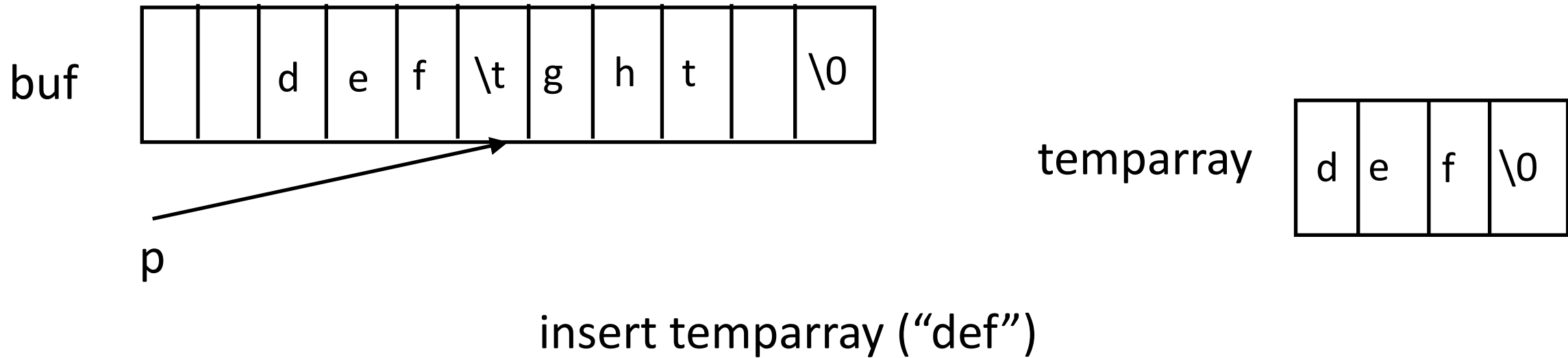
In Pictures

- Input line " def ght " (not including quotes)



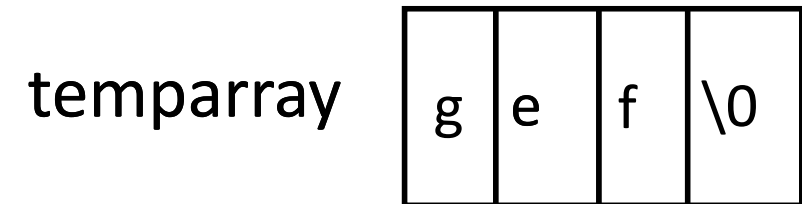
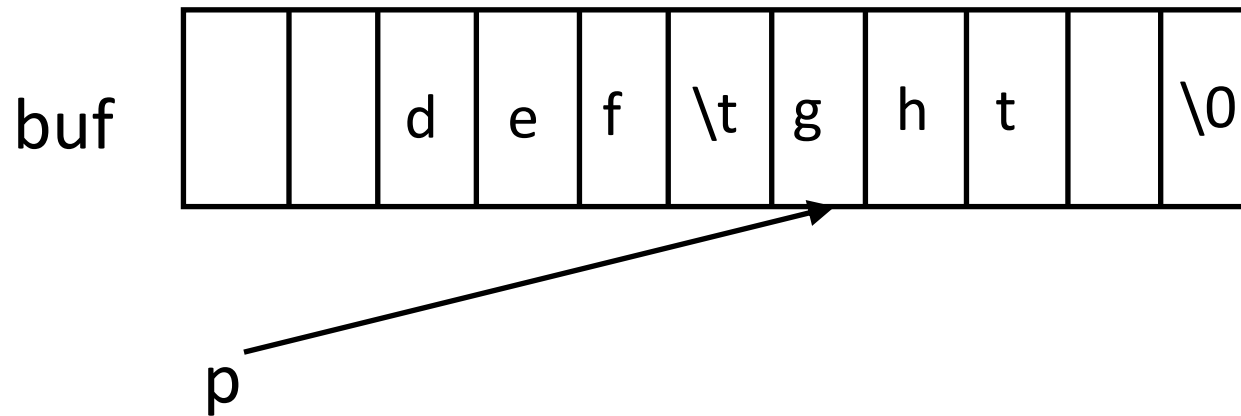
In Pictures

- Input line " def ght " (not including quotes)



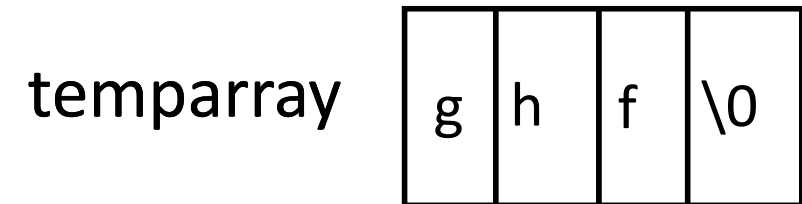
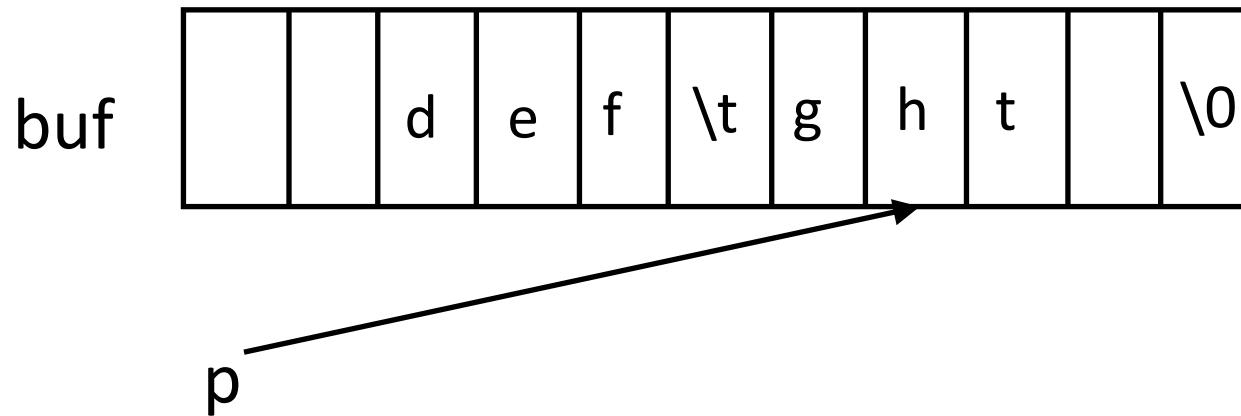
In Pictures

- Input line " def ght " (not including quotes)



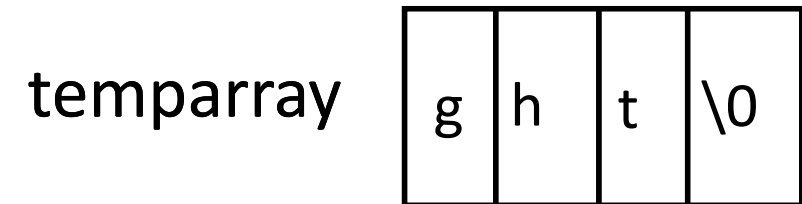
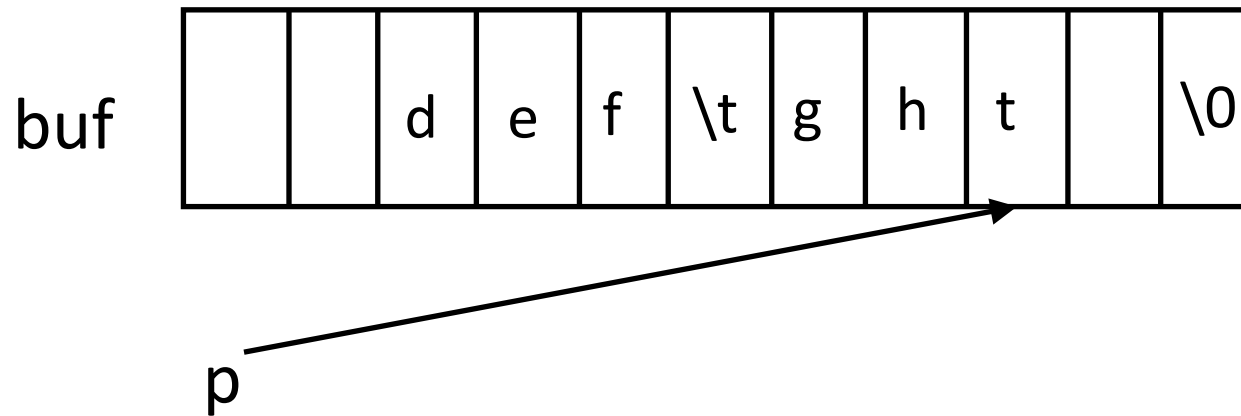
In Pictures

- Input line " def ght " (not including quotes)



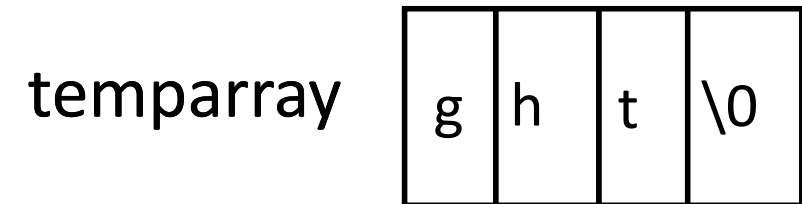
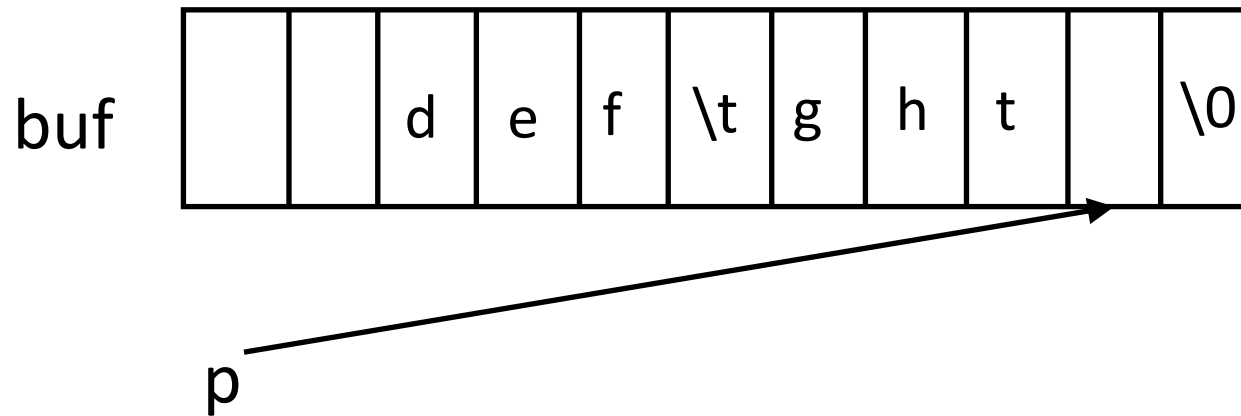
In Pictures

- Input line " def ght " (not including quotes)



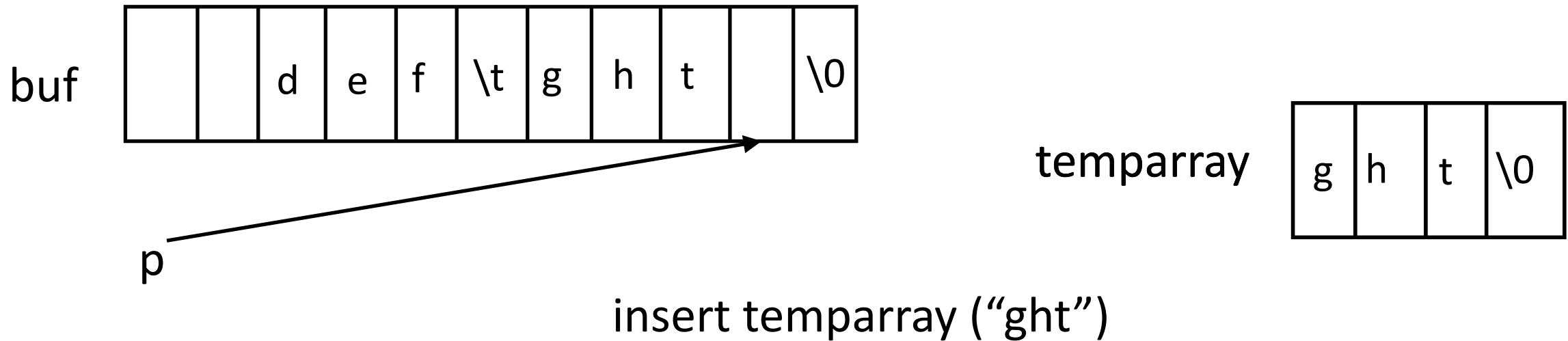
In Pictures

- Input line " def ght " (not including quotes)



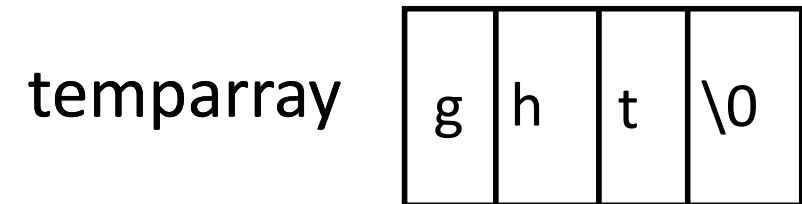
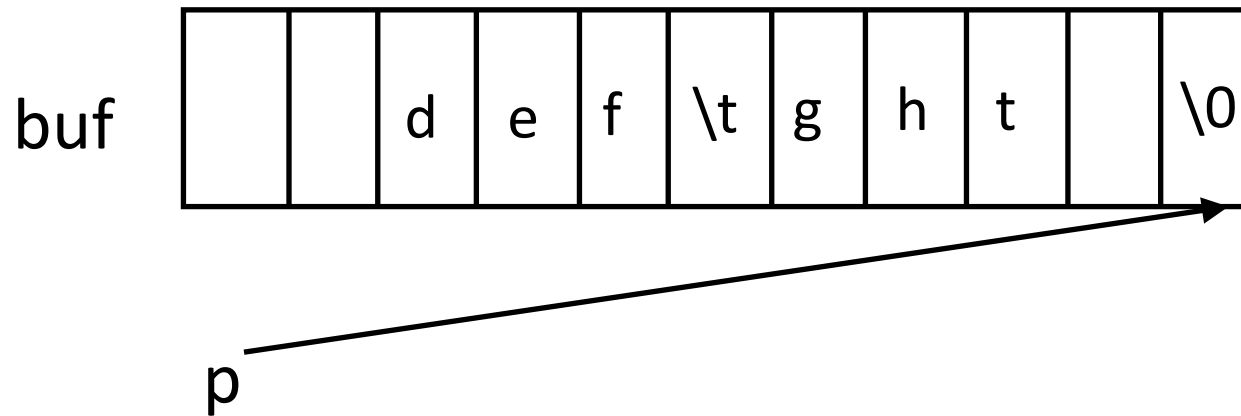
In Pictures

- Input line " def ght " (not including quotes)



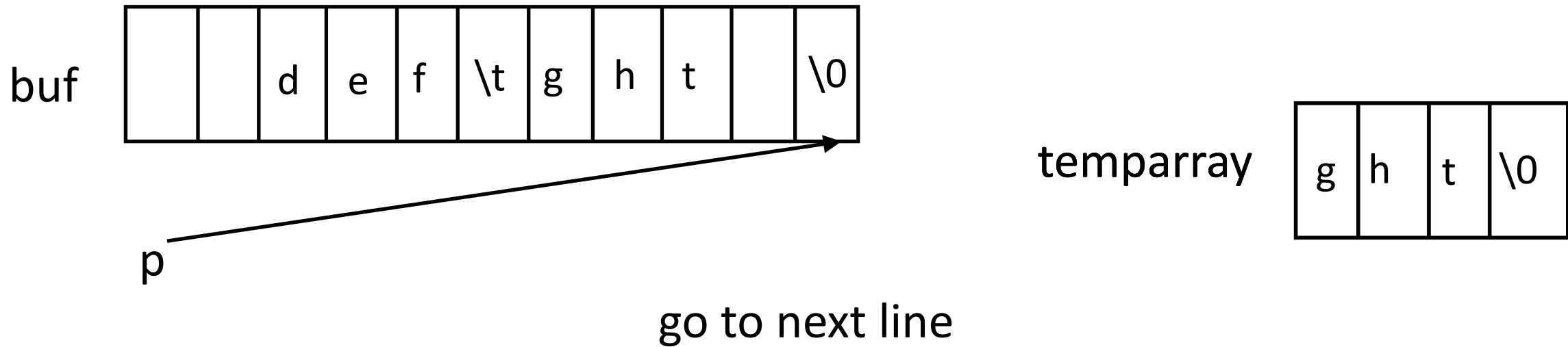
In Pictures

- Input line " def ght " (not including quotes)



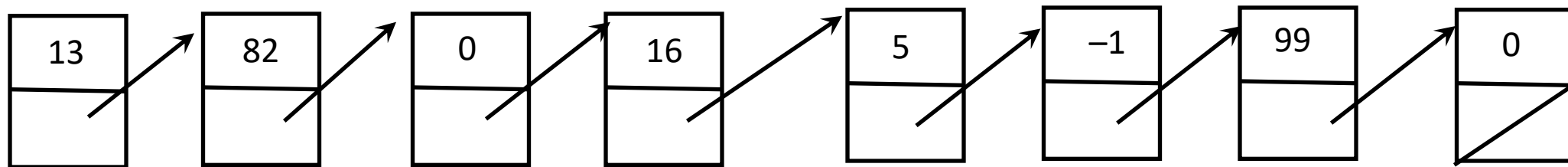
In Pictures

- Input line " def ght " (not including quotes)



Linked List


- A list composed of instantiations of structures
 - One element is whatever is to be sorted (int, for us)
 - Another element is a pointer to the next element; NULL if none



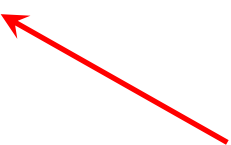
Structure for This List

```
struct node {  
    int num;  
    struct node *next;  
};  
struct node *list;
```

This holds the integer
that you read in



This holds the pointer
to the next element
in the linked list; it's
NULL if it's at the end



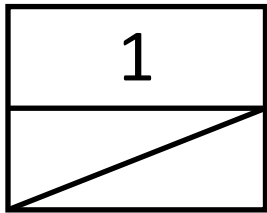
This points to the first
element of the list



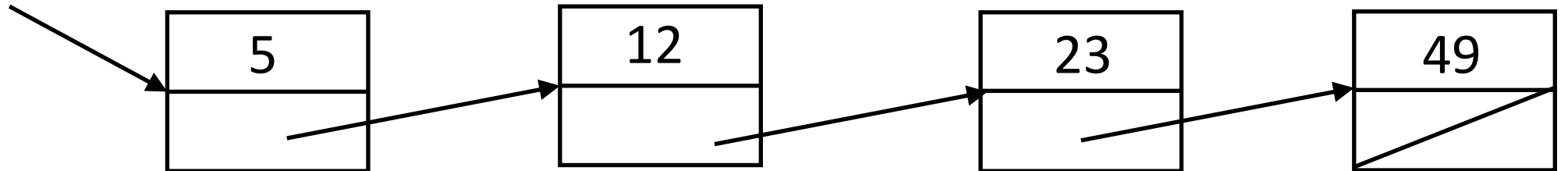
Changing How Memory Is Allocated

- Now you can allocate memory one element (“node”) at a time
- Insertion at beginning is like this (see “linked.c”, ll. 72–76):
 - `new->next = first;`
 - `list = new;`
- Insertion in the middle between `prev` and `succ` is (see “linked.c”, ll. 78–97):
 - `new->next = succ;`
 - `prev->next = new;`
- Insertion at the end nomore of the list (same as above):
 - `nomore->next = new;`

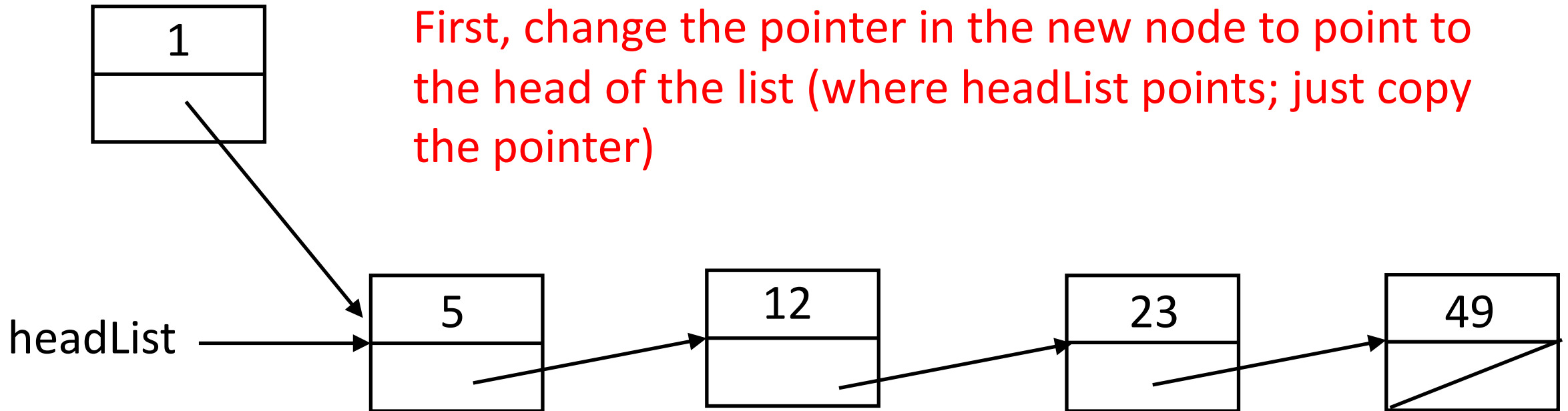
Insertion



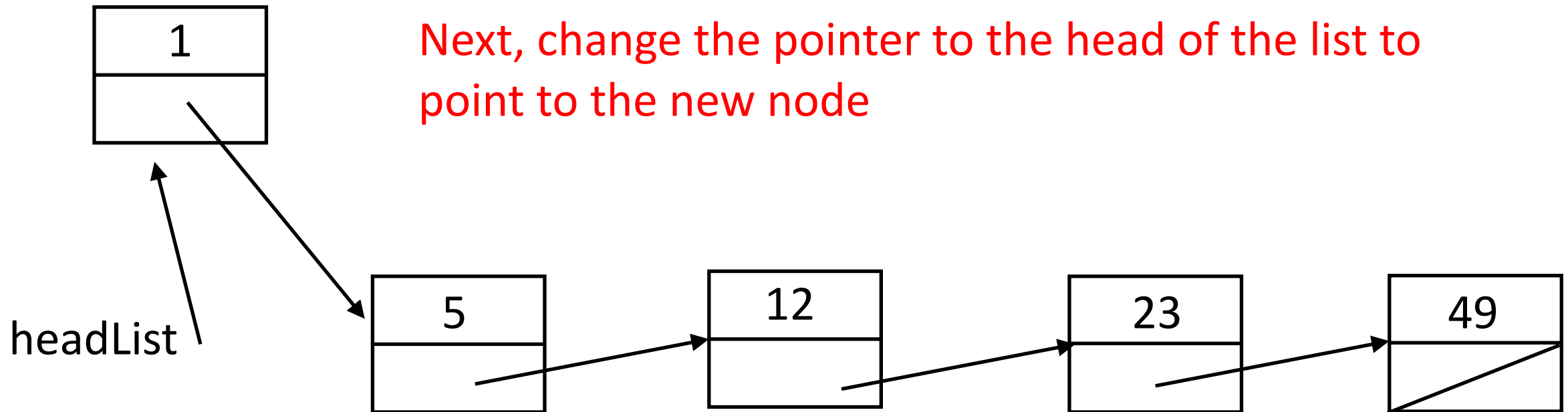
headList



Insertion: At the Beginning of the List



Insertion: At the Beginning of the List



Code for This

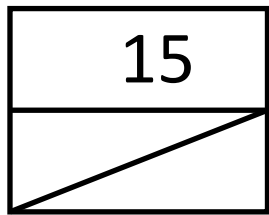
- `new` is a pointer to the new node, `headList` points to the head of the list
- First, make `new` point to the old head. of the list

```
new->next = headList;
```

- Next, make the pointer to the head of the list point to `new`

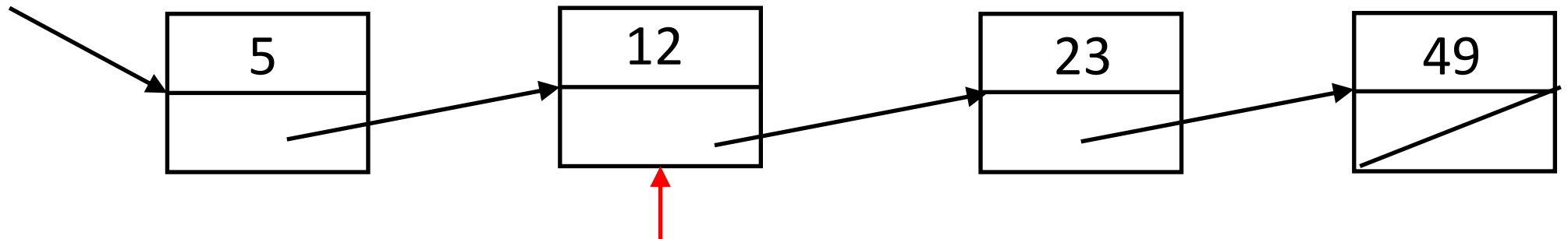
```
headList = new;
```

Insertion: In the Middle of the List



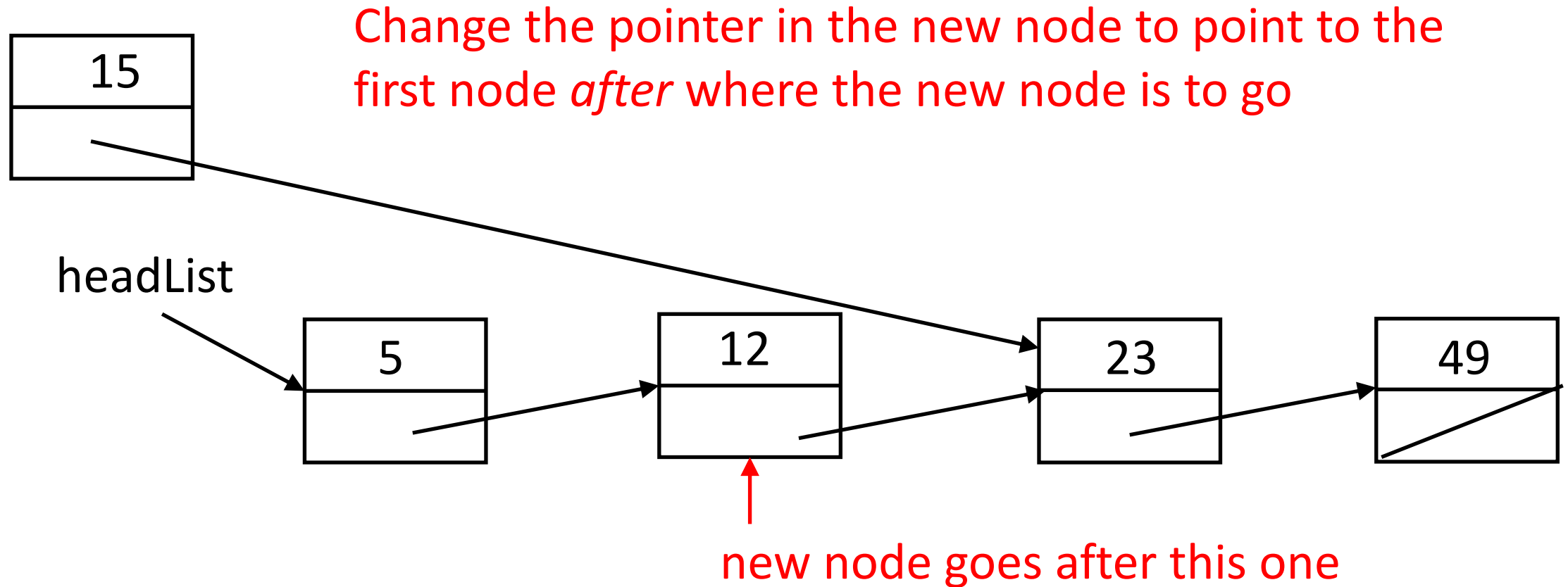
First, scan down the list until you reach the node before which the new node goes.

headList

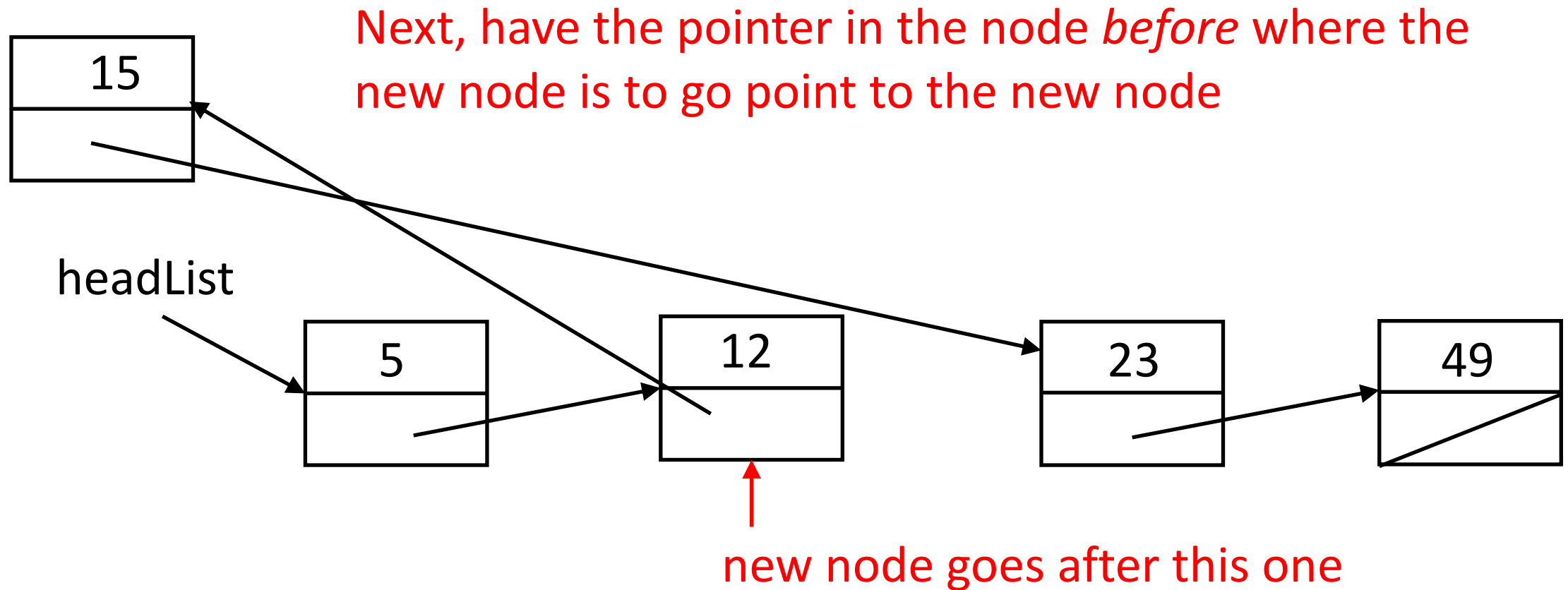


new node goes after this one

Insertion: In the Middle of the List



Insertion: In the Middle of the List



Code for This

- `new` is a pointer to the new node, `headList` points to the head of the list, and `p` is a pointer to node
- First, find the node that `new` goes after

```
for (p = headList;  
     p != NULL && p->next < new->next;  
     p = p->next)  
    /* do nothing ;
```

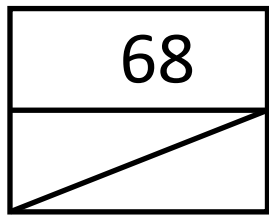
- Next, change the pointer in `new` to point to the node *after* where this one goes

```
new->next = p->next;
```

- Finally, make the node `p` points to point to `new`

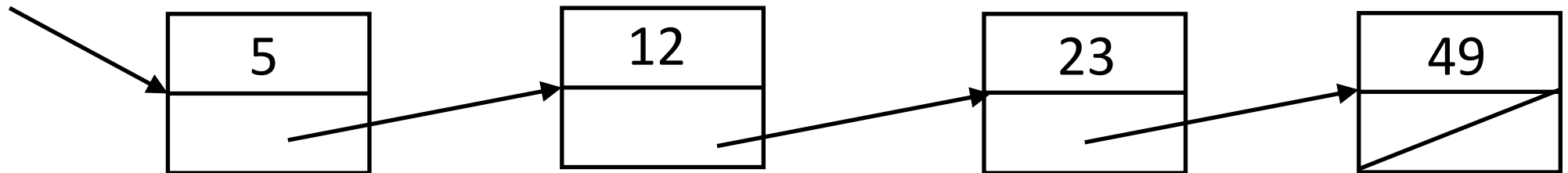
```
p->next = new;
```

Insertion: At the End of the List



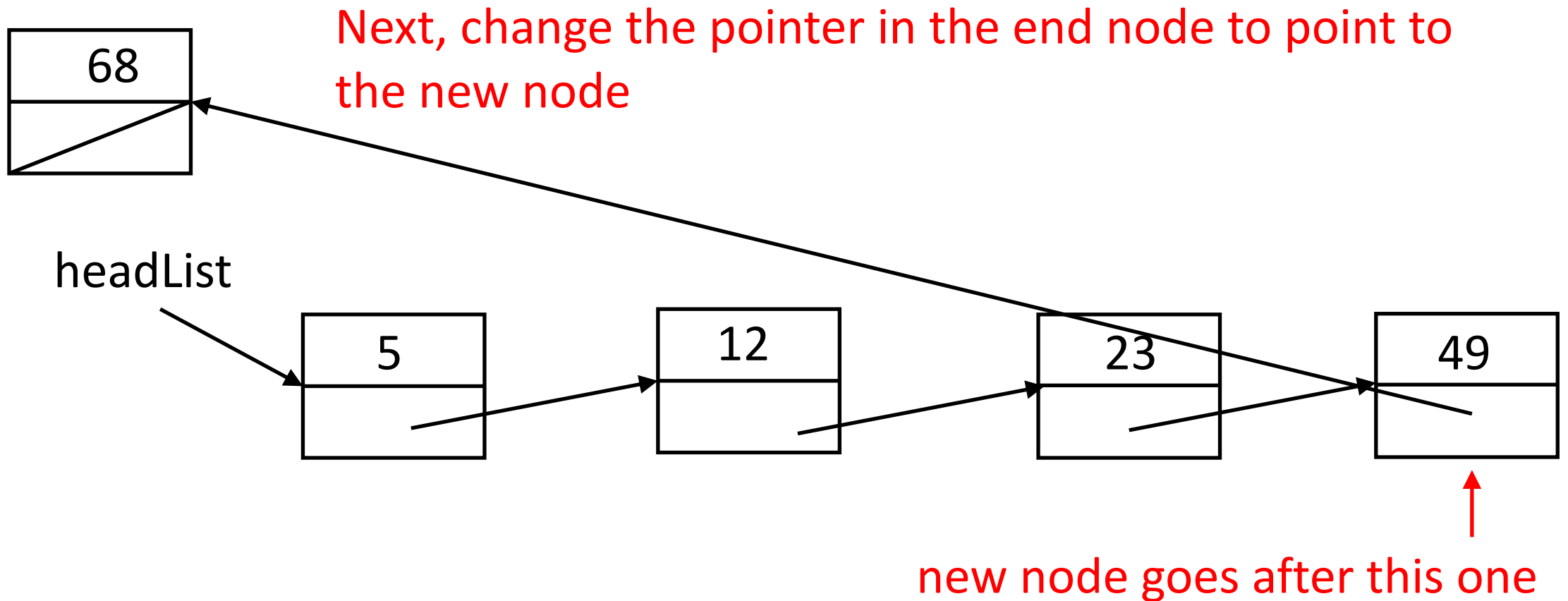
First, scan down the list until you reach the end node

headList



new node goes after this one

Insertion: At the End of the List



Code for This

- `new` is a pointer to the new node, `headList` points to the head of the list, and `p` is a pointer to node

- First, find the node at the end

```
for (p = headList;  
     p != NULL && p->next != NULL;  
     p = p->next)  
    /* do nothing */;
```

- Next, change the pointer in what `p` points to to point to `new`

```
p->next = new;
```

- This may be an excess, but make sure `new`'s pointer field is `NULL`

```
new->next = NULL;
```

Sorting

- Function is:

```
void qsort(void *base, size_t nmemb, size_t size,  
           int (*compar)(const void *, const void *));
```

- Here *compar* is function that takes 2 pointers to elements of the array *base*, with *nmemb* members of size *size*
- *compar* returns negative if first is less than second; 0 if the two are equal; and positive if the first is greater than the second
- You supply *compar*

Example *compar*

```
int cmp(const void *x, const void *y)
{
    int *px, *py;
    px = (int *) x;
    py = (int *) y;

    return(*px - *py);
}
```

Calling *qsort*

```
int arr[100]; /* array of integers to be sorted */
int narr;    /* number of integers in arr */
/* ... put random numbers into arr */
/* now sort them */
qsort(arr, narr, sizeof(int), (int (*)(const void *, const void *)) cmp);
```