

Linked Lists in C and C++

Professor Hugh C. Lauer

CS-2303, System Programming Concepts

(Slides include materials from *The C Programming Language*, 2nd edition, by Kernighan and Ritchie, *Absolute C++*, by Walter Savitch, *The C++ Programming Language*, Special Edition, by Bjarne Stroustrup, and from *C: How to Program*, 5th and 6th editions, by Deitel and Deitel)

Common Data Structures in C and C++

■ Linked lists – Nothing specific in K&R

- One-way
- Doubly-linked
- Circular

■ Trees –K&R §6.5

- Binary
- Multiple branches

■ Hash Tables – K&R §6.6

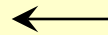
- Combine arrays and linked list
- Especially for searching for objects by value

Definitions

■ *Linked List*

- A *data structure* in which each element is dynamically allocated and in which elements point to each other to define a *linear* relationship
- Singly- or doubly-linked
- Variations: *stack, queue, circular list*

Note: elements are *usually* the same type (but not always).



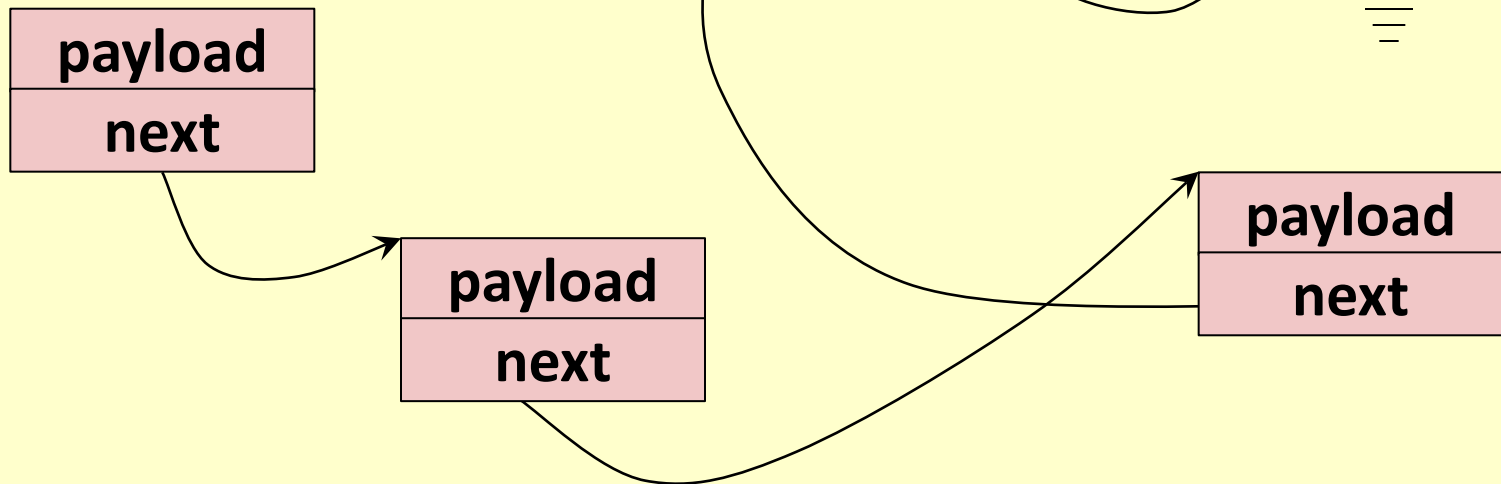
■ *Tree*

- A *data structure* in which each element is dynamically allocated and in which each element has more than one potential *successor*
- Defines a *partial order*

Linked List

```
struct listItem {  
    type payload;  
    struct listItem *next;  
};
```

Note: payload may be multiple members.



Linked List (continued)

- **Items of list are usually same type**
 - Generally obtained from *malloc()*
- **Each item points to next item**
- **Last item points to null**
- **Need “*head*” to point to first item!**

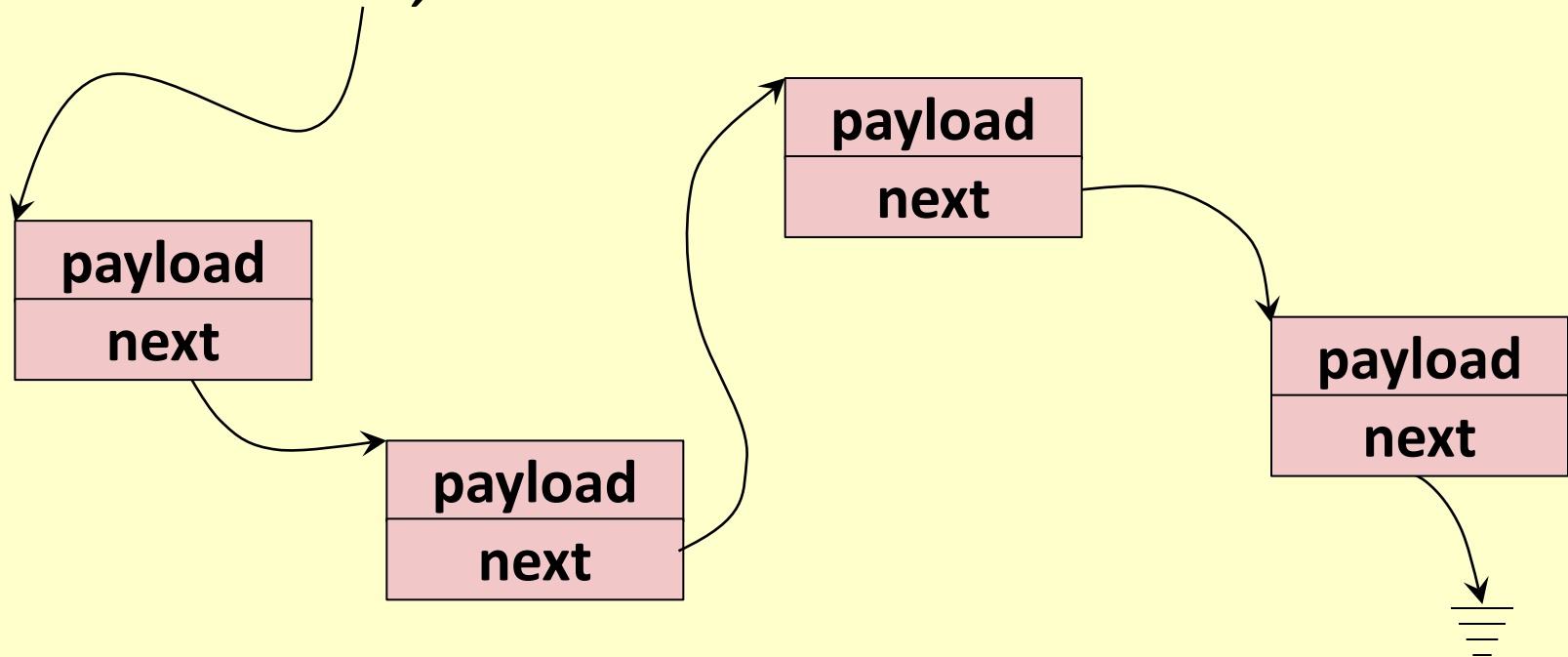
- **“Payload” of item may be almost anything**
 - A single member or multiple members
 - Any type of object whose size is known at compile time
 - Including *struct*, *union*, *char ** or other pointers
 - Also arrays of fixed size at compile time (see p. 214)

Usage of Linked Lists

- **Not massive numbers of items**
 - Linear search is okay
- **Sorting not usually necessary**
 - or sometimes not possible
- **Need to add and delete data “on the fly”**
 - Even from middle of list
- **Items often need to be added to or deleted from the “ends”**

Linked List (continued)

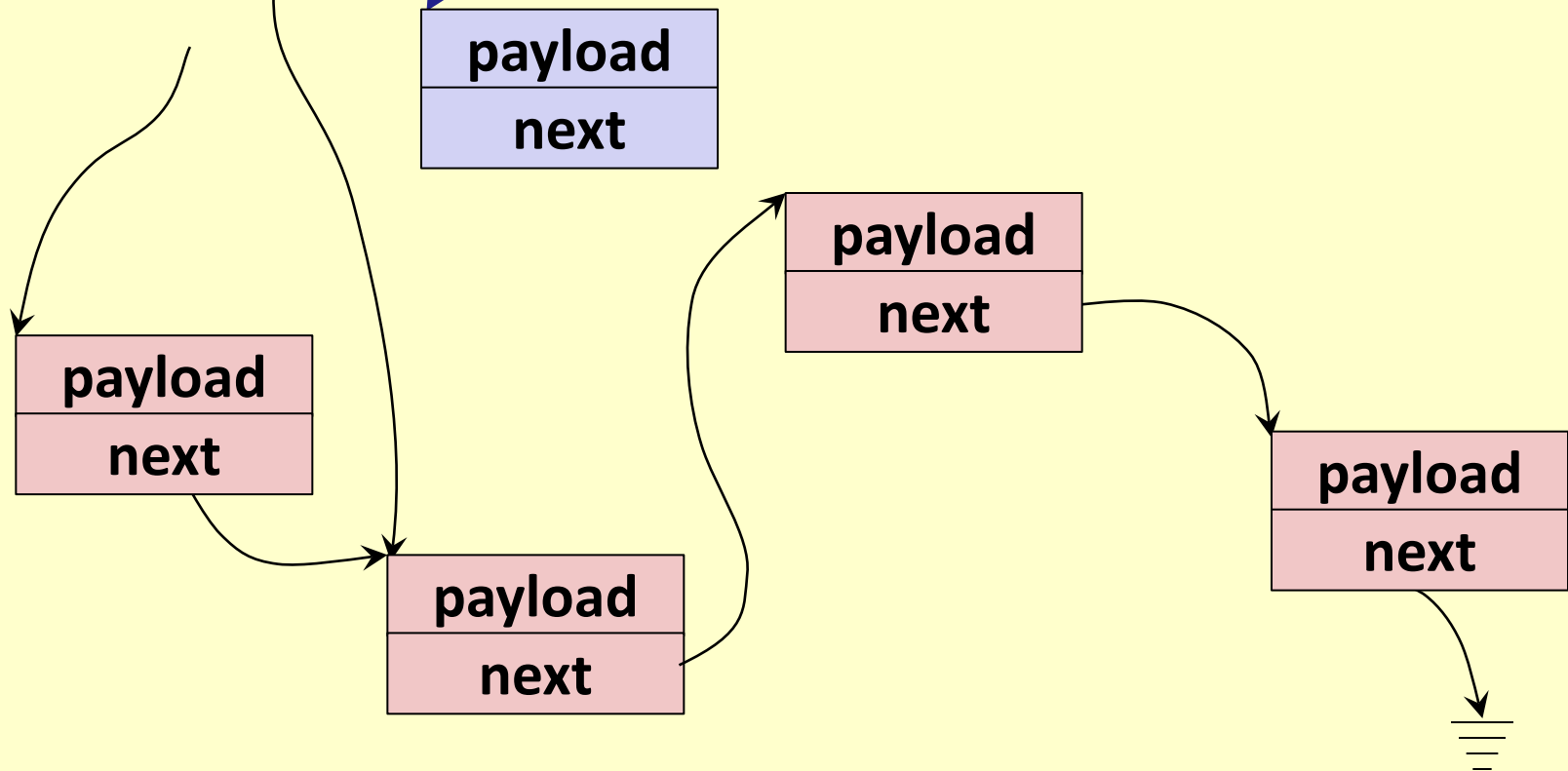
```
struct listItem {  
    type payload;  
    struct listItem *next;  
};  
struct listItem *head;
```



Adding an Item to a List

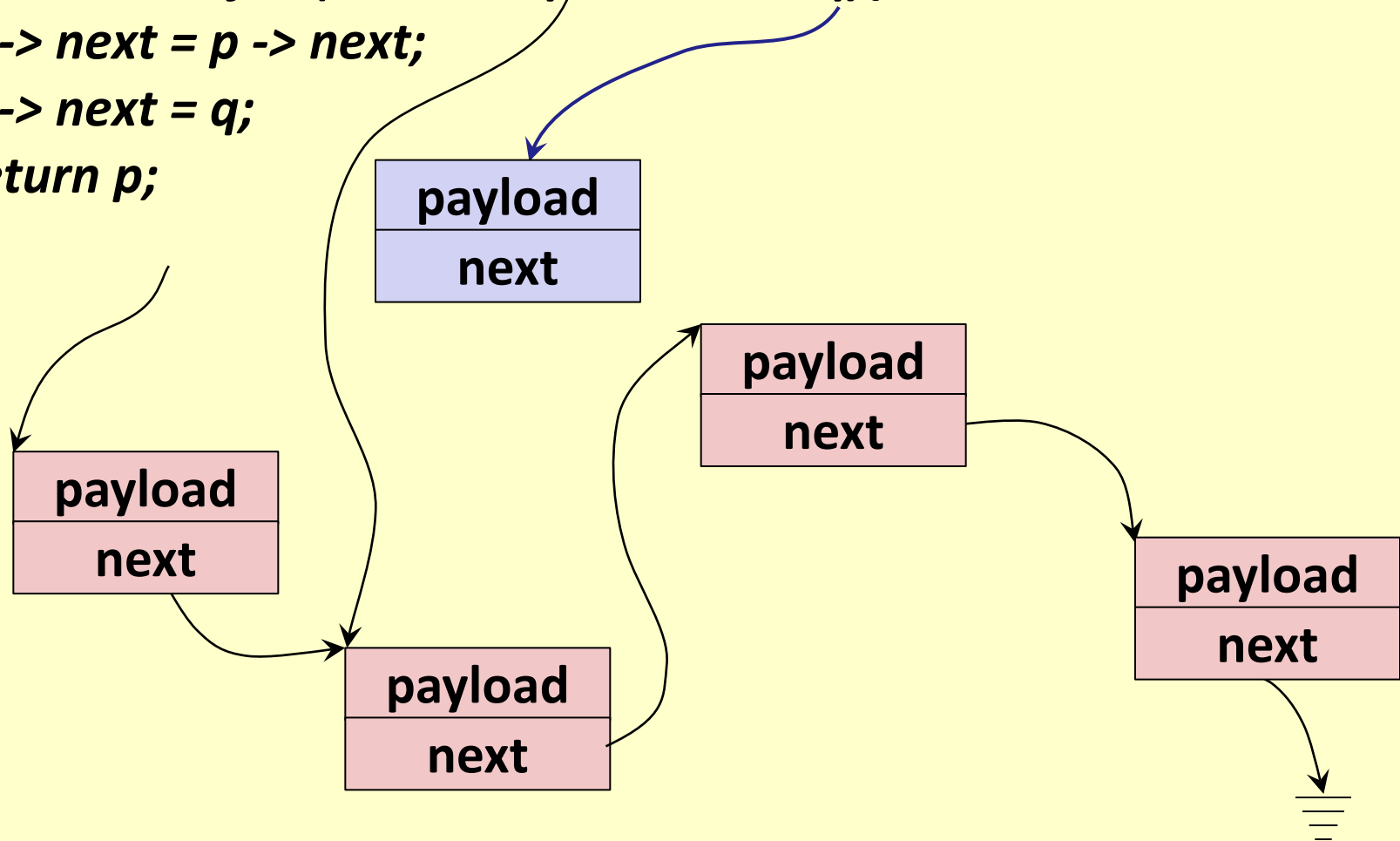
*struct listItem *p, *q;*

- Add an item pointed to by *q* after item pointed to by *p*
 - Neither *p* nor *q* is *NULL*



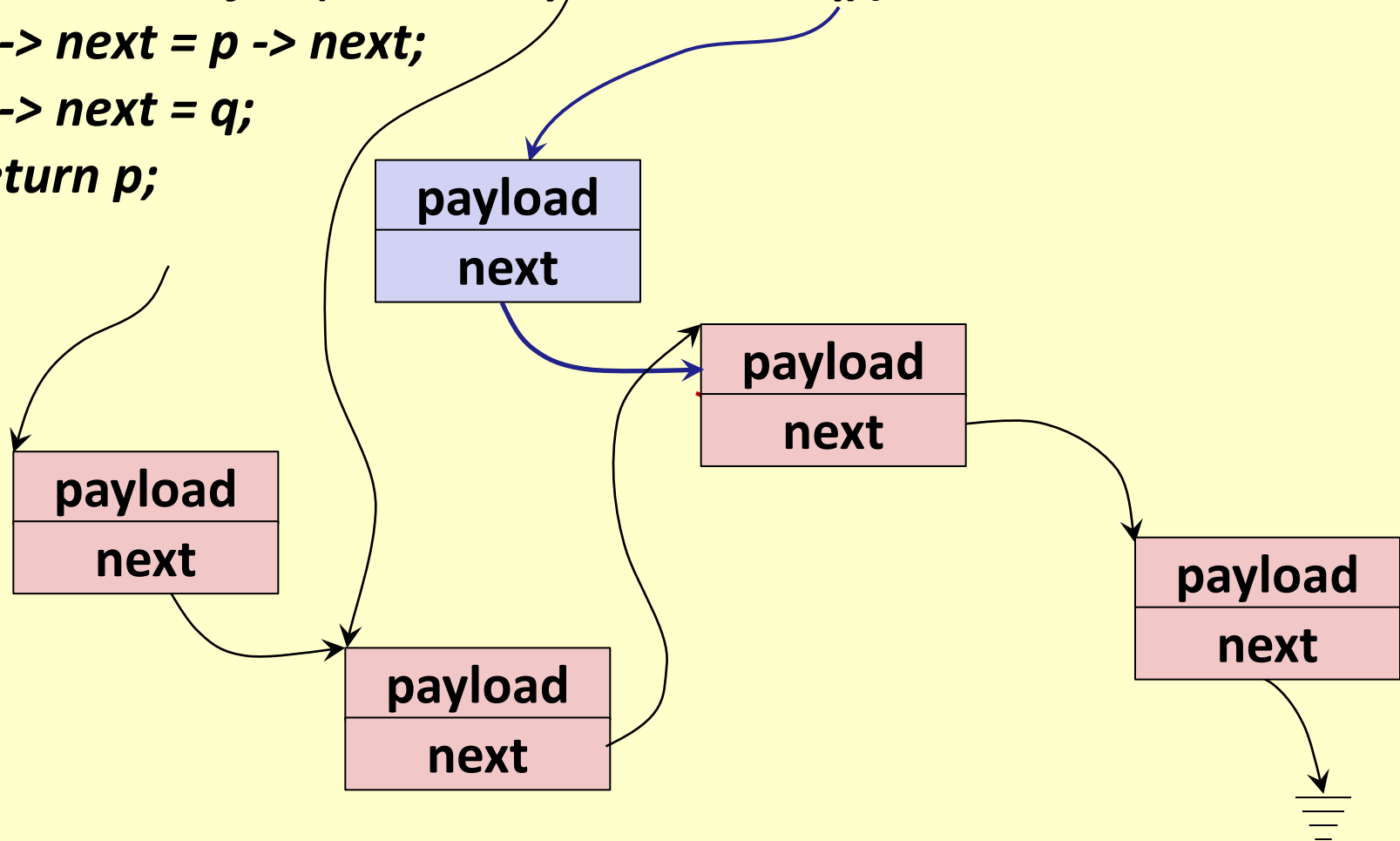
Adding an Item to a List

```
listItem *addAfter(listItem *p, listItem *q){  
    q -> next = p -> next;  
    p -> next = q;  
    return p;  
}
```



Adding an Item to a List

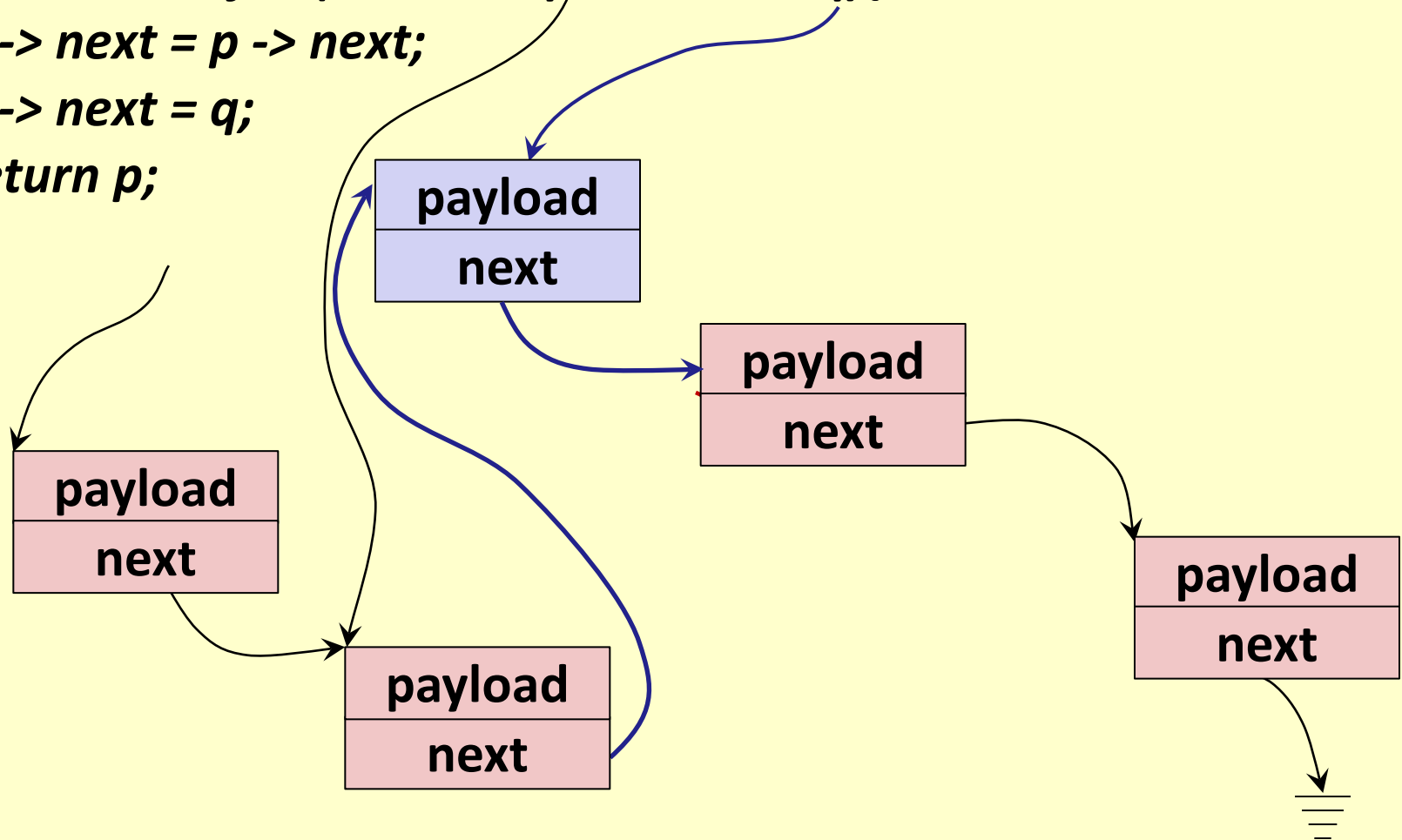
```
listItem *addAfter(listItem *p, listItem *q){  
    q -> next = p -> next;  
    p -> next = q;  
    return p;  
}
```



Question: What to do if we cannot guarantee that p and q are non-NULL?

Adding an Item to a List

```
listItem *addAfter(listItem *p, listItem *q){
    q -> next = p -> next;
    p -> next = q;
    return p;
}
```



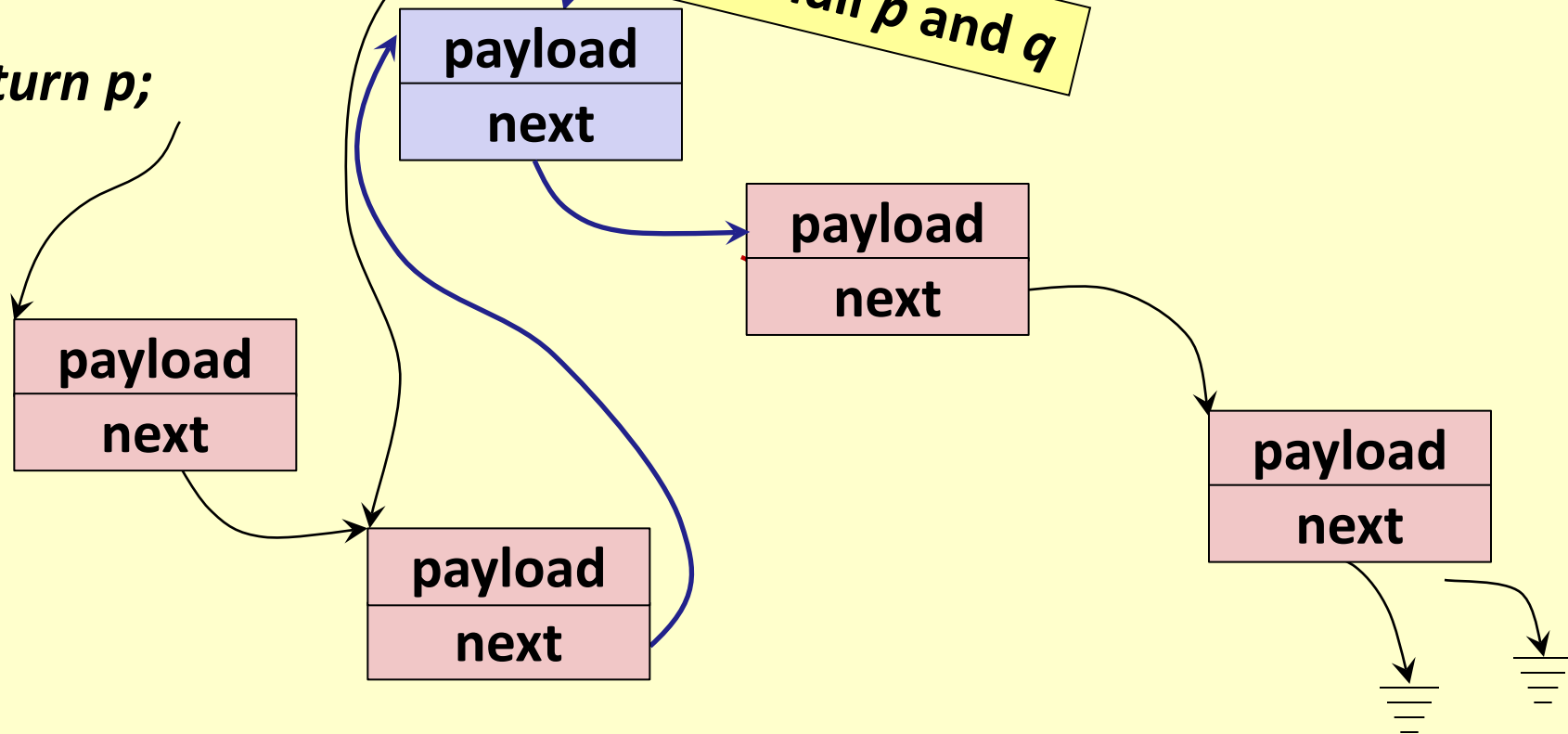
Adding an Item to a List

```

listItem *addAfter(listItem *p, listItem *q){
    if (p && q) {
        q -> next = p -> next;
        p -> next = q;
    }
    return p;
}

```

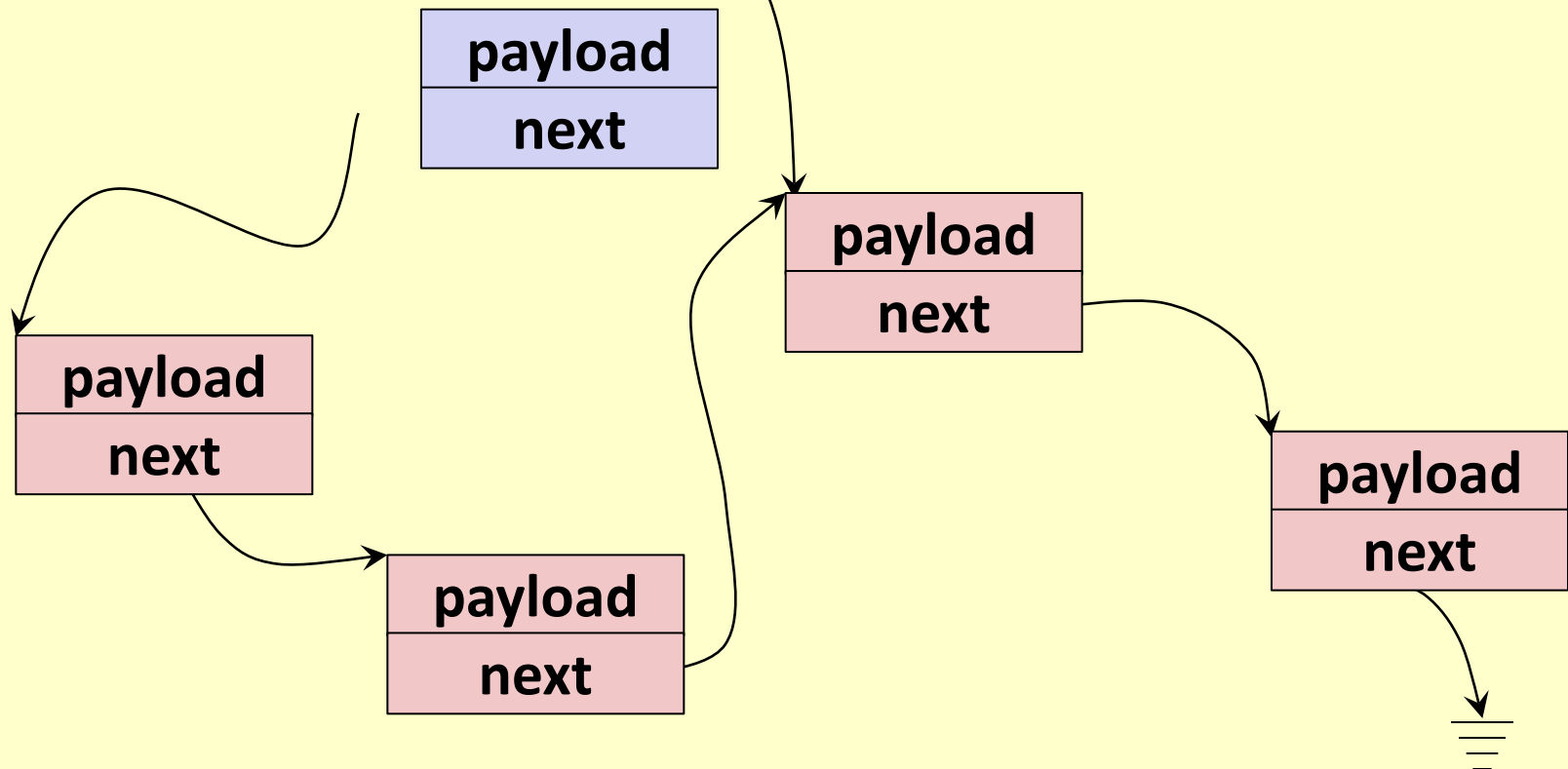
Note test for non-null p and q



What about Adding an Item before another Item?

*struct listItem *p;*

- Add an item *before* item pointed to by *p* (*p* $\neq$ *NULL*)



What about Adding an Item before another Item?

■ Answer:—

- Need to search list from beginning to find previous item
- Add new item after previous item

Doubly-Linked List

```

struct listItem {
    type payload;
    listItem *prev;
    listItem *next;
};

struct listItem *head, *tail;

```

In-class exercise:— how to
add a new item *q* after a list
item *p*

CS-2303, A-Term 2010

Linked Lists in C and C++

15

Other Kinds of List Structures

■ **Queue — FIFO** (First In, First Out)

- Items added at *end*
- Items removed from *beginning*

■ **Stack — LIFO** (Last In, First Out)

- Items added at *beginning*, removed from *beginning*

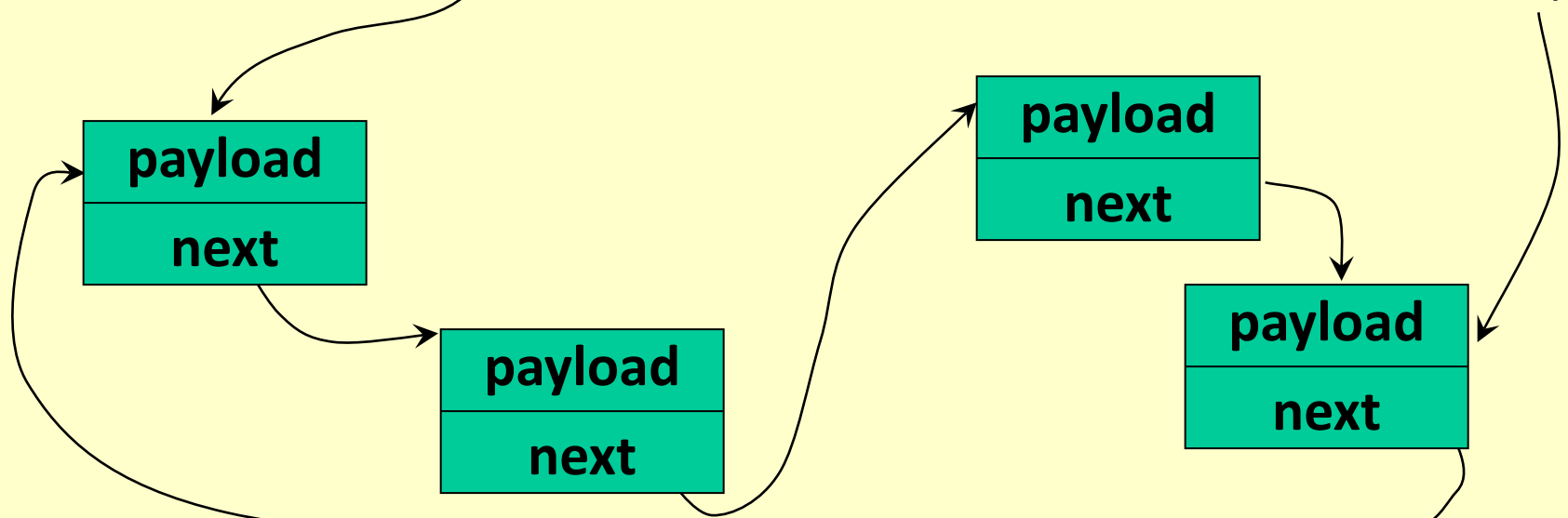
■ **Circular list**

- Last item points to first item
- Head may point to first or last item
- Items added to *end*, removed from *beginning*

Circular List

Optional:–
`struct listItem *head;`

`struct listItem *tail;`



```

listItem *addToTail (listItem *p, listItem *tail){
    if (p && tail) {
        p -> next = tail -> next;
        tail = tail -> next = p;
    } else if (p) {
        tail = p -> next = p;
    }
    return tail;
}
  
```

Questions?