

Over the Thanksgiving weekend, many sales were advertised:
(35% off, 40% off, 50% off)

This got me thinking:

- If one orders long pants, would one get shorts?
- If one orders size 30, would one get size 15?
- If one orders a flowering plant, would one get just the stems?
- If one were to order a three-piece suit, would one get one or two pieces (and which would be included)?
- Best not to think about what would be received, if one ordered a two-piece garment!

Pushing this a bit further, suppose one ordered size XL (Roman numerals for 40). What would one get?

- Just one of the two letters
 - size X (Roman numerals for 10)
 - size L (Roman numerals for 50)
- size XX (Roman numerals for 20)

Announcements

Office Hours this week

Today: Canceled

Tuesday: 2:30-4:00 (adjustments may be needed)

Wednesday: 1:00—3:00

Lab partners: We are in the middle of stacks and queues,
New Partners Wednesday, but announced today

Guest instructor Wednesday (maybe Friday also)

Important to come to Wednesday's class on time and prepared!!!!

All work next week to be submitted to my office, NOT IN CLASS

**Work turned in during class on Wednesday, Friday will be
assigned —10 (negative 10) points and shredded**

(Guest instructor must not be disrupted by assignment logistics!!!)

"Thursday" Extras from Tenure-track Candidates

- Tuesday @ 4:00: William Bares, College of Charleston; intelligent interactive graphical systems
- Thursday @ 4:00: Yucong Jiang, Indiana University; music informatics

Status Reports Distributed Today

- Include all work submitted to date
- Extra credit supplemental problems
 - Scores included on reports
 - Work not returned until exam week (for reasons related to academic honesty)
 - If scores seem surprising, feel free to come see me

Lab partners for Wednesday and Friday

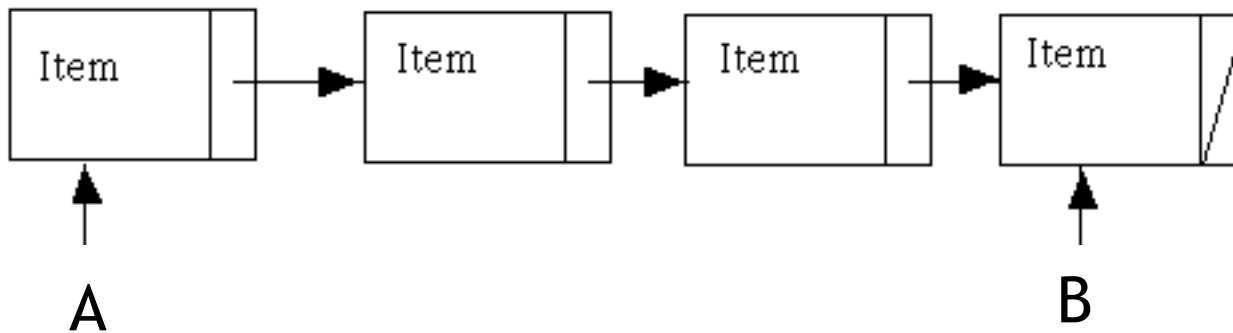
Lab partners for next week

- if you wish to work with a particular partner, both partners should email by noon Sunday
- if I do not hear from you (and your partner), I'll assign partners as in the past (respecting those you do NOT want to work with).

Linked list implementation of queues

Use linked list to contain each element of the queue

- which end for dequeue?
- which end for enqueue?



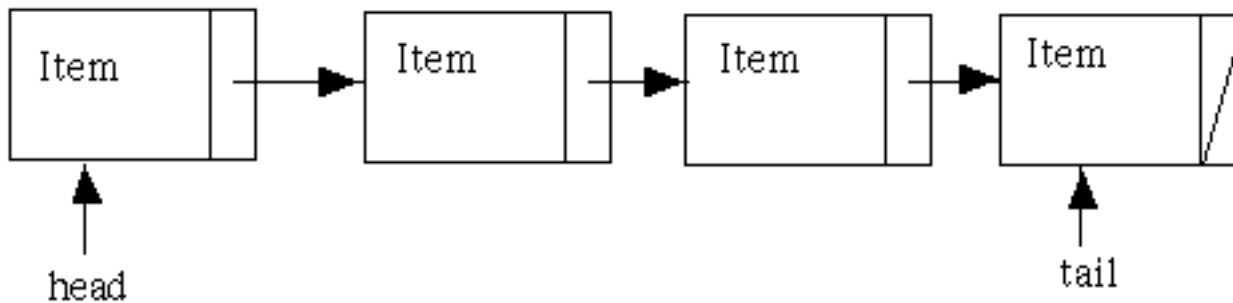
What choices are appropriate for removing an element (e.g., for dequeue)

1. remove at A
 2. remove at B
 3. remove A or B
- A. 1 only B. 2 only C. 3 (1 or 2) D. neither
E. something else

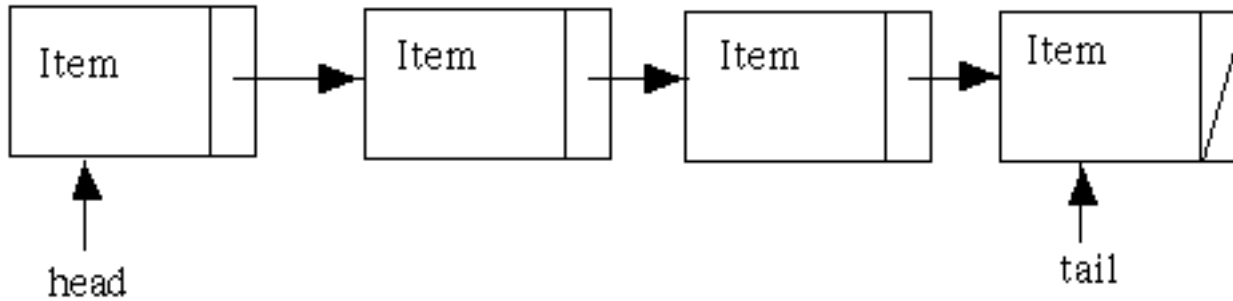
Linked list implementation of queues

Use linked list to contain each element of the queue

- which end for dequeue?
- which end for enqueue?



How to represent a queue data type?



How to represent a queue data type?

- Combine head, tail pointers in struct

```
/* Maximum length of names */
```

```
#define strMax 20
```

```
typedef struct node  
{ char data [strMax];  
  struct node * next;  
} queueNode;
```

```
typedef struct {  
  queueNode * head;  
  queueNode * tail;  
} stringQueue;
```

```
stringQueue queue;
```

Further comments on the Queue user interface
Data type (for storing strings on a queue)

```
typedef . . . {  
    } stringQueue;
```

What is the most appropriate header for isEmpty?

- A. int isEmpty (stringQueue queue)
- B. int isEmpty (stringQueue * queue)

Competing priorities

- Efficiency
- Safety

- A. `int isEmpty (stringQueue queue)`
- B. `int isEmpty (stringQueue * queue)`

Efficiency: Pros

- Just pass 1 pointer—very fast
- Avoid copying large arrays

Efficiency: Cons

- in practice, `stringQueue` object likely to be quite small (e.g., only 2 addresses in list implementation)

```
typedef struct {  
    queueNode * head;  
    queueNode * tail;  
} stringQueue;
```

- if structure large, likely to involve dynamic memory, pointers behind the scenes
- `stringQueue` itself might be a pointer, so `stringQueue *` just adds complexity
- with simple function (e.g., `isEmpty: return queue.count==0;`) optimizing compiler may drop function's code into compiled main and avoid function call entirely.

A. `int isEmpty (stringQueue queue)`

B. `int isEmpty (stringQueue * queue)`

Safety: Pros

- Design follows conceptual view of operation
- Prevent accidental changing of queue, when queue only to be observed

Safety: Cons

- Might copy large structure (although few implementations will utilize large structures directly)

Queue user interface

Data type (for storing strings on a queue)

```
typedef . . . {  
    } stringQueue;
```

Operations:

```
/* initialize queue referenced by the parameter */  
void initializeQueue (stringQueue * queue)  
  
/* return whether the queue is empty */  
int isEmpty (stringQueue queue)  
  
/* return whether the queue is full */  
int isFull (stringQueue queue)  
  
/* insert given item into back of the queue */  
int enqueue (stringQueue * queue, char* item)  
  
/* removes front item from queue referenced by parameter */  
char * dequeue (stringQueue * queue)
```

How to implement isFull?

```
/* return whether the queue is full */  
int isFull (stringQueue queue)
```

At least two approaches possible

Simplistic approach (main memory big, so not likely to run out of space)

```
int isFull (stringQueue queue) {  
    return 0;  
}
```

Better approach approach, check if memory can still be allocated

```
int isFull (stringQueue queue) {  
    stringNode * ptr = malloc (sizeof(stringNode));  
    if (ptr == NULL)  
        return 0; // no space available  
    else {  
        // space available, but free this memory  
        free (ptr);  
        return 1;  
    }  
}
```

Two more details

(first note commonly included in code;
second note often missed)

Enqueue

- end item to tail of queue
- if queue was empty, head also must point to new node

Dequeue

- remove item from head of queue
- if queue becomes empty, set tail to NULL as well.