NANYANG TECHNOLOGICAL UNIVERSITY

CE1107/CZ1107: DATA STRUCTURES AND ALGORITHMS

5B: **Queues**

College of Engineering
School of Computer Engineering

TODAY

- Motivating application
- Queue data structure
- Queue implementation using linked lists
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Worked examples: Applications

LEARNING OBJECTIVES

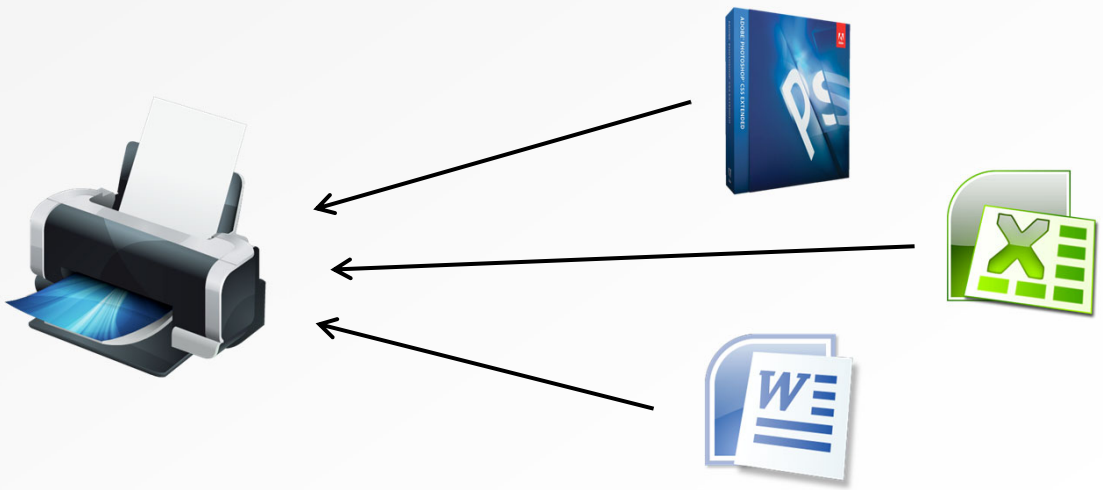
After this lesson, you should be able to:

- Explain how a queue data structure operates
- Implement a queue using a linked list
- Choose a queue data structure when given an appropriate problem to solve

- You should also be able to
  - Implement a queue using an array (but we won't cover or test this)

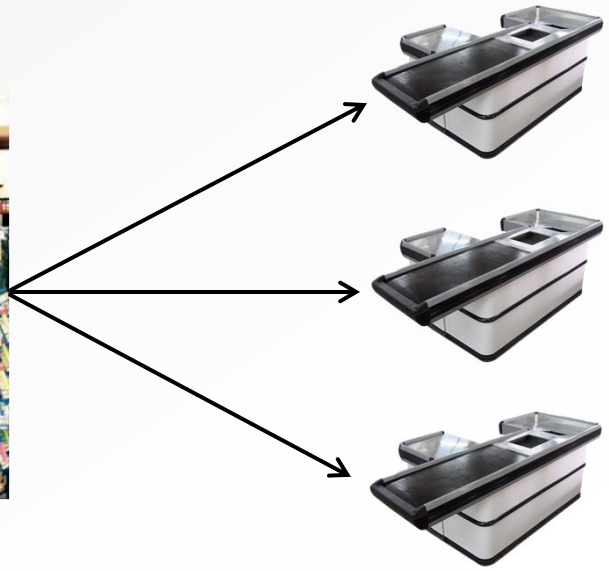
MOTIVATING APPLICATION #1

- Write a printer driver application:
  - Print jobs may be sent to the printer driver at any time
  - A print job must be stored until it can be sent to the printer
  - Print jobs are sent in first-come, first-served order to the printer
  - Print jobs take a long time to complete
    - When a print job completes, the next waiting print job should be sent to the printer



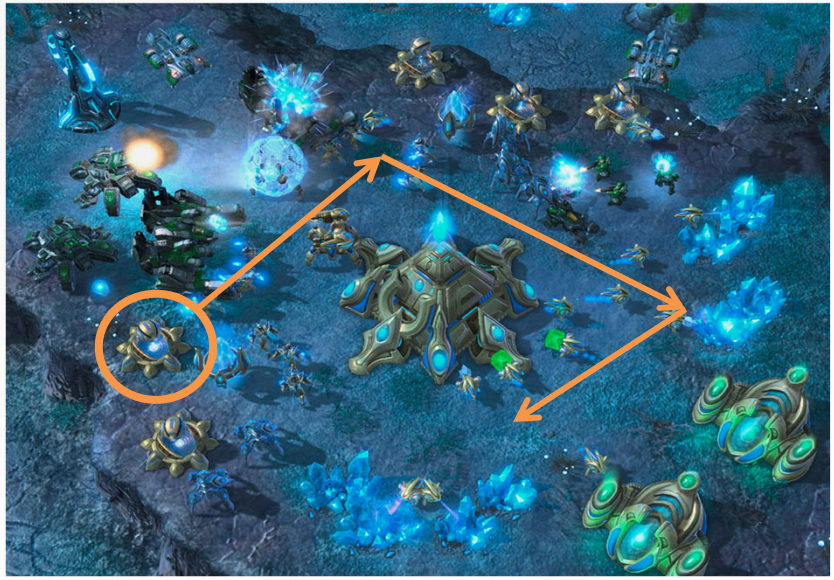
MOTIVATING APPLICATION #2

- Supermarket checkout counter assignment
  - 1 checkout counter OR N checkout counters
  - Single queue of customers
  - First-come, first-served bases
    - Join the back of the queue and wait for your turn



MOTIVATING APPLICATION #3

- Sequence of commands for a unit in a game
- Commands may be added to the sequence at any time
- Must be carried out in this order
  - Move there
  - Attack
  - Move there
  - Etc..
  - Self-destruct



TODAY

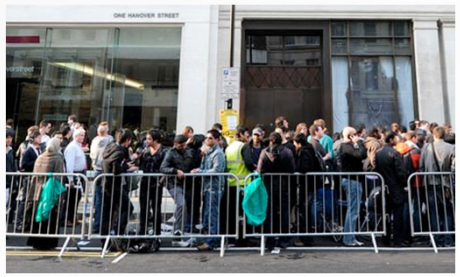
- Motivating application
- **Queue data structure**
- Queue implementation using linked lists
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Worked examples: Applications

PREVIOUSLY

- Arrays
  - Random access data structure
- Linked lists
  - Sequential access data structure
- Limited-access sequential data structures
  - Stack
    - Last In, First Out (LIFO)
- Today, another limited-access sequential data structure

QUEUE DATA STRUCTURE

- A **Queue** is a data structure that operates like a real-world queue
  - Queue to use an ATM or buy food, for example
  - Elements can only be added at the back
  - Elements can only be removed from the front
- Key: First-In, First-Out (FIFO) principle
  - Or, Last-In, Last-Out (LILO)
- As with stacks, often built on top of some other data structure
  - Arrays, Linked lists, etc.
  - We'll focus on a linked-list based implementation again




QUEUE DATA STRUCTURE

- **Core operations**
  - Enqueue: Add an item to the back of the queue
  - Dequeue: Remove an item from the front of the queue
- **Common helpful operations**
  - Peek: Inspect the item at the front of the queue without removing it
  - IsEmptyStack: Check if the queue has no more items remaining
- **Corresponding funtions**
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- We'll build a queue assuming that it only deals with integers
  - But as with linked lists and stacks, can deal with any contents depending on your code

TODAY

- Motivating application
- Queue data structure
- **Queue implementation using linked lists**
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Worked examples: Applications

QUEUE IMPLEMENTATION USING LINKED LISTS

- Recall that we defined a LinkedList structure
- Next, we define a Stack structure
- Now, define a Queue structure
  - We'll build our queue on top of a linked list

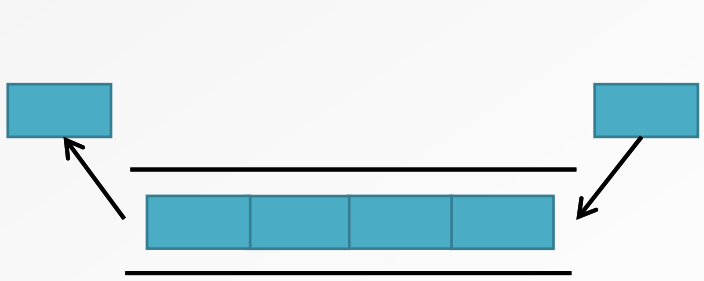
```
typedef struct _queue{
     LinkedList ll;
} Queue;
```

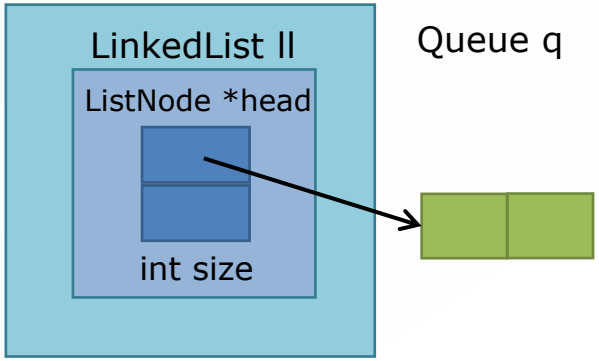
QUEUE IMPLEMENTATION USING LINKED LISTS

- Queue structure

```
typedef struct _queue{
     LinkedList ll;
} Stack;
```

- Again, wrap up a linked list and use it for the actual data storage
- Notice that the LinkedList already takes care of little things like keeping track of # of nodes, etc.
- There is one modification we need for a queue… KIV



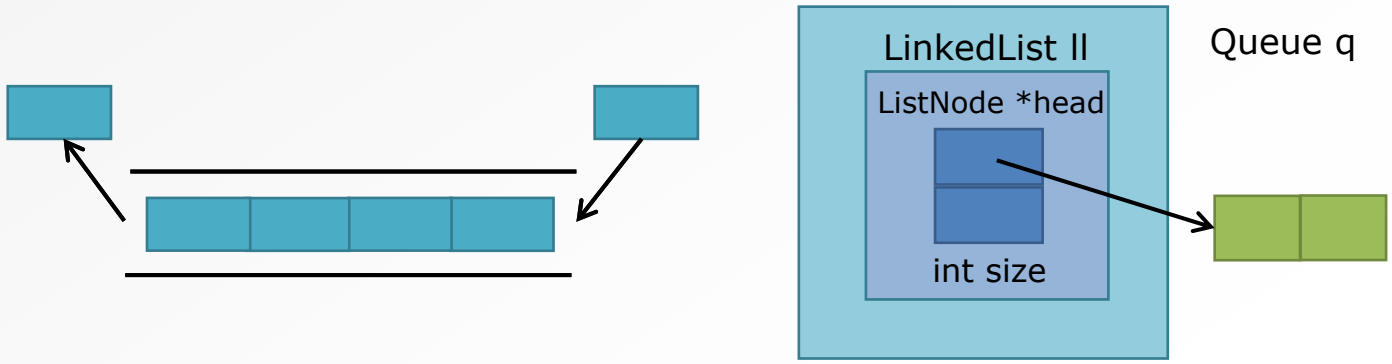


TODAY

- Motivating application
- Queue data structure
- Queue implementation using linked lists
- **Queue functions**
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Worked examples: Applications

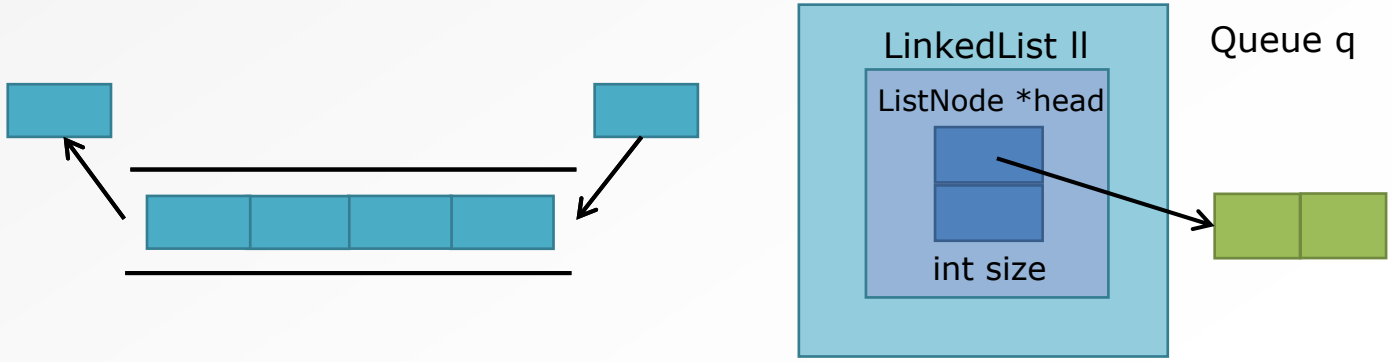
QUEUE FUNCTIONS: enqueue()

- enqueue() function is the only way to add an element to the queue data structure
- Only allowed to enqueue() at the end
- Question:
  - Using a linked list as the underlying data storage, does the first linked list node represent the front or the back of the queue? front
  - Figure out which option makes it easier to implement enqueue() and dequeue()



QUEUE FUNCTIONS: enqueue()

- **Hands-on: Write the enqueue() function**
  - Define the function prototype
  - Implement the function
  - Very similar to what we did for stack: push()
- Answer is a few slides down, so don't look yet
- Requirements
  - Make use of the LinkedList functions we've already defined
  - Insert at the back only (what index position?)



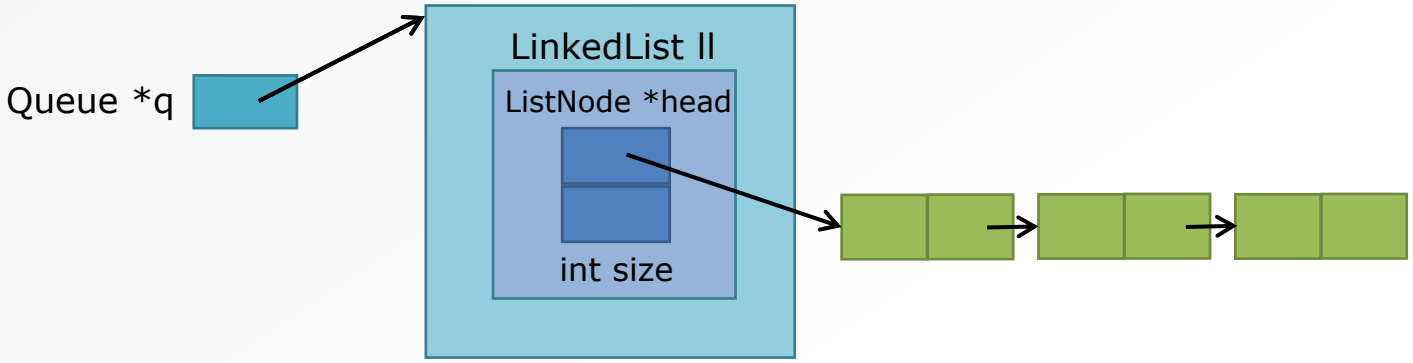
QUEUE FUNCTIONS: enqueue()

- **Hands-on: Write the enqueue() function**
  - Before looking at the code on the next slide, try writing the code for yourself

```
void enqueue(Queue *q, int item ){

    InsertNode(&(q->ll), q->ll.size, item);

}
```



QUEUE FUNCTIONS: enqueue()

- First linked list node corresponds to the front of the queue
- Last linked list node corresponds to the back of the queue
- Enqueueing a new item → adding a new node to the end of the linked list

```
void enqueue(Queue *q, int item){
    insertNode(&(q->ll), q->ll.size, item);
}
```

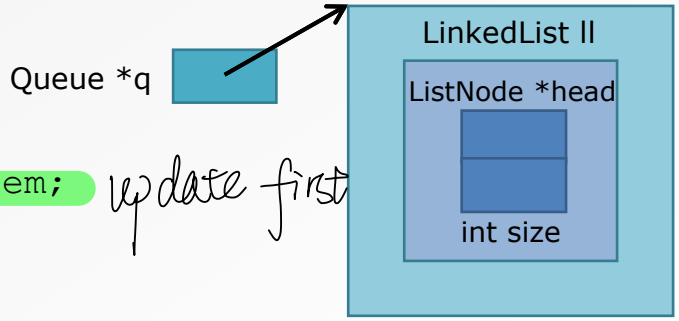
  - Notice that this could be a very inefficient operation if the queue is long
- Need to use a tail pointer to make the operation efficient
  - Gives us direct access to the current last node of the linked list
- Also note that the inefficient version still works

QUEUE FUNCTIONS: dequeue()

- Dequeueing a value is a two-step process again
  - Get the value of the node at the front of the linked list
  - Remove that node from the linked list

```
int dequeue(Queue *q){
    int item;
    item = ((q->ll).head)->item;
    removeNode(&ll, 0);
    return item;
}
```

then remove



- Need a temporary int variable to hold the stored value because we can't get it after we remove the front node

QUEUE FUNCTIONS: peek()

- No change in logic from the stack version
- Peek at the value at the front of the queue
  - Get the value of the node at the front of the linked list
    - Without removing the node

```
int peek(Queue *q){
    return ((q->ll).head)->item;
}
```

QUEUE FUNCTIONS: isEmptyQueue()

- Again, exactly the same logic as isEmptyStack()
- Check to see if # of nodes == 0
- Make use of the built-in size variable in the LinkedList struct

```
int isEmptyQueue(Queue *q){
    if ((q->ll).size == 0) return 1;
    return 0;
}
```

TODAY

- Motivating application
- Queue data structure
- Queue implementation using linked lists
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- **Worked examples: Applications**

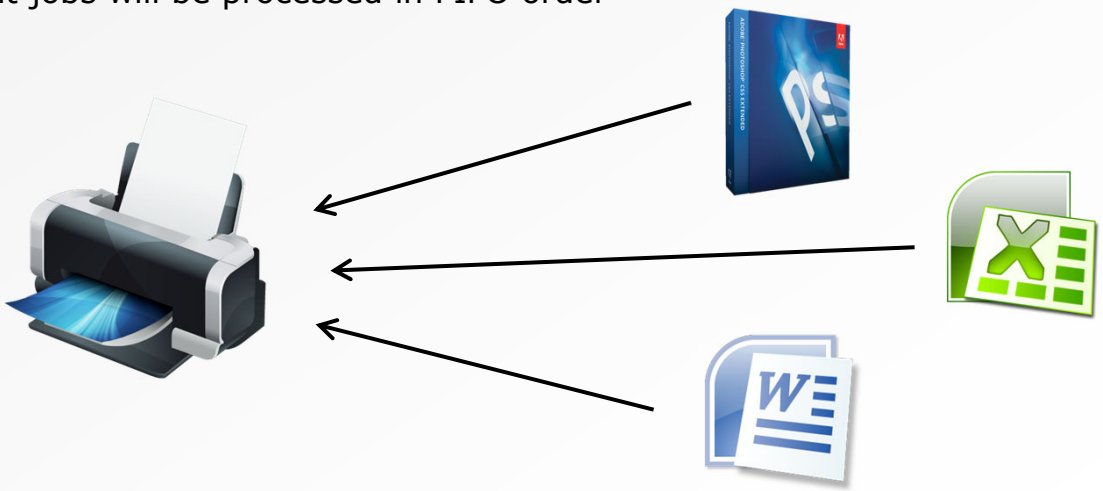
SIMPLE TEST APPLICATION

- Simple application
  - Enqueue some integers
  - Dequeue and print

```
1   int main(){
2       Queue q;
3       q.ll.head = NULL;
4       q.ll.tail = NULL;
5
6       enqueue(&q, 1);
7       enqueue(&q, 2);
8       enqueue(&q, 3);
9       enqueue(&q, 4);
10      enqueue(&q, 5);
11      enqueue(&q, 6);
12
13      while (!isEmptyQueue(&q))
14          printf("%d ", dequeue(&q));
15  }
```

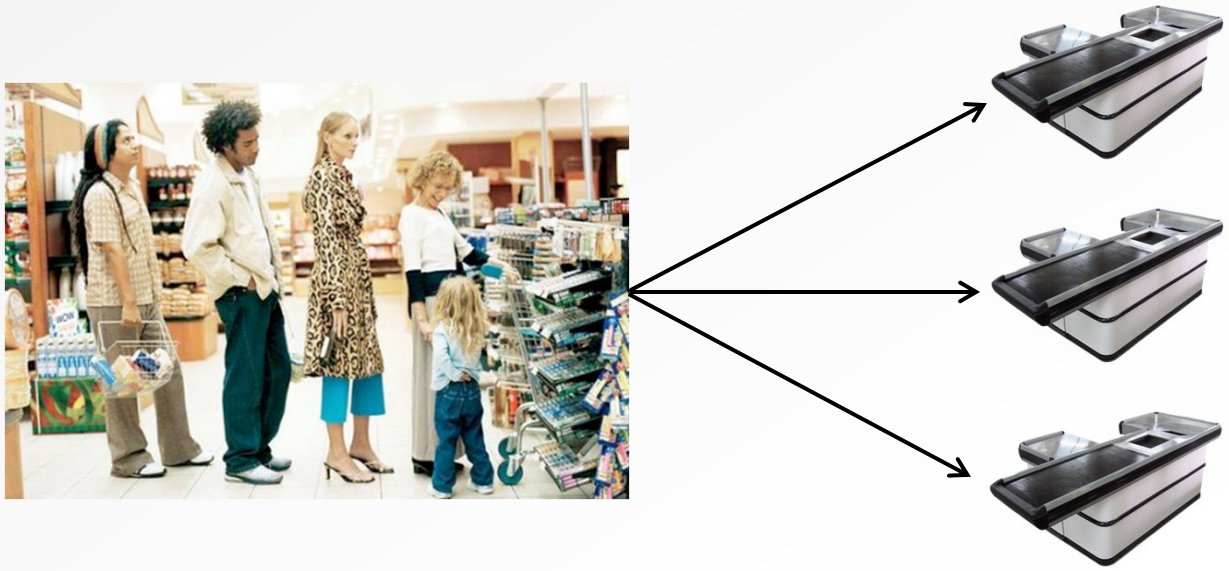
MOTIVATING APPLICATION #1

- Application sends print job to driver by calling addPrintJob()
  - This will enqueue() the print job
- When printer finishes the current print job, it calls getNextPrintJob()
  - This will dequeue() from the queue
- Neither the application nor the printer has to care about other waiting print jobs, etc.
- All print jobs will be processed in FIFO order



MOTIVATING APPLICATION #2

- To checkout, join the queue at the back
- When any of the checkout counters becomes available, it calls getNextCustomer()
- First-come, first-served order of processing guaranteed
- Checkout counters don't have to care about all other waiting customers



TODAY

- Motivating application
- Queue data structure
- Queue implementation using linked lists
- Queue functions
  - enqueue()
  - dequeue()
  - peek()
  - isEmptyQueue()
- Worked examples: Applications