# CMPT307: Elementary Data Structures

Week 1-3

**Xian Qiu**

Simon Fraser University

xianq@sfu.ca

# Outline

- linked lists
- stacks
- queues

# Dictionary Operations

- a dictionary has data items $a_i$, indexed by key $i$ for $i \in \mathbb{N}$
- assume key values are unique

## Basic dictionary operations

- printing data w.r.t. sorted keys
- search
- insertion
- deletion

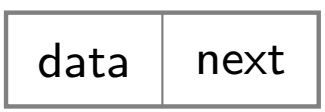
# Lists

arrays

- ▷ advantages: simple and fast
- ▷ disadvantages
  - ∗ must specify size $n$ at construction
  - ∗ $n$ is often unknown and dynamic

linked list: each node contains

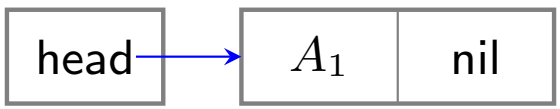
- ▷ the data item
- ▷ a pointer to the next node




```
struct node {
    int key;
    int val;
    struct node *next;
};
```

# Linked Lists

list $L$ with a head: initially nil

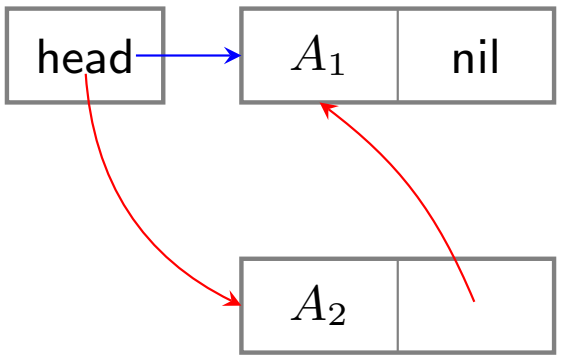



insert the first node

- ▷ set next to nil
- ▷ set head to point to the new node

# Insertion

insert the second node



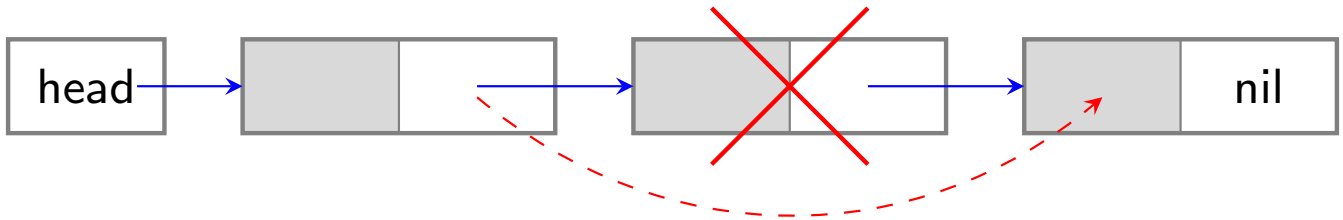

LIST-INSERT$(L, x)$

```
1 if L.head ≠ nil then
2     x.next = L.head;
3 L.head = x;
```

can we add $x$ to the tail?

# Deletion

remove an element $x$ from $L$




LIST-DELETE$(L, x)$

```
1 prev = L.head;
2 if prev == x then
3     L.head = x.next ;                          // x is the head of L
4 else
5     while prev.next ≠ x do
6         prev = prev.next;
7     prev.next = x.next;
```

# Searching

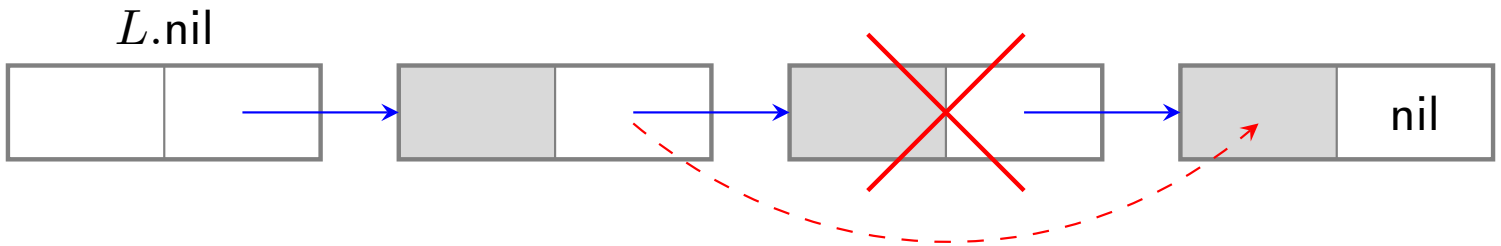
find the first element with key $k$ in list $L$

LIST-SEARCH$(L, k)$

```
1 x = L.head;
2 while x ≠ nil and x.key ≠ k do
3     x = x.next;
4 return x;
```

# Sentinels

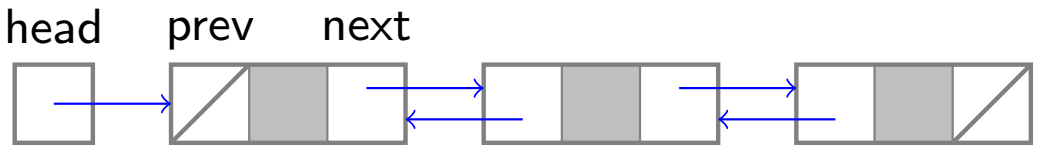
- a sentinel is a dummy object, say L.nil
- it allows us to simplify boundary conditions



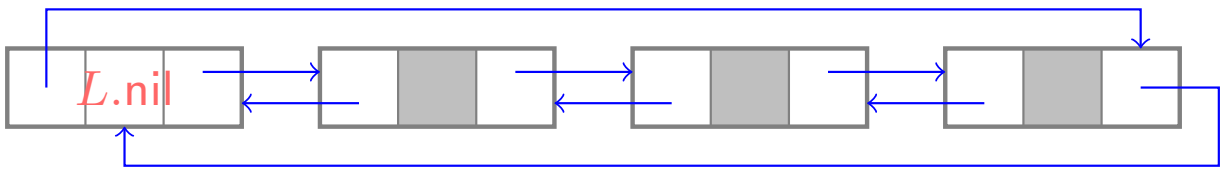

LIST-DELETE(L,x)

```
1 prev = L.nil ;
2 while prev.next ≠ x do
3     prev = prev.next;
4 prev.next = x.next;
```

# Doubly Linked Lists

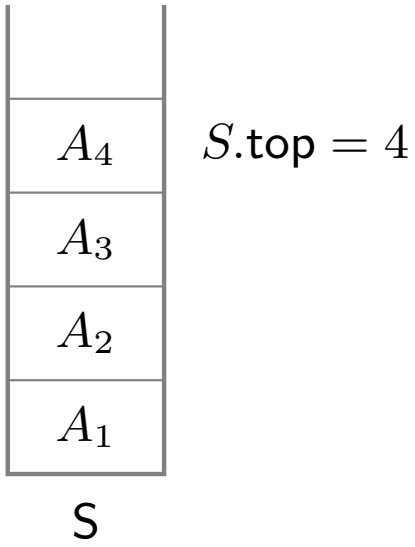



use sentinel: referring $L$.nil as nil yields circular, doubly linked list



# Stacks

- ▷ LIFO: last-in, first-out
- ▷ $S$.top $=$ number of items in $S$
- ▷ operations: push and pop
- ▷ implemented by arrays (or linked-lists)



# Methods

STACK-EMPTY($S$)

```
1 If S.top == 0 then return true;
2 else return false;
```

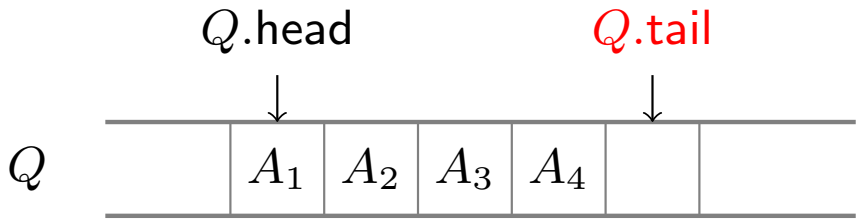
STACK-PUSH($S, x$)

```
1 S.top = S.top + 1;
2 S[S.top] = x;
```

STACK-POP($S$)

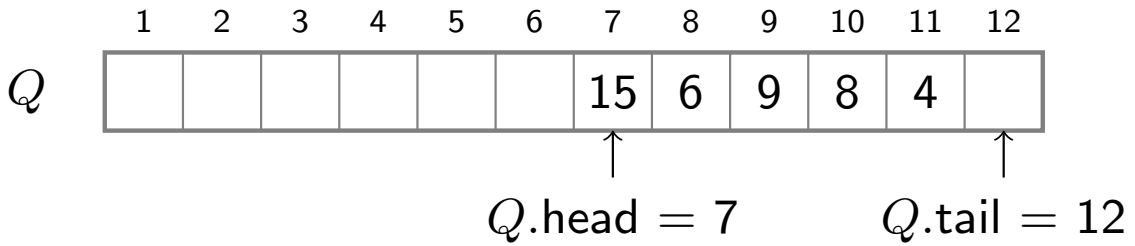
```
1 if STACK-EMPTY(S) then
2     error "underflow";
3 else
4     S.top = S.top − 1;
5 return S[S.top + 1];
```

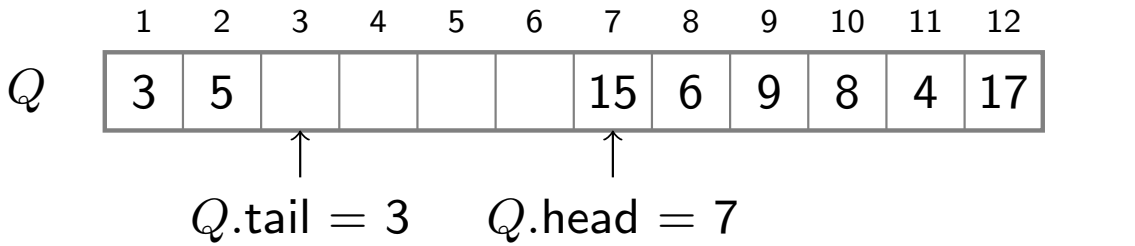
# Queues

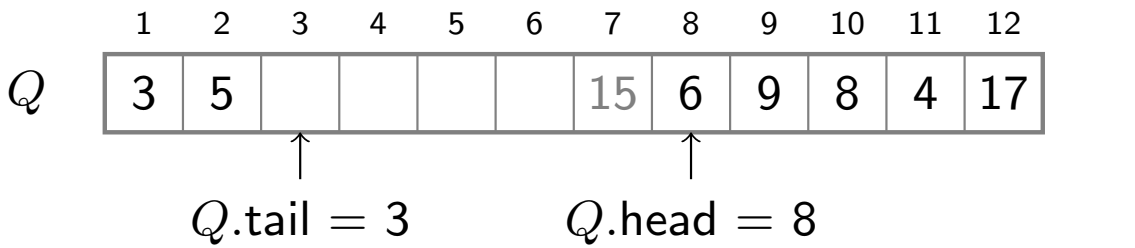



- FIFO: first in, first out
- $Q$.head, $Q$.tail are keys
- operations: enqueue and dequeue
- implemented by arrays (or linked lists)

# Example




note: $Q$ has at most $n - 1$ items (for array size $n$) why?

# Methods

## ENQUEUE($Q, x$)

```
1 Q[Q.tail] = x;
2 if Q.tail == Q.length then
3     Q.tail = 1;
4 else
5     Q.tail = Q.tail + 1;
```

## DEQUEUE($Q$)

```
1 x = Q[Q.head];
2 if Q.head == Q.length then
3     Q.head = 1;
4 else
5     Q.head = Q.head + 1;
6 return x;
```