CSE 373
Data Structures & Algorithms

Lecture 02

Lists, Stacks, and Queues

Announcements

Guest Lecturers next week:

- Monday: Vibhor Rastogi
- Wednesday: Sean Shih-Yen Liu

(I will be giving a keynote lecture at a conference in Brazil, about my research on probabilistic databases)

Today's Outline

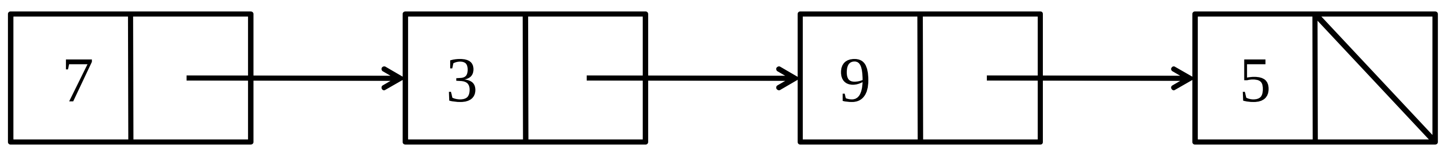
- Lists
- Stacks and queues
- Begin trees

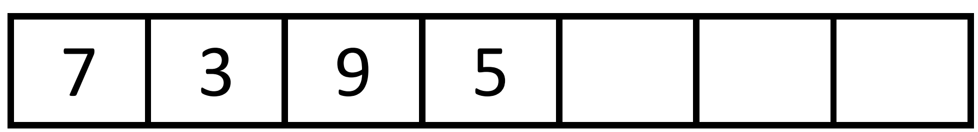
Remember:

- Reading assignment for Monday: Ch. 1- 3
- For today: 3.1, 3.2, 3.3 (did you read them ?)

The List ADT

- List of items, e.g. 7, 3, 9, 5
- Insert, delete, find, replace, …
- Implementations: linked list or array list



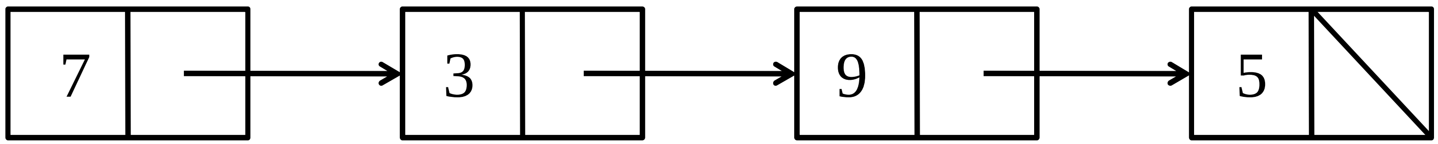


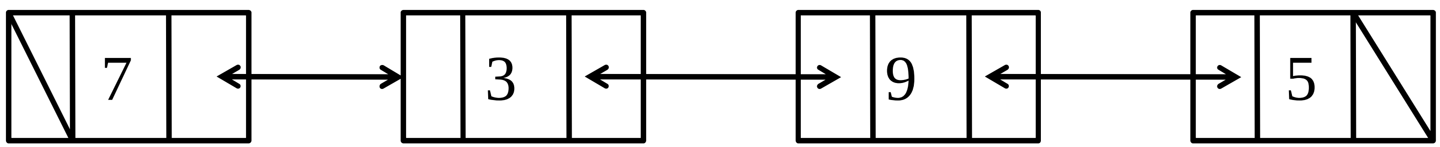
Linked List Implementation

First, need to define a single node:

```
private static class Node<AnyType>
{
 public Node( AnyType d, Node<AnyType> n )
   {
     data = d;  next = n;
   }

    public AnyType data;
    public Node<AnyType>   next;
}
```

Simple Linked v.s. Double Linked





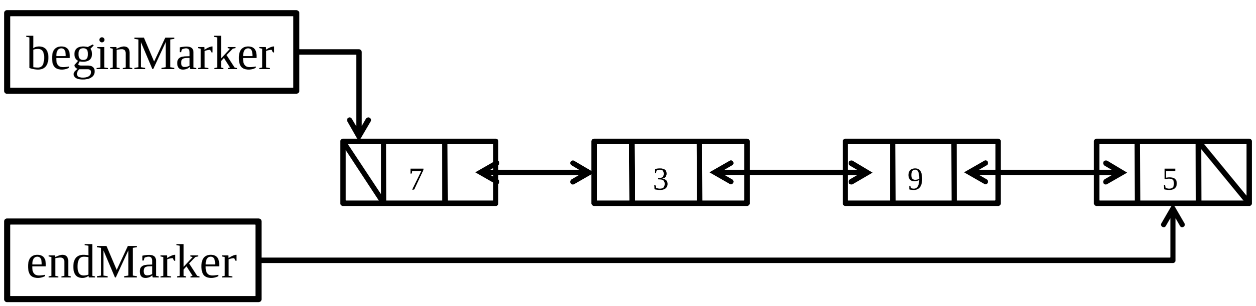
What are the tradeoffs ?

Double Linked List Implementation

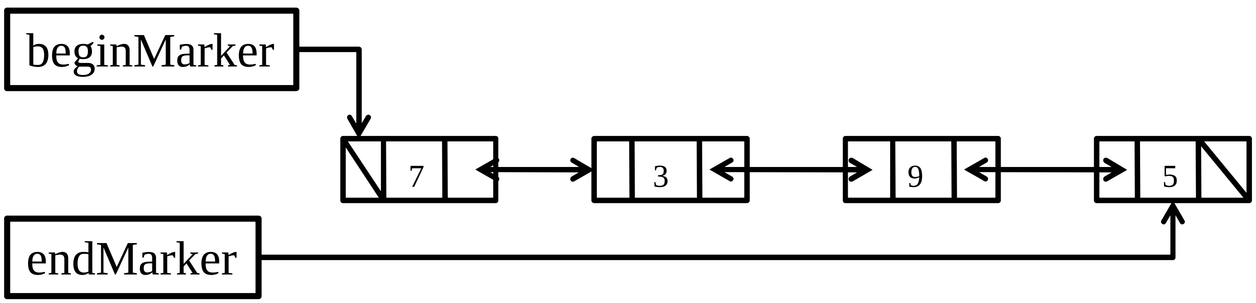
```
private static class Node<AnyType>
{
 public Node( AnyType d, Node<AnyType> p, Node<AnyType> n )
   {
     data = d; prev = p; next = n;
   }

    public AnyType data;
    public Node<AnyType>   prev;   /* now we can go back */
    public Node<AnyType>   next;
}
```

Linked List Implementation

```
public class MyLinkedList<AnyType> implements Iterable<AnyType>
{

  /* all methods go here (see book, code examples) */


    private int theSize;
    private Node<AnyType> beginMarker;
    private Node<AnyType> endMarker;
}
```



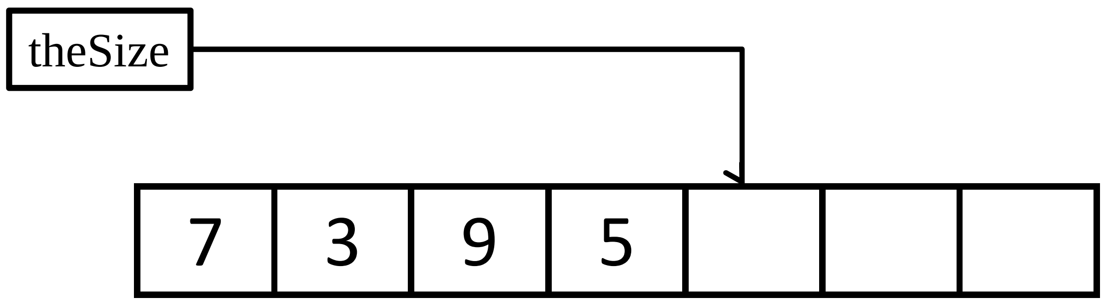
Operations on the Linked List



Discuss in class the following operations:

1. Find(9); Find(2);
2. Insert(11); at the beginning, at the end, in the middle;
3. Remove(9);

Array Implementation



Discuss in class the following operations:

1. Find(9); Find(2);
2. Insert(11); at the beginning, at the end, in the middle;
3. Remove(9);

Review: Big O Notation

DEFINITION: The Big-O notation

$T(n) = O(f(n))$ if there exist constants c and n' such that: $T(n) \le c\, f(n)$ for all $n \ge n'$

Summary: Some Complexities

	Linked list	Array
AnyType get(int idx)	?	?
int find(AnyType x)	?	?
void add(x,idx) /* before/after) */	?	?
void remove(int idx)	?	?
InsertAnywhere(x)	?	?

Summary: Some Complexities

	Linked list	Array
AnyType get(int idx)	O(k) where k=idx	O(1)
int find(AnyType x)	O(n)	O(n)
void add(x,idx) /* before/after) */	O(1)	O(n)
void remove(int idx)	O(1)	O(n)
InsertAnywhere(x)	O(1)	O(1)

Iterators

- Review questions from Weiss, Chapt. 3.3

- What is an *Iterator ?*

- What is an *Iterable ?*

- What is a *Collection ?*

Answers…

```
public interface Iterator<AnyType>
{ boolean hasNext( );
  AnyType next( );
  void remove( );
}
```

An `Iterable` provides a method named `iterator` that returns and object of type `Iterator`. It can be used in a for loop (see book…).

```
public interface Collection<AnyType>
  extends Iterable<AnyType>
{ . . .
  boolean add( AnyType x);
  boolean remove( AnyType x );
   . . .
}
```

Application: Polynomial ADT

Attempt 1:

- linked list implementation:
- A_i is the coefficient of the x^{i-1} term:

$5 + 2x + 3x^2$	**(5 2 3)**
$7 + 8x$	**(7 8)**
$3 + x^2$	**(3 0 1)**

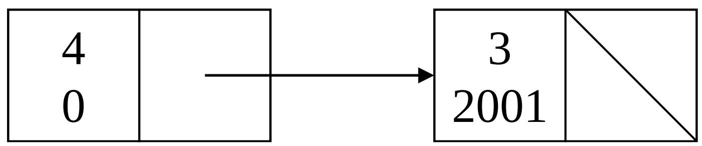
Problem?

$4 + 3x^{2001}$

```
( 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 )
```

Sparse Vector Data Structure: $4 + 3x^{2001}$

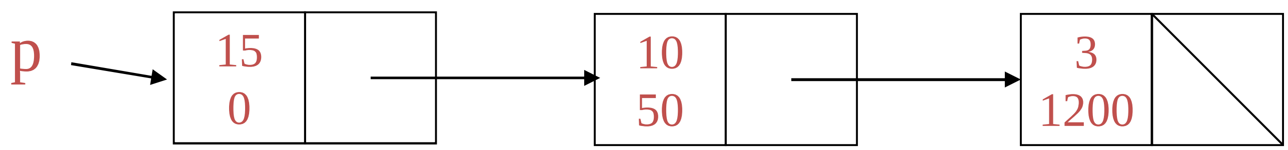
`(<4 0>  <3 2001>)`



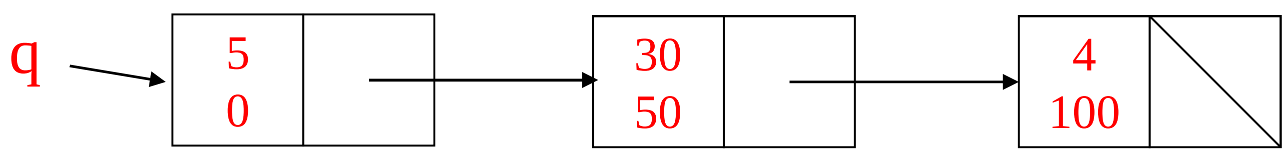
Addition of Two Polynomials?

Complexity?

$15+10x^{50}+3x^{1200}$

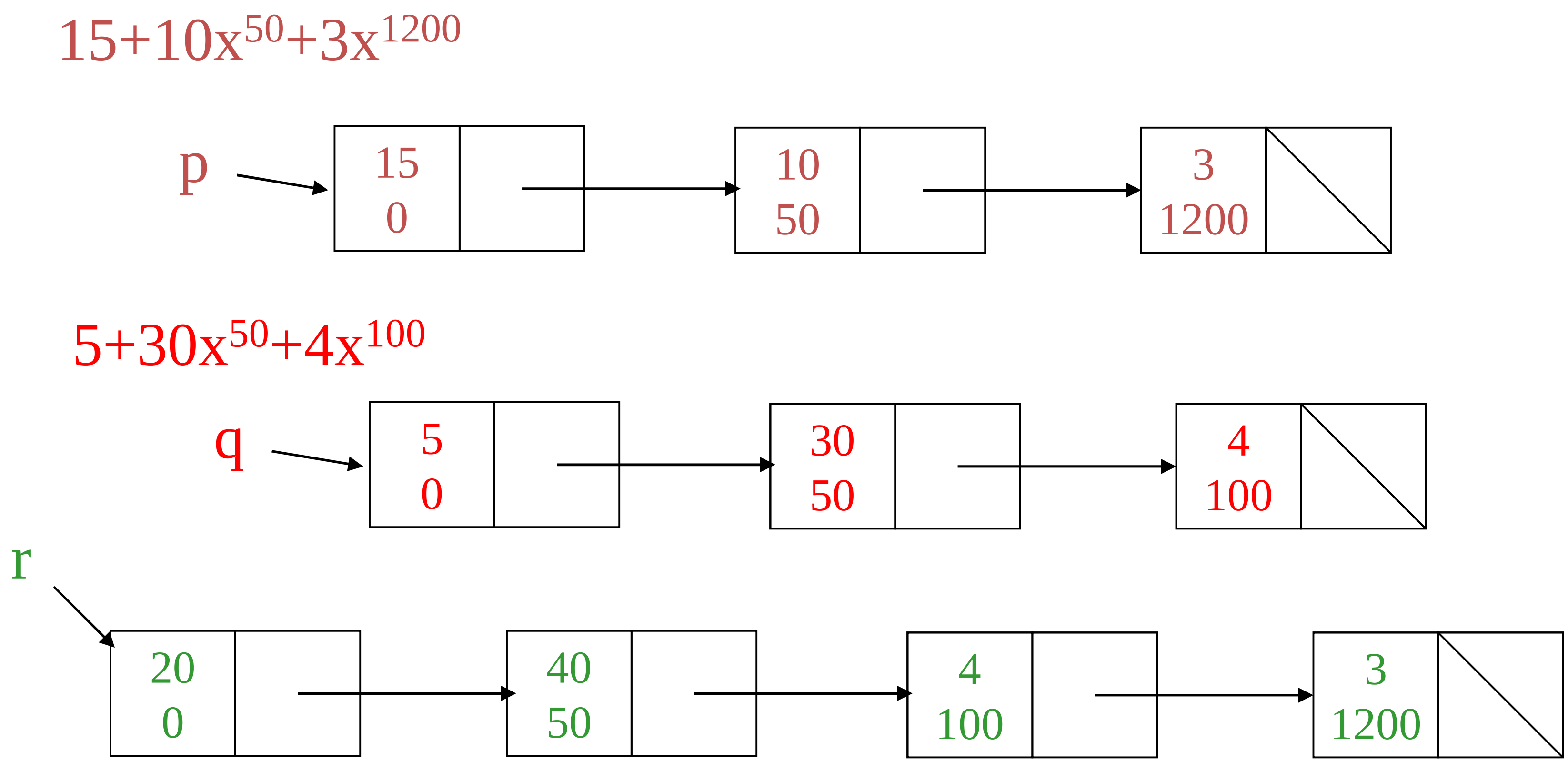


$5+30x^{50}+4x^{100}$



Addition of Two Polynomials

One pass down each list: O(m+n)



Sparse Matrices

18	0	33	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	99	0	0
0	0	0	0	0	0
0	0	0	0	0	27

```
(<row  (<column data> <column data> …)>
 <row  (<column data> <column data> …)>
 … )
```

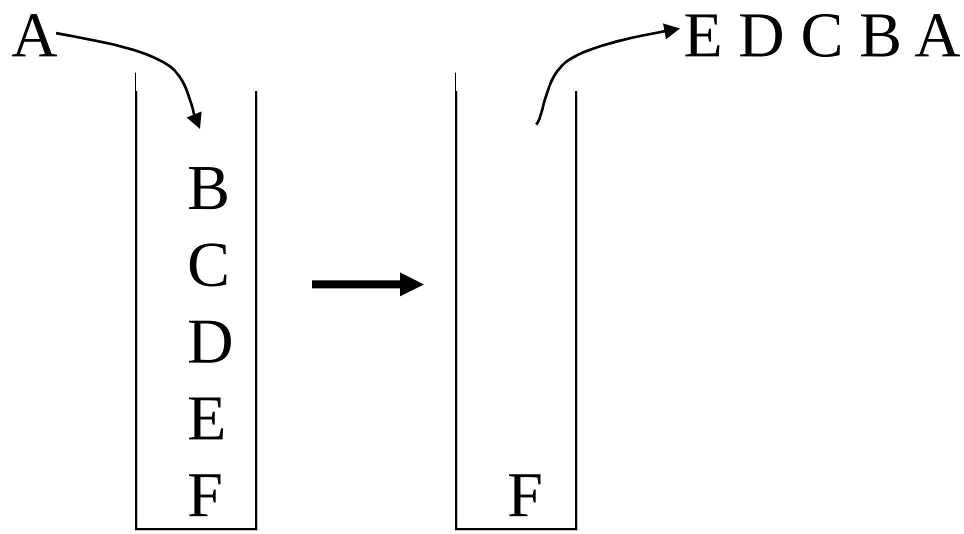
```
(  <1 (<1 18>  <3 33>)>, <4 (<4 99>)>   <6 (<6 27>))
```

```
(<row column data> <row column data> …)
```

```
( <1, 1, 18>  <1, 3, 33> <4, 4, 99>  <6, 6, 27>)
```

First Example: Stack ADT

- LIFO: Last In First Out
- Stack operations
  - create
  - destroy
  - push
  - pop
  - top
  - is_empty



Stacks in Practice

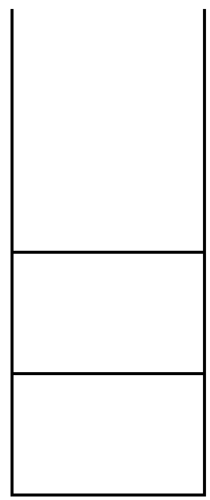
- Function call stack
- Removing recursion
- Balancing symbols (parentheses)
- Evaluating Postfix Notation

  Infix: 7 + 8 * 4 – 30

  Postfix: 7 8 4 * + 30 -

Infix to Postfix with a Stack

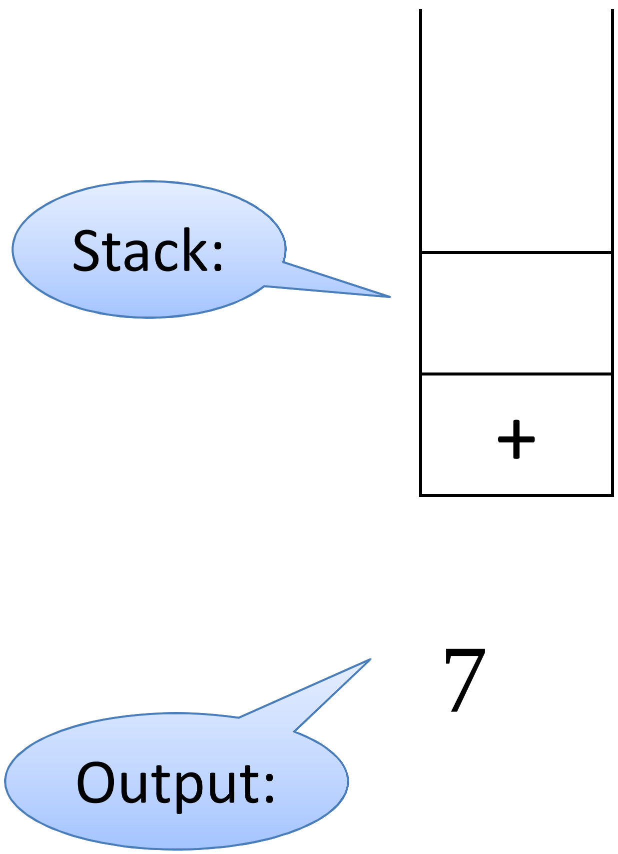
7 + 8 * 4 – 30 #



7

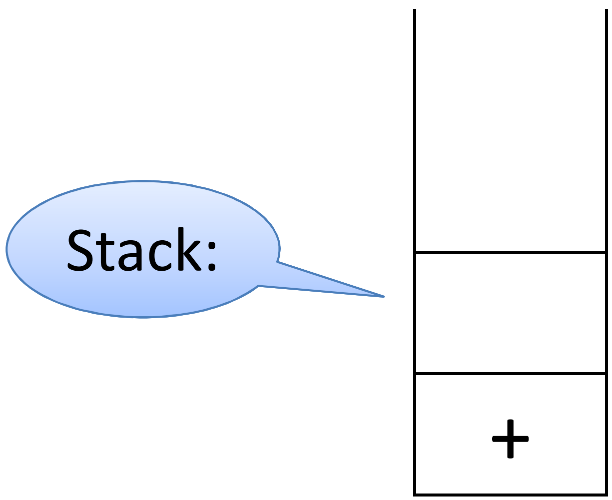
Infix to Postfix with a Stack

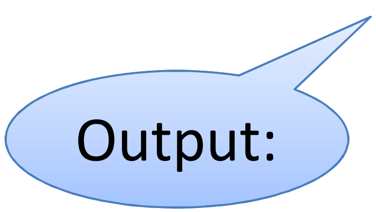
7 + 8 * 4 – 30 #



Infix to Postfix with a Stack

7 + 8 * 4 – 30 #

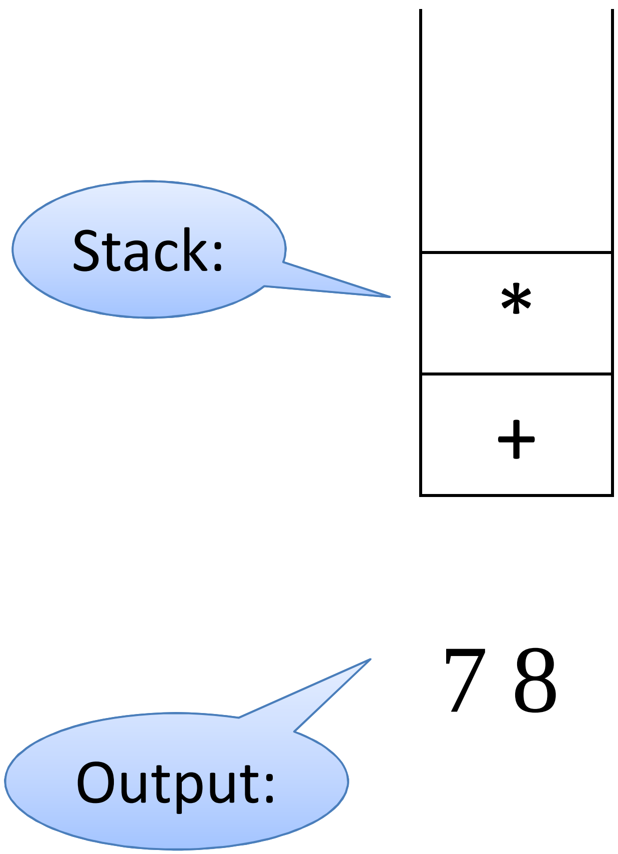




7 8

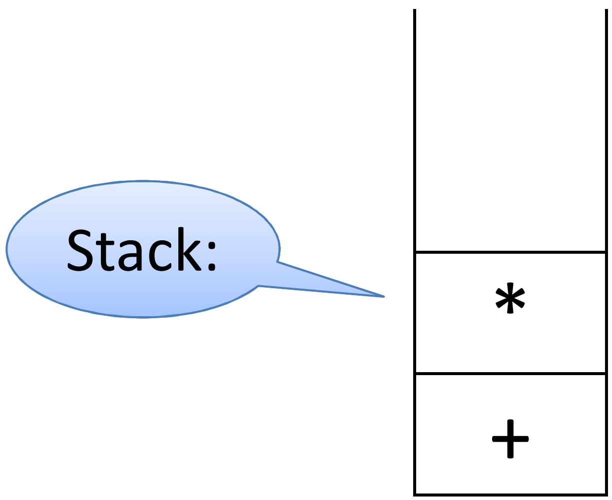
Infix to Postfix with a Stack

7 + 8 * 4 – 30 #



Infix to Postfix with a Stack

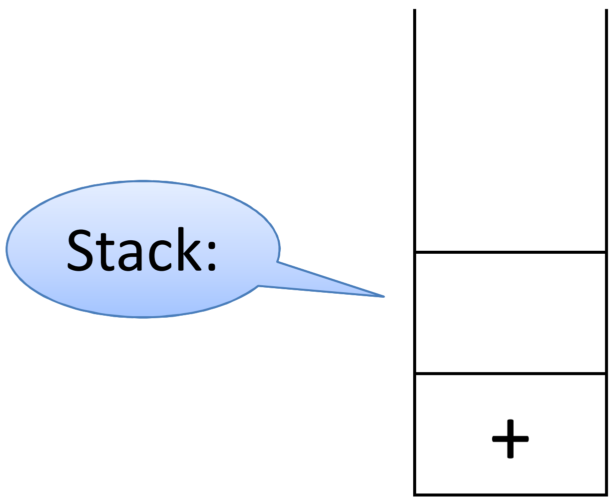
7 + 8 * 4 – 30 #



Output: 7 8 4

Infix to Postfix with a Stack

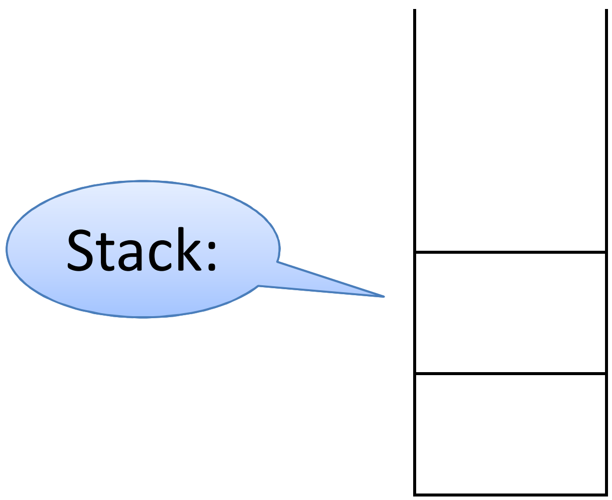
7 + 8 * 4 – 30



Output: 7 8 4 *

Infix to Postfix with a Stack

7 + 8 * 4 – 30 #



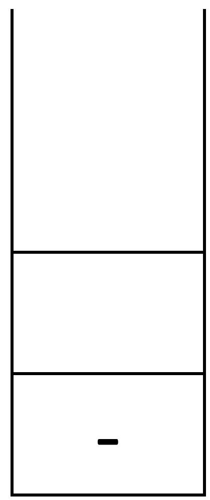
(empty stack)

Output: 7 8 4 * +

Infix to Postfix with a Stack

7 + 8 * 4 – 30 #

Stack:

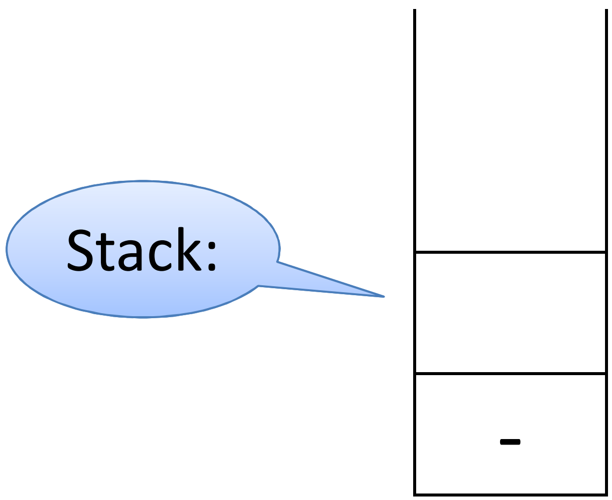


Output:

7 8 4 * +

Infix to Postfix with a Stack

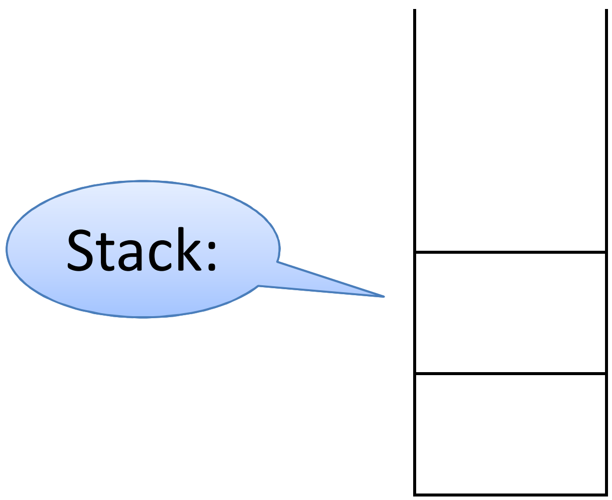
7 + 8 * 4 – 30 #



Output: 7 8 4 * + 30

Infix to Postfix with a Stack

7 + 8 * 4 – 30 #



Output:

7 8 4 * + 30 -

Array vs. Linked List

- Too much space, or not enough space
- Not as complex
- Could make array more robust

- Kth element accessed "easily"
- Insert requires shift

- Can grow as needed
- More memory per item in the queue
- Linked list code more complex

- Kth element access requires linear scan??

Second Example: Queue ADT

We briefly mentioned it in Lecture 01…

- FIFO: First In First Out
- Queue operations

  create

  destroy

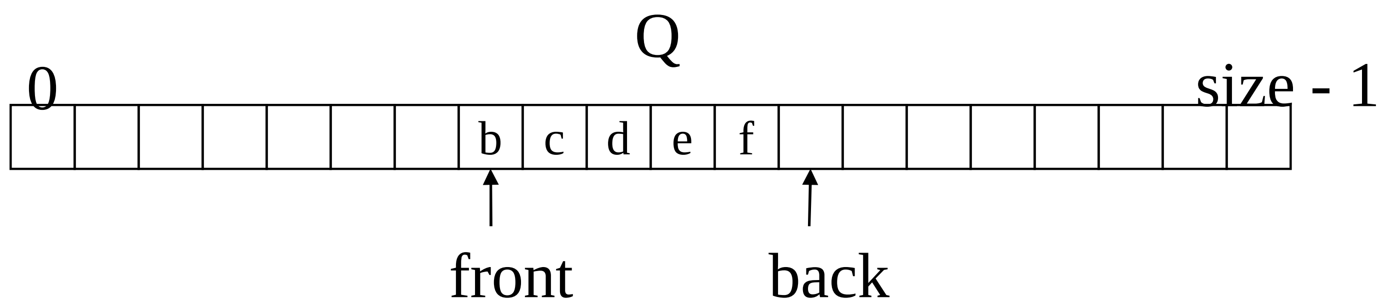
  enqueue

  dequeue

  is_empty

http://courses.cs.vt.edu/csonline/DataStructures/Lessons/QueuesImplementationView/applet.html

Circular Array Queue Data Structure




```
enqueue(Object x) {
   Q[back] = x ;
   back = (back + 1) % size
   }
dequeue() {
   x = Q[front] ;
   front = (front + 1) % size;
   return x ;
   }
```

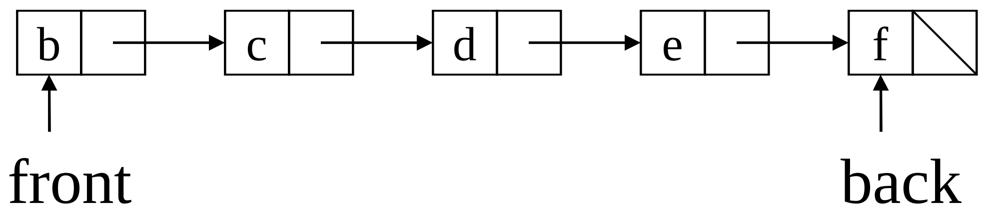
How test for empty list?

How to find K-th element in the queue?

What is complexity of these operations?

Limitations of this structure?

Linked List Queue Data Structure




```
void enqueue(Object x) {
  if (is_empty())
      front = back = new Node(x)
  else
      back.next = new Node(x)
      back = back.next
}
bool is_empty() {
  return front == null
}
```

```
Object dequeue() {
  assert(!is_empty)
  return_data = front.data
  temp = front
  front = front.next
  delete temp
  return return_data
}
```

Circular Array vs. Linked List

- Too much space, or not enough space
- Not as complex
- Could make array more robust

- Can grow as needed
- More memory per item in the queue
- Linked list code more complex

Applications of Queues

- Tree traversals, graph traversals
  - Will see in coming lectures
- Wherever fairness is needed:
  - Printer queues
  - Packets in a network
  - http requests at a web server
  - Etc.