4.3 Stacks, Queues, and Linked Lists
Data Types and Data Structures

Data types: Set of values and operations on those values.

- Some are built into the Java language: `int, double[], String, ...`
- Most are not: `Complex, Picture, Stack, Queue, ST, Graph, ...`

Data structures:

- Represent data or relationships among data.
- Some are built into Java language: arrays.
- Most are not: linked list, circular list, tree, sparse array, graph, ...
Collections

Fundamental data types:
• Set of operations (add, remove, test if empty) on generic data.
• Intent is clear when we insert.
• Which item do we remove?

Stack: [LIFO = last in first out]
• Remove the item most recently added.
• Ex: Pez, cafeteria trays, Web surfing.

Queue: [FIFO = first in, first out]
• Remove the item least recently added.
• Ex: Line for help in TA office hours.

Symbol table:
• Remove the item with a given key.
• Ex: Phone book.
Stack API

public class *StackOfStrings*

*StackOfStrings()*

create an empty stack

boolean *isEmpty()*

is the stack empty?

void *push*(*String item*)

push a string onto the stack

String *pop()*

pop the stack

Section 4.3
Stack Client Example 1: Reverse

```java
public class Reverse {
    public static void main(String[] args) {
        StackOfStrings stack = new StackOfStrings();
        while (!StdIn.isEmpty()) {
            String s = StdIn.readString();
            stack.push(s);
        }
        while (!stack.isEmpty()) {
            String s = stack.pop();
            StdOut.println(s);
        }
    }
}
```

% more tiny.txt
it was the best of times

% java Reverse < tiny.txt
times of best the was it

stack contents when standard input is empty

Section 4.3
Stack: Array Implementation

Array implementation of a stack.

- Use array $a[]$ to store $N$ items on stack.
- $\text{push()}$ add new item at $a[N]$.
- $\text{pop()}$ remove item from $a[N-1]$.

<table>
<thead>
<tr>
<th>a[]</th>
<th>to</th>
<th>be</th>
<th>or</th>
<th>not</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

$N$

```java
public class ArrayStackOfStrings {
    private String[] a;
    private int N = 0;

    public ArrayStackOfStrings(int max) {
        a = new String[max];
    }
    public boolean isEmpty() {
        return (N == 0);
    }
    public void push(String item) {
        a[N] = item; N++;
    }
    public String pop() {
        N--; return a[N];
    }
}
```

Section 4.3

Temporary solution: make client provide capacity

How big to make array? [stay tuned]

Stack and array contents after 4th push operation

Not or be to
## Array Stack: Test Client Trace

<table>
<thead>
<tr>
<th>StdIn</th>
<th>StdOut</th>
<th>N</th>
<th>(a[])</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>push</td>
<td>to</td>
<td>1</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>be</td>
<td>2</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>3</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>not</td>
<td>4</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>to</td>
<td>5</td>
<td>to</td>
</tr>
<tr>
<td>pop</td>
<td>-</td>
<td>4</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>be</td>
<td>5</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>4</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>3</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>that</td>
<td>4</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>3</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>2</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>1</td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>is</td>
<td>2</td>
<td>to</td>
</tr>
</tbody>
</table>

### Section 4.3
Array Stack: Performance

Running time: Push and pop take constant time.

Memory: Proportional to client-supplied capacity, not number of items.

Problem:
• API does not take capacity as argument (bad to change API).
• Client might not know what capacity to use.
• Client might use multiple stacks.

Challenge: Stack where capacity is not known ahead of time.

Section 4.3
Follow the arrow and Punch the appropriate dot.

Bush
Buchanan
Gore
Nader

(c) 2000 Mike Collins, Taterbrains.com
Sequential vs. Linked Allocation

**Sequential allocation:** Put items one after another.
- TOY: consecutive memory cells.
- Java: array of objects.

**Linked allocation:** Include in each object a link to the next one.
- TOY: link is memory address of next item.
- Java: link is reference to next item.

**Key distinctions:**
- Array: random access, fixed size.
- Linked list: sequential access, variable size.

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>B1</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>B2</td>
<td>&quot;Carol&quot;</td>
</tr>
<tr>
<td>B3</td>
<td>-</td>
</tr>
<tr>
<td>B4</td>
<td>-</td>
</tr>
<tr>
<td>B5</td>
<td>-</td>
</tr>
<tr>
<td>B6</td>
<td>-</td>
</tr>
<tr>
<td>B7</td>
<td>-</td>
</tr>
<tr>
<td>B8</td>
<td>-</td>
</tr>
<tr>
<td>B9</td>
<td>-</td>
</tr>
<tr>
<td>BA</td>
<td>-</td>
</tr>
<tr>
<td>BB</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>&quot;Carol&quot;</td>
</tr>
<tr>
<td>C1</td>
<td>null</td>
</tr>
<tr>
<td>C2</td>
<td>-</td>
</tr>
<tr>
<td>C3</td>
<td>-</td>
</tr>
<tr>
<td>C4</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>C5</td>
<td>CA</td>
</tr>
<tr>
<td>C6</td>
<td>-</td>
</tr>
<tr>
<td>C7</td>
<td>-</td>
</tr>
<tr>
<td>C8</td>
<td>-</td>
</tr>
<tr>
<td>C9</td>
<td>-</td>
</tr>
<tr>
<td>CA</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>CB</td>
<td>C0</td>
</tr>
</tbody>
</table>
Singly-Linked Data Structures

From the point of view of a particular object:
all of these structures look the same!

Sequential (this lecture)

Parent-link tree
Rho

Circular
General case

Multiply-linked data structures: Many more possibilities.

Section 4.3
Linked Lists

Linked list:
- A recursive data structure.
- An item plus a pointer to another linked list (or empty list).
  - Unwind recursion: linked list is a sequence of items.

Node data type:
- A reference to a `String`.
- A reference to another `Node`.

```java
public class Node {
    public String item;
    public Node next;
}
```

![Diagram showing a linked list with nodes Alice, Bob, Carol, and null. Alice points to Bob, Bob points to Carol, and Carol points to null. The special pointer value null terminates the list.]
Building a Linked List

Node third = new Node();
third.item = "Carol";
third.next = null;

Node second = new Node();
second.item = "Bob";
second.next = third;

Node first = new Node();
first.item = "Alice";
first.next = second;
Stack Push: Linked List Implementation

first

best → the → was → it

first second

best → the → was → it

Node second = first;

first = new Node();

first second

best → the → was → it

first second

of → best → the → was → it

first item = "of";
first.next = second;

Section 4.3
Stack Pop: Linked List Implementation

first

`String item = first.item;`

"of"

first = first.next;

return item;

garbage-collected

Section 4.3
Stack: Linked List Implementation

```java
public class LinkedStackOfStrings {
    private Node first = null;

    private class Node {
        private String item;
        private Node next;
    }

    public boolean isEmpty() { return first == null; }

    public void push(String item) {
        Node second = first;
        first = new Node();
        first.item = item;
        first.next = second;
    }

    public String pop() {
        String item = first.item;
        first = first.next;
        return item;
    }
}
```
Linked List Stack: Test Client Trace

```
push
  be
  or
  not
  to

pop
  be
  not
  be
  not
  that
  that
  or
  be
  is
```
Stack Data Structures: Tradeoffs

Two data structures to implement Stack data type.

Array:
- Every push/pop operation take constant time.
- But... must fix maximum capacity of stack ahead of time.

Linked list:
- Every push/pop operation takes constant time.
- Memory is proportional to number of items on stack.
- But... uses extra space and time to deal with references.
List Processing Challenge 1

What does the following code fragment do?

```java
for (Node x = first; x != null; x = x.next) {
    StdOut.println(x.item);
}
```

```
first

Alice ➔ Bob ➔ Carol ➔ null
```

Section 4.3
List Processing Challenge 2

What does the following code fragment do?

```java
Node last = new Node();
last.item = StdIn.readString();
last.next = null;
Node first = last;
while (!StdIn.isEmpty()) {
    last.next = new Node();
    last = last.next;
    last.item = StdIn.readString();
    last.next = null;
}
```

Diagram:
- `first` points to `Alice`
- `Alice` points to `Bob`
- `Bob` points to `Carol`
- `Carol` points to `null`
- `last` points to `null`