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: 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.

**Stacks**

**Stack API**

```java
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
}
```

**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);
        }
    }
}
```

```bash
% 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
**SeqENUENONAL 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.

**Building a Linked List**

**Stack: Array Implementation**

Array implementation of a stack.
- Use array $a[]$ to store $N$ items on stack.
- $push()$: add new item at $a[N]$.
- $pop()$: remove item from $a[N-1]$.

```java
class ArrayStackOfStrings {
    private String[] a;
    private int N = 0;
    public ArrayStackOfStrings() {
        a = new String[10];
        N = 0;
    }
    public ArrayStackOfStrings(int max) {
        a = new String[max];
        N = 0;
    }
    public boolean isEmpty() { return (N == 0); }
    public void push(String item) { a[N] = item; ++N; }
    public String pop() { N--; return a[N + 1]; }
}
```

**Linked Lists**

- 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;
}
```

**Stack Push: Linked List Implementation**

**Linked Lists**

- With an array.
- With a linked list.

Building a linked list:

- First.
- Second.
- Third.
- Null.

```java
public class ArrayStackOfStrings {
    private String[] a;
    private int N = 0;
    public ArrayStackOfStrings() {
        a = new String[10];
        N = 0;
    }
    public ArrayStackOfStrings(int max) {
        a = new String[max];
        N = 0;
    }
    public boolean isEmpty() { return (N == 0); }
    public void push(String item) { a[N] = item; ++N; }
    public String pop() { N--; return a[N + 1]; }
}
```
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;
    }
}
```

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 take constant time.
- Memory is proportional to number of items on stack.
- But... uses extra space and time to deal with references.

List Processing Challenge 2

Q. 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;
}
```