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
class StackOfStrings {
    private Stack<String> stack;
    public StackOfStrings() {
        stack = new Stack<>();
    }
    public boolean isEmpty() {
        return stack.isEmpty();
    }
    public void push(String item) {
        stack.push(item);  // push a string onto the stack
    }
    public String pop() {
        return stack.pop();  // pop the stack
    }
    public void printStack() {
        stack.print();
    }
}
```

Stack Client Example 1: Reverse

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

Input:
```
% more tiny.txt
it was the best of times
```

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

Note: stack contents when standard input is empty.
Stack: Array Implementation

Array implementation of a stack.
- Use array $a[]$ to store $n$ items on stack.
  - $push()$: add new item at $a[0]$.
  - $pop()$: remove item from $a[n-1]$.

Sequential vs. Linked Allocation

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

Linked allocation. Include 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

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;

Linked Lists

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

Linked allocation. Include 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.

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.

Stack Push: Linked List Implementation

first = new Node();
first.item = "Alice";
first.next = second;

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

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;

public class ArrayStackOfStrings {
  private String[] a;
  private int N = 0;
  public ArrayStackOfStrings() {
    a = new String[100];
    N = 0;
  }
  public void push(String item) {
    if (N == a.length) {
      a = Arrays.copyOf(a, a.length * 2);
    }
    a[N++] = item;
  }
  public String pop() {
    if (N == 0) {
      return null;
    }
    N--;
    return a[N - 1];
  }
  public boolean isEmpty() {
    return N == 0;
  }
}

public class Node {
  public String item;
  public Node next;
}
Stack Pop: Linked List Implementation

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

    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 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 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;
}
```