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.

Stacks

Stack API

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);
        }
    }
}
```

Guitar Hero Assignment

```
more tiny tat
it was the best of times

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

Stack contents when standard input is empty
Stack: Array Implementation

Array implementation of a stack.
- Use array \( a[] \) to store \( n \) items on stack.
- \( \text{push}(\cdot) \): add new item to \( a[\text{index}] \).
- \( \text{pop}() \): remove item from \( a[\text{index}-1] \).

Linked Lists

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.

Linked Lists

A recursive data structure.
- An item plus a reference to another 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.

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

Singly-Linked Data Structures

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

Multiply-linked data structures. Many more possibilities.

Building a Linked List

Add a new item (in this lecture):
- First item is "Alice".
- Second item is "Bob".
- Third item is "Carol".
- Null after last item.

Null:
- Value: "null".
- First: null.
- Second: null.
- Third: null.
Stack Push: Linked List Implementation

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

Stack Pop: Linked List Implementation

first = first.next;
return item;

First = first.next;
getReference()
first = first.next;
String item = first.item;

Stack: Linked List Implementation

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;
  }
}

List Processing Challenge 1

Q. What does the following code fragment do?

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

```
null
```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```