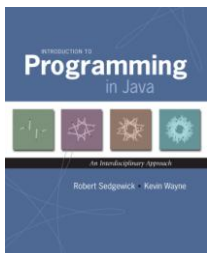


Data Types and Data Structures

4.3 Stacks, Queues, and Linked Lists



Introduction to Programming in Java: An Interdisciplinary Approach · Robert Sedgewick and Kevin Wayne · Copyright © 2002-2010 · 12/11/2012 11:30:53

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`, ...

↑
this lecture

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, ...

↑
this lecture

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]

← This lecture

- Remove the item most recently added.
- Ex: Pez, cafeteria trays, Web surfing.

Queue. [FIFO = first in, first out]

← Guitar Hero Assignment

- 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

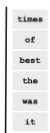
```
public class *StackOfStrings
{
    *StackOfStrings() create an empty stack
    boolean isEmpty() is the stack empty?
    void push(String item) push a string onto the stack
    String pop() pop the stack
}
```



Stack Client Example 1: Reverse

```
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
```

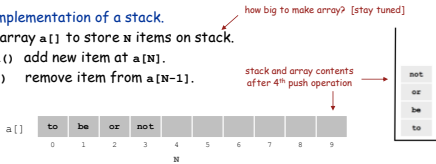


← stack contents when standard input is empty

Stack: Array Implementation

Array implementation of a stack.

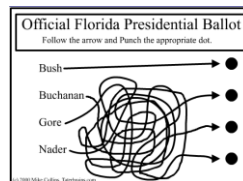
- Use array a[] to store n items on stack.
- push() add new item at a[N].
- pop() remove item from a[N-1].



```

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]; }
}
    
```

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.

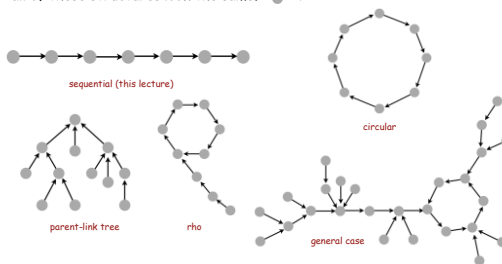
- Array: random access, fixed size.
- Linked list: sequential access, variable size.

addr	value	addr	value
B0	"Alice"	C0	"Carol"
B1	"Bob"	C1	null
B2	"Carol"	C2	-
B3	-	C3	-
B4	-	C4	"Alice"
B5	-	C5	CA
B6	-	C6	-
B7	-	C7	-
B8	-	C8	-
B9	-	C9	-
BA	-	CA	"Bob"
BB	-	CB	CO

array (B0) linked list (C4)

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.

Linked Lists

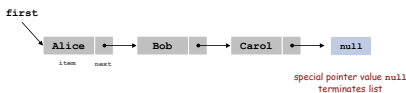
Linked list.

- 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.

```

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



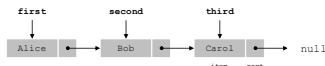
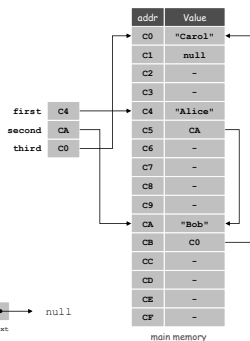
Building a Linked List

```

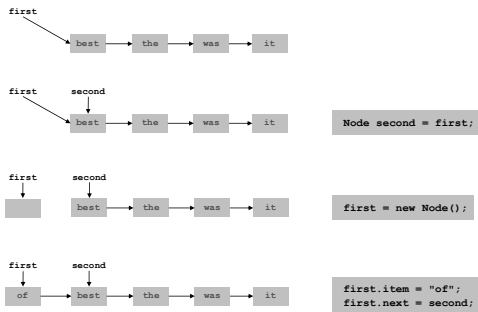
Node third = new Node();
third.item = "Carol";
third.next = null;

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

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

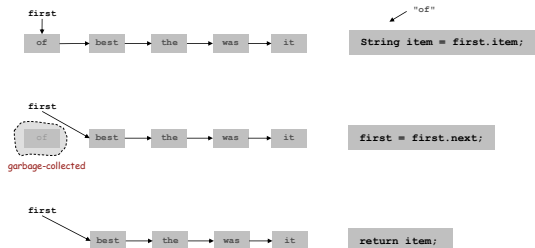


Stack Push: Linked List Implementation



16

Stack Pop: Linked List Implementation



17

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; }

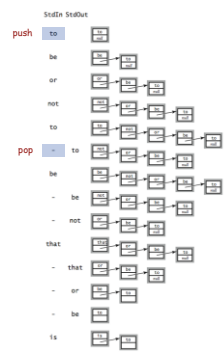
    public void push(String item) {
        Node second = first;
        first = new Node();
        first.item = item;
        first.next = second;
    }

    public String pop() {
        String item = first.item;
        first = first.next;
        return item;
    }
}
    
```

stack and linked list contents after 4th push operation

18

Linked List Stack: Test Client Trace



19

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.



20

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);
}
    
```

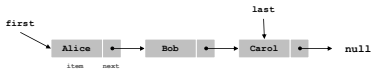


21

List Processing Challenge 2

Q. What does the following code fragment do?

```
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;
}
```



22