

Stacks

Dr. Mattox Beckman

ILLINOIS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Objectives

- ▶ Be able to describe the *stack* operations.
- ▶ Describe two stack implementations (array and linked).
- ▶ Know how long each operation takes in each implementation.
- ▶ The special way to handle functional stacks.
- ▶ Vocabulary: LIFO

Stacks

- ▶ What are the stack operations?
 - ▶ Add things by placing them on the top. “Pushing”
 - ▶ Remove things by taking them off the top. “Popping”
 - ▶ Look at the top of the stack.

Stacks are called LIFO for this reason.

- ▶ How long do they take? $\mathcal{O}(1)$. Always.
- ▶ Idea: if you know how to write a list, you know how to write a stack.

Array Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



- The stack pointer is the first empty space in the array!
- How can we define stacks in Clojure?

Array Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

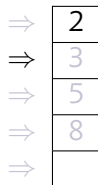
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



- The stack pointer is the first empty space in the array!
- How can we define stacks in Clojure?

Array Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

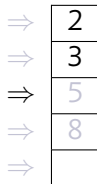
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



- The stack pointer is the first empty space in the array!
- How can we define stacks in Clojure?

Array Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

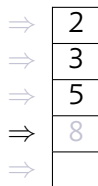
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



- The stack pointer is the first empty space in the array!
- How can we define stacks in Clojure?

Array Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

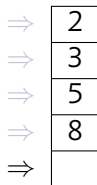
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



- The stack pointer is the first empty space in the array!
- How can we define stacks in Clojure?

Array Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

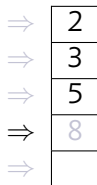
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



- The stack pointer is the first empty space in the array!
- How can we define stacks in Clojure?

Array Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

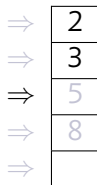
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



- The stack pointer is the first empty space in the array!
- How can we define stacks in Clojure?

Array Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

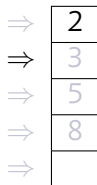
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



- The stack pointer is the first empty space in the array!
- How can we define stacks in Clojure?

Array Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



- The stack pointer is the first empty space in the array!
- How can we define stacks in Clojure?

Stack Class

```
1 (defmtype Stack [top data]) ;; Our custom macro
2 (defn make-stack []
3   (Stack. 0 (vec (take 10 (repeat 0)))))
4 (defn push [s elt]
5   (reset-data! s (assoc (get-data s) (get-top s) elt))
6   (swap-top! s inc))
7 (defn pop [s]
8   (swap-top! s dec)
9   ((get-data s) (get-top s)))
10 (defn top [s]
11   ((get-data s) (dec (get-top s))))
```

Using a Linked List

- ▶ We can use linked lists for a stack. Why would we want to do that, instead of using an array?
- ▶ Where should new items be placed?

List Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop

top 

List Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

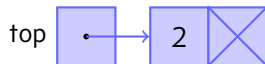
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



List Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



List Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

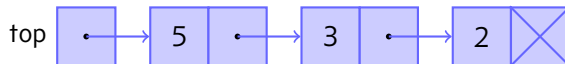
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



List Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



List Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

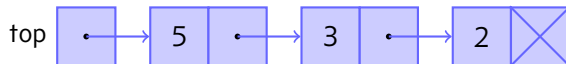
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



List Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



List Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

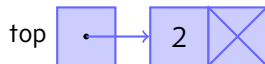
⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop



List Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop

top 

List Operations

- Observe the following sequence of operations:

⇒ push 2

⇒ push 3

⇒ push 5

⇒ push 8

⇒ pop

⇒ pop

⇒ pop

⇒ pop

What should the definition look like?

Linked Stacks

```
1 (defmtype Stack [data])
2
3 (defn make-stack [] (Stack. nil))
4
5 (defn push [s elt]
6   (swap-data! s #(cons elt %)))
7
8 (defn pop [s]
9   (let [tos (first (get-data s))]
10     (swap-data! s rest)
11     tos))
12
13 (defn top [s]
14   (first (get-data s)))
```

Sample Run

```
1 user> (def s (make-stack))
2 #'user/s
3 user> (push s 10)
4 (10)
5 user> (push s 30)
6 (30 10)
7 user> (push s 3)
8 (3 30 10)
9 user> (top s)
10 3
11 user> (pop s)
12 3
13 user> (top s)
14 30
```

When is a list a stack?

- Sometimes a list can behave like a stack.

```
1 (defn calc [xx st]
2   (cond (empty? xx) (first st)
3         (= (first xx) '+)
4           (calc (rest xx)
5                 (cons (+ (first st) (second st))
6                       (rest (rest st)))))
7   :else (calc (rest xx)
8               (cons (first xx) st))))
9 (calc '(2 + 3 4 + +) '(1))
10 ;; => (calc '(+ 3 4 + +) '(2 1))
11 ;; => (calc '(3 4 + +) '(3))
12 ;; => (calc '(4 + +) '(3 3))
13 ;; => (calc '(+ +) '(4 3 3))
14 ;; => (calc '(+) '(7 3))
15 ;; => (calc '() '(10)) ;; => 10
```