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

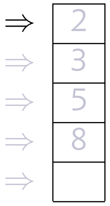
- What are the stack operations?
  - Add things by placing them on the top. “Pushing”
  - Remove things by taking them off the top. “Popping”
  - Look at the top of the stack.

  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

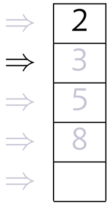


- The stack pointer is the first empty space in the array!
- How can we define stacks in Clojure?

Array Operations

- Observe the following sequence of operations:

⇒ push 2
⇒ push 3
⇒ push 5
⇒ push 8
⇒ pop
⇒ pop
⇒ pop
⇒ pop

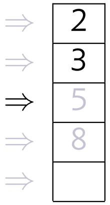

2
3
5
8

- 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

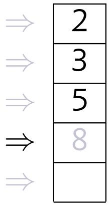

2
3
5
8

- 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

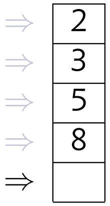

2
3
5
8

- 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

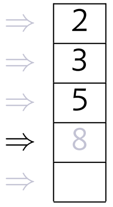

2
3
5
8

- 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



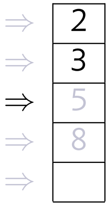
⇒ 2
⇒ 3
⇒ 5
⇒ 8
⇒

- 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



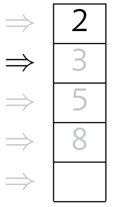
⇒ 2
⇒ 3
⇒ 5
⇒ 8
⇒

- 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



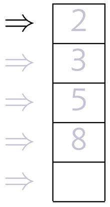
⇒ 2
⇒ 3
⇒ 5
⇒ 8
⇒

- 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



⇒ 2
⇒ 3
⇒ 5
⇒ 8
⇒

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

Stack Class

```clojure
(defmtype Stack [top data]) ;; Our custom macro
(defn make-stack []
   (Stack. 0 (vec (take 10 (repeat 0)))))
(defn push [s elt]
   (reset-data! s (assoc (get-data s) (get-top s) elt))
   (swap-top! s inc))
(defn pop [s]
   (swap-top! s dec)
   ((get-data s) (get-top s)))
(defn top [s]
   ((get-data s) (dec (get-top s))))
```

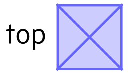
Using a Linked List

- 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:

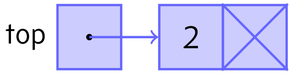
⇒ push 2
⇒ push 3
⇒ push 5
⇒ push 8
⇒ pop
⇒ pop
⇒ pop
⇒ pop



List Operations

- Observe the following sequence of operations:

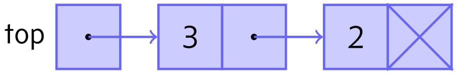
⇒ push 2
⇒ push 3
⇒ push 5
⇒ push 8
⇒ pop
⇒ pop
⇒ pop
⇒ pop



List Operations

- Observe the following sequence of operations:

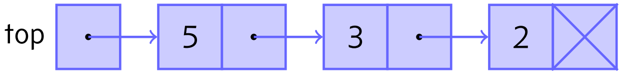
⇒ push 2
⇒ push 3
⇒ push 5
⇒ push 8
⇒ pop
⇒ pop
⇒ pop
⇒ pop



List Operations

- Observe the following sequence of operations:

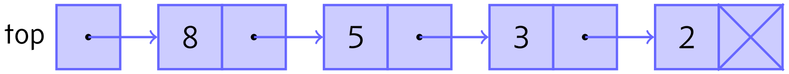
⇒ push 2
⇒ push 3
⇒ push 5
⇒ push 8
⇒ pop
⇒ pop
⇒ pop
⇒ pop



List Operations

- Observe the following sequence of operations:

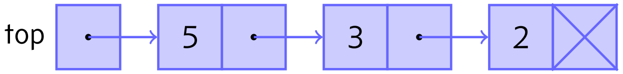
⇒ push 2
⇒ push 3
⇒ push 5
⇒ push 8
⇒ pop
⇒ pop
⇒ pop
⇒ pop



List Operations

- Observe the following sequence of operations:

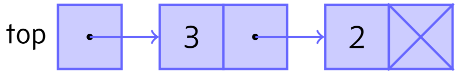
⇒ push 2
⇒ push 3
⇒ push 5
⇒ push 8
⇒ pop
⇒ pop
⇒ pop
⇒ pop



List Operations

- Observe the following sequence of operations:

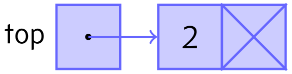
⇒ push 2
⇒ push 3
⇒ push 5
⇒ push 8
⇒ pop
⇒ pop
⇒ pop
⇒ pop



List Operations

- Observe the following sequence of operations:

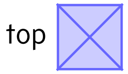
⇒ push 2
⇒ push 3
⇒ push 5
⇒ push 8
⇒ pop
⇒ pop
⇒ pop
⇒ pop



List Operations

- Observe the following sequence of operations:

⇒ push 2
⇒ push 3
⇒ push 5
⇒ push 8
⇒ pop
⇒ pop
⇒ pop
⇒ pop



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

```
(defmtype Stack [data])

(defn make-stack [] (Stack. nil))

(defn push [s elt]
  (swap-data! s #(cons elt %)))

(defn pop [s]
  (let [tos (first (get-data s))]
    (swap-data! s rest)
    tos))

(defn top [s]
   (first (get-data s)))
```

Sample Run

```
user> (def s (make-stack))
#'user/s
user> (push s 10)
(10)
user> (push s 30)
(30 10)
user> (push s 3)
(3 30 10)
user> (top s)
3
user> (pop s)
3
user> (top s)
30
```

When is a list a stack?

- Sometimes a list can behave like a stack.

```
(defn calc [xx st]
  (cond (empty? xx) (first st)
        (= (first xx) '+)
          (calc (rest xx)
                (cons (+ (first st) (second st))
                      (rest (rest st))))
        :else (calc (rest xx)
                    (cons (first xx) st))))
(calc '(2 + 3 4 + +) '(1))
;; => (calc '(+ 3 4 + +) '(2 1))
;; => (calc '(3 4 + +) '(3))
;; => (calc '(4 + +) '(3 3))
;; => (calc '(+ +) '(4 3 3))
;; => (calc '(+) '(7 3))
;; => (calc '() '(10))  ;; => 10
```