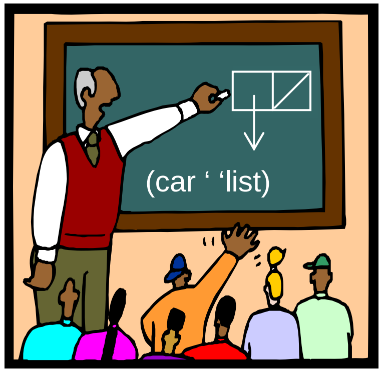
6.001 recitation 3/16/07

- tags
- stacks and queues



Dr. Kimberle Koile

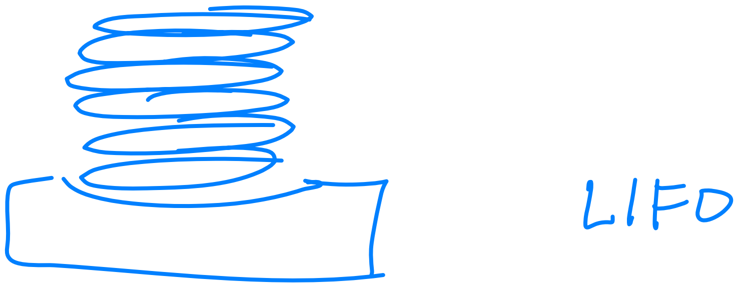
tags

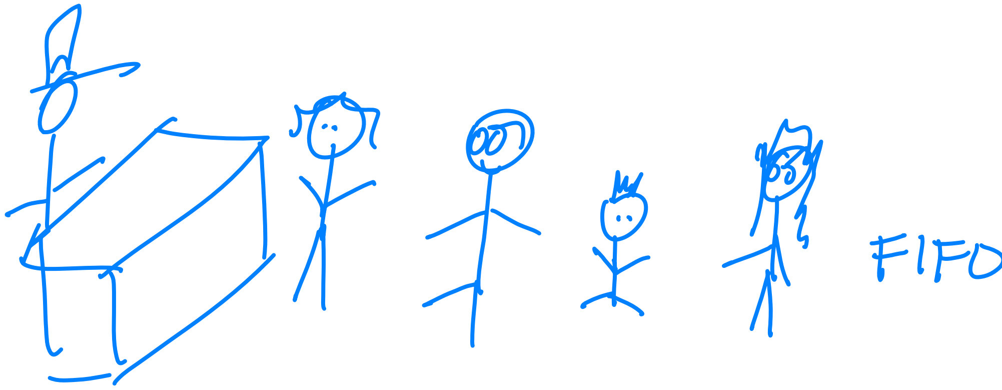
what are tags?

How are they useful?

what is an example of a tagged data structure?

stacks and queues





stacks

- **constructor:**
  (make-stack)

- **selectors:**
  (top stack)

  (stack-elements stack)

- **operations:**

  (stack? stack)

  (empty-stack? stack)

  (insert-stack-elt stack elt)

  (delete-stack-elt stack)

stacks

```
(define (tagged-list? tag l) (and (pair? l) (eq? tag (car l))))

(define *stack-tag* 'stack)
```

- **constructor:**
  ```
  (define (make-stack)
  ```
- **selectors:**
  ```
  (define (top stack)
  ```
- **operations:**
  ```
  (define (stack?  stack)

  (define (empty-stack? stack)
  ```

```
(define (make-stack)
  (list *stack-tag*))

(define (top stack)
  (if (stack? stack)
      (if (not (empty-stack? stack))
          (car (stack-elements stack))
          (error "stack is empty; no top"))
      (error "top on a non-stack")))

(define (stack? stack)
        (tagged-list?  *stack-tag* stack))

(define (empty-stack? stack)
  (if (stack? stack)
      (null? (stack-elements stack))
      (error "empty-stack on a non-stack")))
```

stack problems

1. Fill in the code for insert-stack-elt (aka push) for a stack.

```
(define (insert-stack-elt  element stack)

  (if (stack? stack)

     (cons [                    ]

            (cons [                ] [                          ]

            )

  (error "Insert on a non-stack")))
```

stack-tag

(cons element (stack-elements stack)))

stack problems

2. Write delete-stack-elt (aka pop) for a stack. This version of pop should return a new stack that contains all elements except the top. (Don't forget the two error checks.)

```
(define delete-stack-elt  (stack)

)
```

```
(define (delete-stack-elt stack)
  (if (stack? stack)
      (if (empty-stack? stack)
           (error "stack is empty; can't delete")
          (cons *stack-tag* (cdr (stack-elements stack))))))
```

queues

```
(define (tagged-list? tag l) (and (pair? l) (eq? tag (car l))))
```

```
(define *queue-tag* 'queue)
```

- **constructor:**
  (make-queue)

- **selectors:**
  (front-queue queue)

  (queue-elements queue)

- **operations:**
  (queue? queue)

  (empty-queue? queue)

  (insert-queue-elt queu elt)

  (delete-queue-elt queue)

queue problems

3. Write insert-queue-elt for a queue. (Don't forget an error check.)

```
(define insert-queue-elt  (queue)

)
```

```
(define (insert-queue-elt element queue)
  (if (queue? queue)
      (cons *queue-tag*
           (append (queue-elements queue)  (list element)))
      (error "Push on a non-queue")))
```

queue problems

4. Write delete-queue-elt for a queue. (Don't forget an error check.)

```
(define delete-queue-elt  (queue)

)
```

```
(define (delete-queue-elt queue)
  (if (queue? queue)
      (if (empty-queue? queue)
          (error "queue is empty; can't delete element")
          (cons *queue-tag* (cdr (queue-elements queue))))
      (error "can't delete element on a non-queue")))
```

stacks and queues

- **constructor:**
  (make-it)

- **selectors:**
  (first-elt s-or-q)

  (elements s-or-q)

- **operations:**

  (is-type? s-or-q)

  (empty? s-or-q)

  (insert-element s-or-q elt)

  (delete-element s-or-q)

* dispatch on type using check with stack? + queue?

stacks and queues

5. Write a delete-elt procedure that works on either stacks or queues.

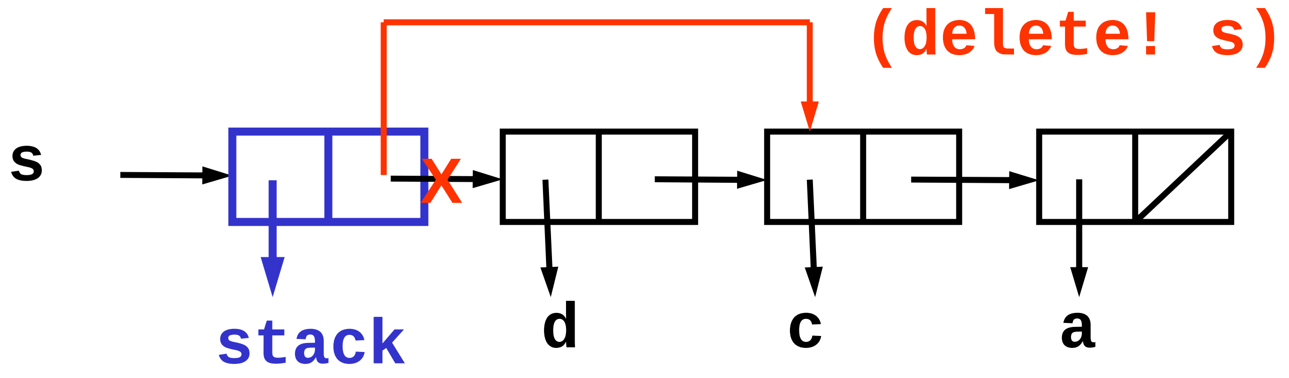
```
(define delete-elt s-or-q)

                                        (define (delete-elt q-or-s)
                                         (cond ((queue? queue)
                                                 (delete-queue-elt q-or-s))
                                               ((stack? q-or-s)
                                                 (delete-stack-elt q-or-s))
                                               (else (error "not queue or stack")))

)
```

stacks and queues

What if we could change the data structures rather than copying them?


(delete! s)
s
x
stack
d
c
a