

CS61A Lecture 13

2011-07-12
Colleen Lewis



get and put

Demo



get and put

```
STk> (put 'richmond 'berkeley '$1.75)
ok
```

```
STk> (get 'richmond 'berkeley)
$1.75
```

```
STk> (get 'berkeley 'richmond)
#f
```

The reverse isn't automatically added. Returns #f if there is no entry



get and put

```
STk> (put 'berkeley 'fremont '$4.30)
ok
```

```
STk> (get 'berkeley 'fremont)
$4.30
```

We can add lots of things with the same first word



get and put

```
(put 'berkeley 'richmond '$1.75)
(put 'richmond 'berkeley '$1.75)
```

```
(put 'berkeley 'fremont '$4.30)
(put 'fremont 'berkeley '$4.30)
```

```
(put 'berkeley 'SFO '$8.65)
(put 'SFO 'berkeley '$8.65)
```

```
(define (bart-cost stop1 stop2)
  (get stop1 stop2))
```



get and put

```
STk> (bart-cost 'fremont 'berkeley)
$4.30
```

```
STk> (bart-cost 'fremont 'SFO)
#f
```



put creates a table

Berkeley	Fremont	\$4.30
	Richmond	\$1.75
	SFO	\$8.65
SFO	Berkeley	\$8.65
Richmond	Berkeley	\$1.75
Fremont	Berkeley	\$4.30
	SFO	\$10.55
	Richmond	\$4.85

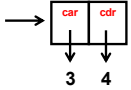
How do I add more Bart Stops?

```
(define (bart-cost stop1 stop2)
  (get stop1 stop2))
```

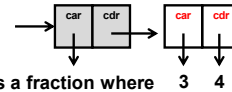
- A) Modify the function `bart-cost`.
- B) Call `put` for more stops
- C) All of the above
- D) None of the above
- E) ??

Tagged Data

Removing Ambiguity – Tagging data

- What does this represent? → 

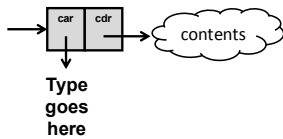
- A) $\frac{3}{4}$
- B) $3+4i$
- C) 3×4 rectangle



“The next thing is a fraction where the `car` is the numerator and the `cdr` is the denominator”

attach-tag

```
(define (attach-tag type contents)
  (cons type contents))
```

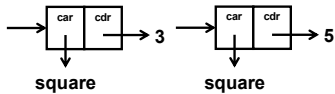


Selectors

```
(define (type-tag tagged-data)
  (car tagged-data))
```

```
(define (contents tagged-data)
  (cdr tagged-data))
```

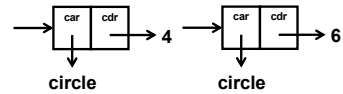
tagging data



```
(define (make-square side)
  (attach-tag 'square side))
```

```
(define square3 (make-square 3))
(define square5 (make-square 5))
```

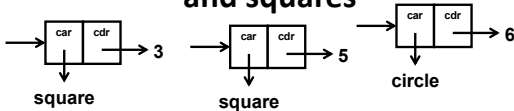
tagging data



```
(define (make-circle side)
  (attach-tag 'circle side))
```

```
(define circle4 (make-circle 4))
(define circle6 (make-circle 6))
```

Write area to work for circles and squares



```
(define (area shape)
```

How many times did you call `type-tag`?

A)1 B)2 C)3 D)4 E)5

How do you call area?

A)
STk> ((make-square 4) 'area)
16

B)
STk> (area (make-square 4))
16

C)
STk> (area (contents (make-square 4)))
16

What is returned by `make-square`?

- A) A shape
- B) A pair, where the `car` is the word `square`.
- C) Tagged data
- D) A function

Could you re-write `area`?

```
(put 'square 'area
  (lambda (s) (* s s)))
(put 'circle 'area
  (lambda (r) (* pi r r)))
```

```
STk> (area (make-square 3))
9
```

Show the answer with:

A) Live coding B) PowerPoint C) Chalk

How do I add another shape?

- A) Modify the function area2.
- B) Call put for more shapes
- C) All of the above
- D) None of the above
- E) ??



```
(put 'square 'area
  (lambda (s) (* s s)))
(put 'circle 'area
  (lambda (r) (* pi r r)))
(put 'square 'perimeter
  (lambda (s) (* 4 s)))
(put 'circle 'perimeter
  (lambda (r) (* 2 pi r)))

(define (area2 shape)
  ((get (type-tag shape) 'area)
   (contents shape)))
(define (perimeter2 shape)
  ((get (type-tag shape) 'perimeter)
   (contents shape)))
```

perimeter



Error checking: operate

```
(define (operate op obj)
  (let ((proc (get (type-tag obj) op)))
    (if proc
        (proc (contents obj))
        (error "Unknown op for type"))))
(define (area3 shape)
  (operate 'area shape))
```



area data directed solution w/out and w/ put

```
(define (area shape)
  (cond
    ((eq? (type-tag shape) 'square)
     (* (contents shape) (contents shape)))
    ((eq? (type-tag shape) 'circle)
     (* pi (contents shape) (contents shape)))
    (else (error "Unknown shape --- AREA"))))

(define (area3 shape)
  (operate 'area shape))
(put 'square 'area (lambda (s) (* s s)))
(put 'circle 'area (lambda (r) (* pi r r)))
```



Message Passing

VERY DIFFERENT



Message passing make-square

```
(define (make-square-mp side)
  (lambda (message)
    (cond
      ((eq? message 'area)
       (* side side))
      ((eq? message 'perimeter)
       (* 4 side))
      (else (error "unknown msg")))))
```




How do you call area?

A)
STk> ((make-square-mp 4) 'area)
16


B)
STk> (area (make-square-mp 4))
16

C)
STk> (area (contents (make-square-mp 4)))
16




Easier way to call area

```
(define (make-square-mp side)
  (lambda (message)
    (cond
      ((eq? message 'area)
       (* side side))
      ((eq? message 'perimeter)
       (* 4 side))
      (else (error "unknown msg")))))
(define (area shape)
  (shape 'area))
```




Message passing versus data directed?

The constructors for this style return tagged data:

- A) Data Directed
 - B) Message Passing
 - C) Both
 - D) Neither
- 


Message passing versus data directed?

This method CAN use `put/get` tables

- A) Data Directed
 - B) Message Passing
 - C) Both
 - D) Neither
- 


Message passing versus data directed?

This method used `put/get` tables in today's lecture

- A) Data Directed
 - B) Message Passing
 - C) Both
 - D) Neither
- 

Message passing versus data directed?

The constructors for this style return a function:

- A) Data Directed
 - B) Message Passing
 - C) Both
 - D) Neither
- 

Message passing versus data directed?

If you don't use `put`, there are a bunch of `cond` cases to handle each type of data in any "operation" e.g. `area`:

- A) Data Directed
- B) Message Passing
- C) Both
- D) Neither



Message passing versus data directed?

If you don't use `put`, there are a bunch of `cond` cases to handle each type of "operation" for any data (e.g. `circle`):

- A) Data Directed
- B) Message Passing
- C) Both
- D) Neither



Solutions



area solution

```
(define (area shape)
  (cond
    ((eq? (type-tag shape) 'square)
     (* (contents shape) (contents shape)))
    ((eq? (type-tag shape) 'circle)
     (* pi (contents shape) (contents shape)))
    (else (error "Unknown shape --- AREA"))))
```



area2 solution

```
(put 'square 'area
     (lambda (s) (* s s)))
(put 'circle 'area
     (lambda (r) (* 3.1415 r r)))

(define (area2 shape)
  ((get (type-tag shape) 'area)
   (contents shape)))
```



SOLUTION count-pairs

```
STk>(count-pairs (list 1 2 3 4))
4
STk>(count-pairs (list (cons 1 2) 3 4))
4
(define (count-pairs struct)
  (if (not (pair? struct))
      0
      (+ 1 (count-pairs (car struct))
         (count-pairs (cdr struct)))))
```

