```
#lang racket

(struct store (hash last) #:transparent)

(define (make-store)
  (store (make-immutable-hash) -1)) ; our store entries will start at 0

(define s-init (make-store))
s-init

(define (hash-update old k v)
  (let ((oldpairs (filter (lambda (x) (not (equal? (first x) k))) ; drop existing k entry if any
                          (hash->list old))))
    (make-immutable-hash (cons (list k v) oldpairs)))) ; add new k entry and return

(hash-update (store-hash s-init) 0 30)

(define (add-to-store old v)
  (let ((last (store-last old)))
    (store (hash-update (store-hash old) (+ 1 last) v) ; add new entry to hash
           (+ 1 last) ; update last identifier
           )))

; As always, this returns a copy, not a modified store!

(add-to-store s-init 10)
(define s0 (add-to-store s-init "Calvin"))
(define s1 (add-to-store s0 "Captain Haddock"))
s1

(define (print-contents s)
  (printf (foldl (lambda (x y)(string-append (number->string (first x))
                                             ":"
                                             (if (string? (second x))
                                                 (second x)
                                                 (number->string (second x)))
                                             "\n"
                                             y))
                 ""
                 (hash->list (store-hash s)))))

(print-contents s1)

(define (update-store old k v)
  (let ((last (store-last old))
        (h (store-hash old)))
    (store (hash-update h k v) last)))
```

```
48
49  (print-contents (update-store s1 0 "Peaches"))
50
51  (define (get-value s k)
52    (first (hash-ref (store-hash s ) k)))
53
54  (get-value s1 0)  ; the update-store call above did not change the value!
55
56  ;-----------------------------------------------------------------------
56  --
57  ; Implementing banking using store-passing style
58  ;-----------------------------------------------------------------------
58  --
59
60  (define (open-bank)
61    (make-store))
62
63  (define bank-init (open-bank))
64  bank-init
65
66  (define (open-account bank)
67    (let ((updated (add-to-store bank 0))) ; new accounts start with 0
67  balance
68      (let ((acctno (store-last updated)))
69        (printf (string-append "Opened account no. " (number->string
69  acctno) ".\n"))
70        updated))) ;return updated bank
71
72  (define bank0 (open-account bank-init))
73  (define bank1 (open-account bank0))
74  (define bank2 (open-account bank1))
75
76  (print-contents bank2)
77
78  (define (deposit bank acct amt)
79    (printf (string-append "Deposited " (number->string amt) " in acct. "
79  (number->string acct) ".\n"))
80    (update-store bank acct (+ amt (get-value bank acct))))
81
82  (define bank3 (deposit (deposit (deposit bank2 0 100) 1 25) 2 50))
83
84  (print-contents bank3)
85
86  (define (withdraw bank acct amt)
87    (printf (string-append "Withdrew " (number->string amt) " from acct. "
87  (number->string acct) ".\n"))
88    (update-store bank acct (- (get-value bank acct) amt)))
89
90  (define bank4 (withdraw bank3 2 100))
91
92  (print-contents bank4)
```

```
 93
 94 (define (transfer bank from to amt)
 95   (printf (string-append "Tranferred" (number->string amt)
 96                          "to acct. " (number->string to)
 97                          " from acct. " (number->string from) ".\n"))
 98   (let ((withdrew (withdraw bank from amt)))
 99     (deposit withdrew to amt)))
100
101 (define bank5 (transfer bank4 0 2 101))
102 (print-contents bank5)
103 (transfer bank1 0 1 100)
```