

A Tool for Defining Recursive Data Types

Exercise 2.21

```
(define-datatype env env?
  (empty-env)
  (extended-env
   (saved-var identifier?)
   (saved-val schemeval?)
   (saved-env env?)))

(define has-binding?
  (lambda (s e)
    (cases env e
      (empty-env () #f)
      (extended-env (var val env)
                     (if (eqv? s var)
                         #t
                         (has-binding? s env))))))

(define identifier? symbol?)
(define schemeval?
  (lambda (a)
    #t))
```

Exercise 2.22

```
(define-datatype stack stack?
  (empty-stack)
  (non-empty-stack
   (first schemeval?)
   (rest stack?)))

(define push
  (lambda (v s)
```

```

(non-empty-stack v s)))

(define pop
  (lambda (s)
    (cases stack s
      (empty-stack () (report-stack-is-empty 'pop))
      (non-empty-stack (first rest)
                        rest))))

(define top
  (lambda (s)
    (cases stack s
      (empty-stack () (report-stack-is-empty 'top))
      (non-empty-stack (first rest)
                        first))))

(define empty-stack?
  (lambda (s)
    (cases stack s
      (empty-stack () #t)
      (non-empty-stack (f r) #f))))

(define report-stack-is-empty
  (lambda (observer)
    (eopl:error 'empty-stack "Called ~s on an empty stack"
                 observer)))

```

Exercise 2.23

```

(define identifier2?
  (lambda (i)
    (and (symbol? i)
         (not (eqv? i 'lambda)))))

```

Exercise 2.24

```
(define bintree-to-list
  (lambda (btree)
    (cases bintree btree
      (leaf-node (num) (list 'leaf-node num))
      (interior-node (key left right)
        (list 'interior-node
              key
              (bintree-to-list left)
              (bintree-to-list right))))))
```