Scheme

1. Special Forms

   case - (case expr clauses)
   Works like cond, except the test of each clause is a list of numbers and symbols to compare against the value of expr.

Problems

A message-passing object definition:

(define (make-binary-operation name op)
  (lambda (message)
    (case message
      ((NAME)
       name)
      ((OPERATE)
       (lambda (a b)
         (op a b)))
      (else
       (error "binop can’t" message))))))

(define binop (make-binary-operation ’glue (lambda (x y) (append x y))))

((binop ’OPERATE )’(1) ’(2))

1. Stack object implementation

(a) Complete the skeleton for the stack object given below. The skeleton comprises everything but the method definitions.

(define (make-stack)
  (let ((vals ’())))
(b) Add a method called EMPTY? which returns #t if the stack is empty.
(c) Add a method called CLEAR which empties the stack of any elements it may contain.
(d) Add a method called PEEK which returns the top element of the stack, leaving the stack unchanged. If the stack is empty, signal a “stack underflow” error.
(e) Add a method called PUSH which allows an element to be added to the top of the stack.
(f) Add a method called POP which removes and returns the top element of the stack. Remember to program defensively.

2. Write a procedure called push-all which takes a stack and a list and pushes all the elements of the list onto the stack. It should return the stack.

(define (push-all stack lst))

3. Write a procedure called pop-all which takes a stack and pops elements off it until it becomes empty, adding each element to the output list.

(define (pop-all stack))

4. Write reverse.

(define (reverse lst))