```
;;; File: save_continuation.scm

;;; This is a simple example to show how
;;; call-with-current-continuation can be used to implement exception
;;; handling.  The example is so trivial that it could be done more
;;; simply; the point is just to show how call/cc can be used.

;;; We will define a procedure (main_loop) which when invoked asks the
;;; user to input a number different from 0.  If the user inputs the
;;; number 0 or anything other than a number, a message is generated
;;; explaining what the user did wrong, and the loop starts over.  If
;;; the input is acceptable, the procedure echoes it and quits.



;;; Define the target symbol in the global environment.  What it is
;;; defined to is immaterial, since it will be overwritten.

(define target '())

;;; Define a procedure which needs to escape.  Use the target to tell
;;; it where to escape to.

(define (f x)
  (cond ((= x 0)
         (display "0 entered; try again.")
         (newline)
         (target x))   ;;; the argument x will be ignored.
        (else
         (display "Success: ")
         (display x)
         (newline))
        )
  )

;;; Define the calling routine, which in turn defines the target.

(define (main_loop)
  (call/cc
   (lambda(here)
     (set! target here)))
  (display "Type a number different from 0:  ")
  (let ((n (read)))

    ;; First check to make sure that a number was entered.
    (if (not (number? n))
        (begin
          (display n)
          (display " is not a number; try again.")
          (newline)
          (target n)   ;;; the argument n will be ignored.
          )
        )

    ;; OK; a number was entered.  Now call f to do the rest of the
    ;; processing.
    (f n)
    )
  )

;;; The continuation that is passed to "here" can't really be expressed in
  ;;; simple Scheme.  It would be pretty close to this, however (note that we
  ;;; still need to use an "escape" procedure named "exit"):

;; (lambda (val)
;;   (display "Type a number different from 0:  ")
;;   (let ((n (read)))
;;
;;     ;; First check to make sure that a number was entered.
;;     (if (not (number? n))
;;         (begin
;;           (display n)
;;           (display " is not a number; try again.")
;;           (newline)
;;           (target n)   ;;; the argument n will be ignored.
;;           )
;;         )
;;
;;     ;; OK; a number was entered.  Now call f to do the rest of the
;;     ;; processing.
;;     (f n)
;;     )
;;   (exit) ;; well, escape to the top level!
;;   )
```