

```

;; Function To Be Tested: defstruct
;;
;; Source: Common Lisp the Language by Guy Steele
;;         Section 19: Structures, page 305
;;
;; Created By: John Park   Reviewed by Peter Reidy (as a brief test of main features
;; already implemented 7 aug.)
;;
;; Creation Date: Aug 5, 86
;;
;; Last Update:  April 9, 86 (CSW)
;;
;; History:  Added regression tests thru lyric patch 4 (CW)
;;
;; Filed as:      {eris}<lispcore>cml>test>19-defstruct.test
;;
;; Syntax: (defstruct name-and-options [doc-string] {slot-description}+)
;;
;; Function Description: This function defines a record-structure data type.
;; A general call to defstruct looks like the following:
;;
;;         (defstruct (name option-1 option-2 ...)
;;           doc-string
;;           slot-description-1
;;           slot-description-2
;;           ....)
;;
;; Argument(s):
;;
;;         Name: must be a symbol; it becomes the name of a new data type
;;               consisting of all instances of the structure. The name is
;;               returned as the value of the defstruct form.
;;         Doc-String: This is attached to the name as a documentation
;;                     string of type structure.
;;         Slot-description-j: Each slot-description-j is of the form
;;
;;         (slot-name default-init
;;           slot-option-name-1 slot-option-value-1
;;           slot-option-name-2 slot-option-value-2
;;           ..... )
;;
;; Returns: The value of the defstruct form.
;;
;; Constraints/limitations: Defstruct options "initial-offset", and "type"
;; (vector), are not implemented as of Aug 7, 86.
;; Comments:
;;
;; Test Case I (simple-defstruct-test): This test checks for data-type of a created
;; structure, make and copy functions, and resetting of the structure components.
;;
;; Test Case II (slot-option-test): This test determines if defstruct slot options
;; can be specified

;; Test Case III: This test determines if each of the options can be given to
;; defstruct. Options include conc-name, constructor, copier, predicate, include,
;; print-function, type, named, and initial-offset.

(do-test-group ("defstruct-test-setup"
  :before (progn
    (setq ship-test-case-1
      (defstruct ship x-position y-position x-velocity y-velocity mass))
    (setq ship-1 (make-ship))
    (setq ship-2 (make-ship :x-position 10 :y-position 0
                           :x-velocity 54 :y-velocity 99))
    (setq ship-3 (copy-ship ship-2))
    (setq ship-4 (make-ship :x-position 100 :y-position 1))

    (setq *default-ship-mass* 777.0)
    (setq test-case-2 (defstruct new-ship
      (x-position 0.0 :type short-float)
      (y-position 0.0 :type short-float)
      (x-velocity 0 :type fixnum)
      (y-velocity 0 :type fixnum)
      (mass *default-ship-mass* :type short-float
        :read-only t)))
    (setq new-ship-1 (make-new-ship
      :x-position 10.9
      :y-position 222.99
      :x-velocity 50

```

```

                                :y-velocity 100
                                :mass *default-ship-mass*)))))

(do-test "simple-defstruct-test"
  (and (typep ship-1 'ship)
        (ship-p ship-1)
        (eq ship-test-case-1 'ship)
        (eq (ship-x-position ship-2) 10)
        (eq (ship-y-position ship-2) 0)
        (eq (ship-x-velocity ship-2) 54)
        (eq (ship-y-velocity ship-2) 99)
        (eq (ship-mass ship-2) nil)
        (eq (ship-x-position ship-3) 10)
        (eq (ship-y-position ship-3) 0)
        (eq (ship-x-velocity ship-3) 54)
        (eq (ship-y-velocity ship-3) 99)
        (eq (ship-mass ship-3) nil)
        (eq (ship-x-position ship-4) 100)
        (eq (ship-y-position ship-4) 1)
        (eq (ship-x-velocity ship-4) nil)
        (eq (ship-y-velocity ship-4) nil)
        (eq (ship-mass ship-4) nil)
        (setf (ship-x-position ship-3) 0)
        (eq (ship-x-position ship-3) 0))))

(do-test "slot-option-test"
  (and (new-ship-p new-ship-1)
        (typep (new-ship-x-position new-ship-1) 'short-float)
        (typep (new-ship-y-position new-ship-1) 'short-float)
        (typep (new-ship-x-velocity new-ship-1) 'fixnum)
        (typep (new-ship-y-velocity new-ship-1) 'fixnum)
        (typep (new-ship-mass new-ship-1) 'single-float)
        (setf (new-ship-x-position new-ship-1) 100.0)
        (eql (new-ship-mass new-ship-1) *default-ship-mass*)
        (typep (new-ship-y-position new-ship-1) 'short-float))))

(do-test "conc-name-option-test"
  (and (defstruct (employer (:conc-name manager-)) name age sex)
        (setq new-employer (make-employer :name 'smith :age 40 :sex 'm))
        (eq (manager-name new-employer) 'smith)
        (eq (manager-age new-employer) 40)
        (eq (manager-sex new-employer) 'm)))

(do-test "constructor-option-test"
  (and (defstruct auto engine body)
        (fboundp 'make-auto)
        (defstruct (auto (:constructor build-auto)) engine body)
        (fboundp 'build-auto)
        (setq new-auto (build-auto :engine '8cyl :body 'convert))
        (eq (auto-engine new-auto) '8cyl)
        (defstruct (auto (:constructor design-auto)) engine body)
        (fboundp 'design-auto)
        ))

(do-test "copier-option-test"
  (and (defstruct (truck (:copier duplicate-truck)) engine body)
        (setq prototype (make-truck :engine '16cyl :body 'wide))
        (setq new-truck (duplicate-truck prototype))
        (eq (truck-engine new-truck) '16cyl)
        (eq (truck-body new-truck) 'wide)
        (defstruct (sports-car (:copier nil)) engine body)
        (not (fboundp 'copy-sports-car))
        ))

(do-test "predicate-option-test"
  (and (defstruct (tools (:predicate is-tool?)) name size direction)
        (setq tool1 (make-tools))
        (is-tool? tool1)))

(do-test "include-option-test"
  (and (defstruct person name age sex)
        (defstruct (astronaut (:include person)
                              (:conc-name astro-))
          helmet-size
          (favorite-beverage 'tang))
        (setq astro-1 (make-astronaut :name 'buzz
```

```

:age 47
:sex 'm
:helmet-size 17.5))

(eq (person-name astro-1) 'buzz)
(eq (astro-name astro-1) 'buzz)
(eq (astro-age astro-1) 47)
(eq (astro-sex astro-1) 'm)
(equalp (astro-helmet-size astro-1) 17.5)
(eq (astro-favorite-beverage astro-1) 'tang)))

(do-test "print-function-option-test"
  (and (defstruct (numbers (:print-function default-structure-printer))
    x y z)
    (setq number1 (make-numbers :x 100 :y 200 :z 300))
    (eq (numbers-x number1) 100)
    (eq (numbers-y number1) 200)
    (eq (numbers-z number1) 300)
    (numbers-p number1)))

(do-test "type-option-test"
  (and (defstruct (binop (:type list))
    (operator '? :type symbol) operand-1 operand-2)
    (setq binop-1 (make-binop :operator '+'
      :operand-1 'x :operand-2 5))
    (equal binop-1 ' (+ x 5))
    (setq binop-2 (make-binop :operand-2 4 :operator '*))
    (equal binop-2 ' (* nil 4))
    (defstruct (trinop (:type vector)) element1 element2)
    (vectorp (make-trinop :element1 0 :element2 1))))

(do-test "named-option-test"
  (and (defstruct (named-binop (:type list) :named)
    (operator '? :type symbol) operand-1 operand-2)
    (equal (make-named-binop :operator '+'
      :operand-1 'x :operand-2 5)
      ' (named-binop + x 5))
    (equal (make-named-binop :operand-2 4 :operator '*')
      ' (named-binop * nil 4))))

(do-test "initial-offset-option-test"
  (and (defstruct (offset-binop (:type list) (:initial-offset 2))
    (operator '? :type symbol) operand-1 operand-2)
    (setq offset-binop-1 (make-offset-binop :operator '+'
      :operand-1 'x :operand-2 5))
    (equal offset-binop-1 ' (NIL NIL + X 5))
    (defstruct (offset-binop2 (:type list) :named (:initial-offset 2))
      (operator '? :type symbol) operand-1 operand-2)
    (setq offset-binop-3 (make-offset-binop2 :operator '+'
      :operand-1 'x :operand-2 5))
    (equal offset-binop-3 ' (NIL NIL OFFSET-BINOP2 + X 5))))

;;
;; Regression tests
(do-test "AR 7650 Regression test"
  (and (defstruct (foo (:type (vector fixnum))) s1 (s2 2) s3))
    (setq s (make-foo :s1 1))
    (eq (foo-s1 s) 1)))
)

STOP

```