UO-LISP NEWSLETTER

October 1985 Vol. 2 No. 4

Version 3.2 Learn Lisp Announced

Version 3.2 of the UO-LISP Learn Lisp system is now available from Northwest Computer Algorithms. This newest version contains many long awaited features:

1. **Generic Arithmetic.** Bignumber arithmetic is now automatic when the small integer magnitude is exceeded. Small number arithmetic is accomplished with the usual functions, but each is prefixed with an I: IADD1, IPLUS2, etc.. Bignumber-only arithmetic is accomplished by the same functions prefixed with a B: BADD1, BPLUS2, etc.. On the other hand, the PLUS2 function both verifies numeric operands are present and decides whether IPLUS2, BPLUS2, or FPLUS2 (for floating point) is to be called.

2. **Floating Point.** 7 digits of accuracy, arbitrary exponents. Floating point numbers are entered in the usual format. Output is in standard notation for reasonable sized numbers and scientific notation when a fixed exponent range is exceeded. The generic arithmetic package converts fixed operands to floating point whenever at least on argument is floating.

3. **Space Sizes** Dotted-pair space has been expanded to 16384 possible pairs. String space has expanded to accomodate about 2000 strings with an aggregate length of 32k bytes.

4. **New Functions.** Several new functions have been added: TRINTS: turn on saving the name of interpreted functions on the alist for traceback. UNTRINTS Turn off the interpeted function tracing. CLOSEALL Close all open files. MAPOBL Apply a function to every identifier in the symbol table.

Blackjack

In place of the promised music generation program, we offer this program submitted by Andrew Parker. The program is divided into two parts, the first is a general program for shuffling a deck

of cards. The second plays the game. We present this pretty much as we received it.

```
%*****************************************************
%
%  This program shuffles a deck of cards. The shuffled
%  deck is returned as a list. The deck of cards is
%  maintained in the global variable "deck". The deck
%  is a standard 52 card deck with no JOKERs. The user
%  can modify "deck" by adding the additional elements
%  required. For example to add two JOKERs you would do
%  the following:
%
%      (SETQ deck (APPEND deck '(JOKER JOKER)))
%
%  Two decks can be shuffled together by APPENDING the
%  global variable "deck" to itself.


% create  the deck of cards
(GLOBAL '(deck))

(SETQ deck '(!2H !3H !4H !5H !6H !7H !8H !9H !1!0H JH
             QH KH AH !2D !3D !4D !5D !6D !7D !8D !9D
             !1!0D JD QD KD AD !2C !3C !4C !5C !6C !7C
             !8C !9C !1!0C JC QC KC AC !2S !3S !4S !5S
             !6S !7S !8S !9S !1!0S JS QS KS AS))

%*****************************************************
%  remove!-nth - removes the nth element of a list.
%
%  ARGUMENTS:  l -  a list (the deck of cards)
%              element!-number - the position of the
%                   nth element
%
%  RETURNS:    the list l with the nth element removed
%
%  SIDE EFFECTS: none

(DE remove!-nth (l element!-number)
   (COND ((NULL l) NIL)
         ((EQ element!-number 1) (CDR l))
         (T (CONS (CAR l)
                  (remove!-nth (CDR l)
                               (SUB1 element!-number))))))

%*****************************************************
% return!-nth: returns the nth element of the list l as
% an atom.
%
% ARGUMENTS:  l - a list (the deck of cards)
%             element!-number: the position of the
%                 nth element of the list
%
% RETURNS:    the nth element of the list l as an atom
```

```
%
% SIDE EFFECTS: none

(DE return!-nth (l element!-number)
   (COND ((NULL l) NIL)
         ((EQ element!-number 1) (CAR l))
         (T (return!-nth (CDR l)
                         (SUB1 element!-number)))))

%*******************************************************
%  shuffle
%
% ARGUMENTS: l - a list (the deck of cards)
%
% RETURNS:   the list l randomly rearranged.
%
% SIDE EFFECTS: none

(DE shuffle (l)
  (PROG (card!-position length!-l)
    (COND ((NULL l) (RETURN NIL))
          ((EQN 1 (SETQ length!-l (LENGTH l)))
           (SETQ card!-position 1))
          (T (SETQ card!-position
              (ADD1 (QUOTIENT (RANDOM)
                              (QUOTIENT 8191
                                        length!-l))))))
    (RETURN (CONS (return!-nth l card!-position)
                  (shuffle
                   (remove!-nth l card!-position))))))

%*******************************************************
%  get!-deck
%
%  ARGUMENTS:  none
%
%  RETURNS:    a list that represents a shuffled deck
%              of cards
%
%  SIDE EFFECTS: reassigns the global variable "deck"
%              to the order

(DE get!-deck ()
  (SETQ deck (shuffle deck)))

STOP
```

The second program section plays the game. There are a few machine dependent parameters at the beginning that must be fiddled for non-IBM compatible machines (this worked on the Tandy Model 2000 without change).

```
(COND ((NOT (GETD 'CURSOR)) (FLOAD "TERMINAL")))


(GLOBAL '(
 my!-deck            % deck of cards to be delt
 dimond              % diamonds
 club                % clubs
 heart               % hearts
 spade               % spades
 joker               % joker
 upper!-left         % upper left corner of card
 upper!-right        % upper right corner of card
 lower!-left         % lower left corner of card
 lower!-right        % lower right corner of card
 top!-edge           % top edge of card
 side!-edge          % side edge of card
 card!-back          % back of card
 your!-current!-bet % amount you are betting
 your!-hand          % A-list representing your hand
 your!-total         % amount of money you have left
 computer!-hand      % A-list of the computers hand
 double!-hand))      % A-list of your double hand

%
% The following values must be set for the type of
% computer you are using. The default is for the IBM PC
% and the COMPAQ deskpro.

(SETQ dimond 4)
(SETQ club   5)
(SETQ heart  3)
(SETQ spade  6)
(SETQ joker  2)
(SETQ upper!-left  218)
(SETQ upper!-right 191)
(SETQ lower!-left  192)
(SETQ lower!-right 217)
(SETQ top!-edge    196)
(SETQ side!-edge   179)
(SETQ card!-back   178)




%
% your!-hand and the computer!-hand are A-list
% representing the cards that have been dealt. The
% A-list looks as follows:
%
%  ( ((x!-position . y!-position) (card . suit))
%    ((x!-position . y!-position) (card . suit)).....)
%
% There is one entry for each card a player has been
% dealt. At the beginning of each hand the lists are
```

```
% set to NIL and the totals to zero.
%


(SETQ deck '(
 1  2  3  4  5  6  7  8  9 10 11 12 13     % hearts
 14 15 16 17 18 19 20 21 22 23 24 25 26    % diamonds
 27 28 29 30 31 32 33 34 35 36 37 38 39    % clubs
 40 41 42 43 44 45 46 47 48 49 50 51 52))  % spades

%*****************************************************
% get!-normal!-card -
%
% ARGUMENTS - card!-number: numeric value representing
%                           one playing card.  If the
%                           number is greater then 52
%                           it is assumed to be a joker
%
% RETURNS - a dotted pair in which the CAR is the
%           card value 2-10, J-A and the CDR is the
%           suit 3-hearts, 4-dimonds, 5-clubs,
%           and 6-spades.

(DE get!-normal!-card (card!-number)
  (PROG (suit card)
     (COND ((LESSP card!-number 14)
                (SETQ suit heart)
                (SETQ card card!-number))
           ((LESSP card!-number 27)
                (SETQ suit dimond)
                (SETQ card (DIFFERENCE card!-number 13)))
           ((LESSP card!-number 40)
                (SETQ suit club)
                (SETQ card (DIFFERENCE card!-number 26)))
           ((LESSP card!-number 53)
                (SETQ suit spade)
                (SETQ card (DIFFERENCE card!-number 39)))
           (T   (SETQ suit joker)
                (SETQ card "J") ))
     (COND ((EQ card 11) (SETQ card "J"))
           ((EQ card 12) (SETQ card "Q"))
           ((EQ card 13) (SETQ card "K"))
           ((EQ card 1)  (SETQ card "A")))
     (RETURN (CONS card suit)) ))

%*****************************************************
%
%                      OUTPUT SECTION
%
%*****************************************************
% print!-card!-id - displays the card value 2-10, J-A
% and the graphic representation of the suit.

(DE  print!-card!-id (card!-frame)
    (TERPRI)
```

```
    (CURSOR (PLUS (CAAR card!-frame) 4)
            (DIFFERENCE (CDAR card!-frame) 2))
    (PRIN2 (CADR card!-frame))
    (!$PA (CDDR card!-frame)) )

%*******************************************************
% print!-card!-back - if the card is delt face down,
% fill in the back

(DE print!-card!-back (card!-frame)
  (PROG (x y)
    (TERPRI)
    (SETQ x (ADD1 (CAAR card!-frame)))
    (SETQ y (CDAR card!-frame))
    (FOR (FROM I 1 6)
      (DO
        (CURSOR x (DIFFERENCE y I))
        (FOR
          (FROM counter 1 8)
           (DO (!$PA card!-back) )) )) ))

%*******************************************************
%  print!-card - displays one card on the screen. The
% x,y position is the upper left hand corner of the card.
%
% ARGUMENTS: card!-frame - an A-list representing the
%                          card and its position
%                      ((x!-position . y!-position)
%                       (card!-value . suit))
(DE print!-card (card!-frame)
  (PROG (x y card)
    (SETQ x (CAAR card!-frame))
    (SETQ y (CDAR card!-frame))
    (SETQ card (CDR card!-frame))
    (TERPRI)
    (CURSOR x y)
    (!$PA upper!-left)
    (FOR (FROM I 1 8) (DO (!$PA top!-edge)))
    (!$PA upper!-right)
    (FOR (FROM I 1 6)
      (DO
        (CURSOR x (DIFFERENCE y I))
        (!$PA side!-edge)
        (PRIN2 "        ")
        (!$PA side!-edge) ))
    (CURSOR x (DIFFERENCE y 7))
    (!$PA lower!-left)
    (FOR (FROM I 1 8) (DO (!$PA top!-edge)))
    (!$PA lower!-right) ))

%*******************************************************
% erase!-card - removes a card from the screen. It does
% not repaint any underlying cards.

(DE erase!-card (card!-frame)
```

```
  (PROG (x y)
    (SETQ x (CAAR card!-frame))
    (SETQ y (CDAR card!-frame))
    (FOR (FROM i 0 7)
      (DO
        (TERPRI)
        (CURSOR x (DIFFERENCE y i))
        (PRIN2 "          ") )) ))

%*******************************************************
% redraw!-hand - re-paints a hand on the screen.

(DE redraw!-hand (hand)
   (COND ((NOT (NULL hand))
           (print!-card (CAR hand))
           (print!-card!-id (CAR hand))
           (redraw!-hand (CDR hand)) )) )

%*******************************************************
% get!-card - get the first card off the deck
% The card returned is an A-list (card value .  suit)

(DE get!-card ()
 (PROG (card)
   (COND ((NULL my!-deck)
          (CURSOR 1  2)
          (CLEAR!-EOP)
          (PRIN2T "I'm shuffling a new deck, please wait")
          (SETQ my!-deck (get!-deck)) ))
     (SETQ card (get!-normal!-card (CAR my!-deck)))
     (SETQ my!-deck (CDR my!-deck))
     (RETURN card) ))

%*******************************************************
% card!-position
% ARGUMENTS hand -  A-list  representing one hand
% RETURNS: a dotted pair contianing the x!-position and
%          y!-position to display the next card

(DE card!-position (hand)
  (COND ((LESSP (DIFFERENCE (CDAAR hand) 4) 14)
           (CONS (PLUS2 (CAAAR hand) 8) 22))
        (T (CONS (CAAAR hand)
                 (DIFFERENCE (CDAAR hand) 4)))))

%*******************************************************
%
%                          BLACK JACK
%
%*******************************************************
% deal - deals out the first two cards for black jack.

(DE deal ()
  (PROG (card)
    (CLEAR)
```

```
    (TERPRI)
    (CURSOR 5 24)
    (PRIN2T "The House")
    (CURSOR 58  24)
    (PRIN2T "Your Hand")
    (SETQ computer!-hand
           (SETQ double!-hand
                  (SETQ your!-hand NIL)))
    (FOR (FROM i 1 2)
      (DO
       (SETQ your!-hand
          (APPEND
           (LIST
            (CONS (CONS 58
                        (DIFFERENCE 26 (TIMES i 4)))
                  (get!-card)))
           your!-hand))
       (print!-card (CAR your!-hand))
       (print!-card!-id (CAR your!-hand))
       (SETQ computer!-hand
          (APPEND
           (LIST
            (CONS (CONS 5
                        (DIFFERENCE 26 (TIMES i 4)))
                  (get!-card)))
           computer!-hand))
       (print!-card (CAR computer!-hand))
       (COND ((NEQ i 1)
              (print!-card!-id (CAR computer!-hand)))
             (T (print!-card!-back
                 (CAR computer!-hand)))))))


%*******************************************************
% total!-cards
% ARGUMENTS: hand - A-list containing all the cards in
%        a hand.
% RETURNS: a numeric value representing the total of
%        the hand.

(DE total!-cards (hand)
  (PROG (aces total card!-value)
    (SETQ total (SETQ aces 0))
    (WHILE (NOT (NULL hand))
     (DO
      (COND ((NUMBERP (CADAR hand))
             (SETQ card!-value (CADAR hand)))
            ((EQ (CADAR hand) "A")
             (SETQ card!-value 11)
             (SETQ aces (ADD1 aces)))
            (T (SETQ card!-value 10) ))
      (SETQ total (PLUS total card!-value))
      (SETQ hand (CDR hand)) ))
    (WHILE (AND (GREATERP total 21)
                (GREATERP aces 0))
```

```
      (DO
        (SETQ total (DIFFERENCE total 10))
        (SETQ aces (SUB1 aces)) ))
    (RETURN total) ))

%*******************************************************
% deal!-your!-cards
% ARGUMENTS: your!-hand - A-list representing a players
%     hand

(DE deal!-your!-cards ()
  (PROG (command ender)
    (SETQ ender (SETQ command 'H))
    (SETQ PROMPT!* "(H)it, (S)tay, (D)double, (Q)uit: ")
    (WHILE (AND (NOT (MEMQ command '(!S !s)))
                (LESSP (total!-cards your!-hand) 21)
                (NOT (MEMQ command '(!Q !q))) )
     (DO
      (TERPRI)
      (CURSOR 1 2)
      (CLEAR!-EOL)
      (SETQ command (READ))
      (COND ((MEMQ command '(!H !h))
             (SETQ your!-hand (deal!-card your!-hand)))
            ((AND (MEMQ command '(!D !d))
                  (EQ (LENGTH your!-hand) 2)
                  (EQ (CADAR your!-hand)
                      (CADADR your!-hand)))
                    (do!-double)
                    (SETQ command 'S) )) ))
    (CURSOR 1 2)
    (CLEAR!-EOL)
    (RETURN command) ))

%*******************************************************
% deal!-card
% ARGUMENTS  hand - A-list representing one hand
% RETURNS    an A-list with the additional card in it.

(DE deal!-card (hand)
 (PROG ()
    (SETQ hand (APPEND (LIST (CONS (card!-position hand)
                                   (get!-card)))
                       hand))
    (print!-card (CAR hand))
    (print!-card!-id (CAR hand))
    (RETURN hand) ))

%*******************************************************
% do!-double - control function for when you double.
% The computer hand will never double.
%
% ARUGMENTS - none we will always double your!-hand

(DE do!-double ()
```

```
  (PROG (command)
    (SETQ double!-hand (LIST (CAR your!-hand)))
    (SETQ your!-hand (CDR your!-hand))
    (erase!-card (CAR double!-hand))
    (print!-card (CAR your!-hand))
    (print!-card!-id (CAR your!-hand))
    (SETQ double!-hand
      (LIST (CONS (CONS
                   (DIFFERENCE (CAAAR double!-hand) 26)
                   (PLUS (CDAAR double!-hand) 4) )
                  (CDAR double!-hand) )))
    (CURSOR (CAAAR double!-hand) 24)
    (PRIN2 "Double Hand")
    (print!-card (CAR double!-hand))
    (print!-card!-id (CAR double!-hand))
    (SETQ your!-hand (deal!-card your!-hand))
    (SETQ double!-hand (deal!-card double!-hand))
    (SETQ PROMPT!* "YOUR HAND - (H)it, (S)tay: ")
    (WHILE (AND (NOT (MEMQ command '(!S !s)))
                (LESSP (total!-cards your!-hand) 21))
     (DO
      (TERPRI)
      (CURSOR 1 2)
      (CLEAR!-EOL)
      (SETQ command (READ))
      (COND ((MEMQ command '(!H !h))
             (SETQ your!-hand
                   deal!-card your!-hand)) )) ))
    (SETQ command 'H)
    (SETQ PROMPT!* "DOUBLE HAND - (H)it, (S)tay: ")
    (WHILE (AND (NOT (MEMQ command '(!S !s)))
                (LESSP (total!-cards double!-hand) 21))
     (DO
      (TERPRI)
      (CURSOR 1 2)
      (CLEAR!-EOL)
      (SETQ command (READ))
      (COND ((MEMQ command '(!H !h))
             (SETQ double!-hand
                   (deal!-card double!-hand)) )) )) ))

%*****************************************************
% who!-won - determines who won the game.  It displays
% the results on the screen and adjusts your!-total
% winnings.
%
% ARGUMENTS:  hand -       A-list representing the hand
%                          to compare against the
%                          computer.
%             hand!-type - discription for the display
%                          `double or `first

(DE who!-won (hand hand!-type)
  (COND ((AND (LEQ (total!-cards hand) 21)
              (LEQ (total!-cards computer!-hand) 21))
```

```
                (COND ((LESSP
                         (total!-cards hand)
                         (ADD1
                          (total!-cards computer!-hand)))
                          (PRIN2 "Computer wins ")
                          (PRIN2 hand!-type)
                          (PRIN2 " hand")
                          (SETQ your!-total
                                (DIFFERENCE
                                  your!-total
                                  your!-current!-bet)))
                      (T (PRIN2 "You win ")
                         (PRIN2 hand!-type)
                         (PRIN2 " hand")
                         (SETQ your!-total
                               (PLUS
                                your!-total
                                your!-current!-bet)))))
        ((LEQ (total!-cards hand) 21)
           (PRIN2 "You win ")
           (PRIN2 hand!-type)
           (PRIN2 " hand")
           (SETQ your!-total
                (PLUS your!-total your!-current!-bet)))
        ((LEQ (total!-cards computer!-hand) 21)
           (PRIN2 "Computer wins ")
           (PRIN2 hand!-type)
           (PRIN2 " hand")
           (SETQ your!-total
                (DIFFERENCE your!-total
                            your!-current!-bet)))
        (T (PRIN2 "We're both over, no winner") )) )

%*******************************************************
% BLACKJACK - main controller loop for playing
%      blackjack.

(DE BLACKJACK ()
   (PROG (command)
      (CLEAR)
      (SETQ your!-total 2000)
      (TERPRI)
      (CURSOR 1 12)
      (PRIN2 "Type any character to start....")
      (WHILE (EQ (DIRECTIO 255) 0)
             (DO (SETQ SEED!* (QUOTIENT (RANDOM) 819))))
      (WHILE (AND (NOT (MEMQ command '(!Q !q)))
                  (GREATERP your!-total 0))
        (DO
          (SETQ your!-current!-bet 0)
          (WHILE (OR (EQ your!-current!-bet 0)
                     (NOT (NUMBERP your!-current!-bet)))
           (DO
            (SETQ PROMPT!* " ")
            (TERPRI)
```

```
        (CURSOR 1 2)
        (CLEAR!-EOP)
        (PRIN2
  "Please enter your bet, you can wager up to $")
        (PRIN2 your!-total)
        (PRIN2 " :")
        (SETQ your!-current!-bet (READ)) ))
      (deal)
      (COND ((NOT (MEMQ
                   (SETQ command (deal!-your!-cards))
                   '(!Q !q)))
        (redraw!-hand (REVERSE computer!-hand))
        (COND ((OR (LEQ (total!-cards your!-hand) 21)
                   (AND double!-hand
                        (LEQ
                         (total!-cards double!-hand)
                         21)))
                 (WHILE (LEQ
                        (total!-cards computer!-hand)
                        16)
                  (DO
                   (SETQ computer!-hand
                      (deal!-card computer!-hand))))))
        (CURSOR 1 4)
        (CLEAR!-EOP)
        (who!-won your!-hand 'first)
        (CURSOR 1 3)
        (CLEAR!-EOP)
        (COND (double!-hand
               (who!-won double!-hand 'double))) )) ))
    (CLEAR)
    (TERPRI)
    (CURSOR 1 12)
    (PRIN2 "Your total is $")
    (PRIN2 your!-total)
    (CURSOR 1 2)
    (SETQ PROMPT!* "*") ))
```

The program is executed by simply calling the function BLACKJACK.