# N E W S L E T T E R  # 4

**March 1981**

This is the fourth issue of The Soft Warehouse Newsletter. The newsletters are devoted to bringing our customers up to date on the latest news, updates, and products of The Soft Warehouse. In addition, they provide a medium for the growing community of **muLISP** and **muMATH** users to exchange ideas and application programs. We are always in need of suggestions, gripes, short articles and/or program listings for publication. This newsletter can be as useful as you want to make it. Before we get down to business, several announcements are in order:

## muMATH and muLISP Reviews

Buried in the back of the November 1980 issue of BYTE on page 324 is an excellent review of muSIMP and muMATH-79 by **Gregg Williams** of BYTE. He gives a very lucid explanation of the capabilities of the system. Since the article is not listed in the table of contents you may have missed it. In the forthcoming April 1981 issue there will be a comparative review of muLISP and some of our competitors' LISPs. It should prove to be an interesting comparison and give LISP some badly needed press. Also of interest is a review of the TRS-80 version of muMATH on page 81 of the October 1980 issue of SoftSide magazine.

## SYMSAC '81
## ACM Symposium on Symbolic and Algebraic Computation

**SIGSAM** (ACM's Special Interest Group on Symbolic and Algebraic Manipulation) is sponsoring a major conference this summer. Topics include i) design and analysis of algorithms, ii) languages, systems, and machine architectures, iii) algebraic, elementary, and transcendental function computations, iv) computational number, group, and ring theory, and v) the interface of numeric and symbolic computation methods. It will be held from August 5 to 7, 1981 in Snowbird, Utah, U.S.A. For more information write the General Chairman, **B. F. Caviness**, Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY, 12181, U.S.A.

## IJCAI '81
## International Joint Conference on Artificial Intelligence

The seventh IJACI Conference will be held from August 24 to 28 at the University of British Columbia in Vancouver, B.C., Canada. Papers covering all areas of interest in AI are planned. General topics include i) applications of AI, ii) vision processing, iii) problem solving, iv) expert systems, and v) natural language. In

parallel with the conference will be an R&D exhibition where the latest AI hardware and software will be demonstrated. The Soft Warehouse hopes to be there with some of our new products including an even higher performance implementation of muLISP and muMATH on a 16-bit microprocessor. For information you can write **Pat Hayes**, General Chairman, IJCAI-81, Computer Science Dept., University of Rochester, Mathematical Sciences Bldg., Rochester, NY, 14627, U.S.A.

### Newsletter Subscription Policy

Generally, copies of The SWH Newsletter included with shipment of a system (if any) are not charged against the three (3) free copies promised in the SWH License Agreement. Unless you have renewed your subscription, this is the last free newsletter for owners of systems with SWH serial numbers ending in 137 through 216. Those with serial numbers ending in 217 through 274 will receive one more issue. Finally, those with serial numbers ending in 275 through 356 will receive two more issues. If you would like to subscribe or extend your subscription for another three issues, please send $5 (US) cash or check to The Soft Warehouse, P.O. Box 11174, Honolulu, Hawaii, 96828, U.S.A.

## * * * * * * * * * * * **The muMATHematician** * * * * * * * * * * *

### Handy muSIMP Utility Functions

Often in muMATH when you have gone astray it is necessary to restore a number of variables to their original unbound status in order to begin anew. Or you may be running out of available memory space and it is necessary to delete some old expressions "hanging around" as forgotten variable values. Remembering and then deleting these values will be greatly facilitated by the functions BOUND and CLEAR which are defined in the file CLEAR.MUS. The basic idea for the functions was contributed by **Pierre Schwob** of PRS Corporation (Program Research & Software), NY, NY.

```
% File:     CLEAR.MUS               02/25/81    The Soft Warehouse %

% Function BOUND returns a list of all the bound variables (i.e.
muSIMP names which have a value other than themselves) in the
current oblist if the value of FLAG is FALSE.  If FLAG is NON-FALSE,
the value of bound variables is displayed instead. %

FUNCTION BOUND (FLAG,
    %Local:% LST1, LST2),
  LST1: OBLIST (),
  LOOP
    WHEN EMPTY (LST1),
      WHEN FLAG, NEWLINE () EXIT,
      REVERSE (LST2) EXIT,
    BLOCK
      WHEN FIRST(LST1) EQ EVAL(FIRST(LST1)) EXIT,
      WHEN FIRST(LST1) EQ 'LST1 EXIT,
      WHEN FIRST(LST1) EQ 'LST2 EXIT,
      WHEN FIRST(LST1) EQ 'FLAG EXIT,
      BLOCK
      WHEN FLAG,
        NEWLINE (), PRINT (FIRST(LST1)), SPACES (10-SPACES()),
        PRINT (" = "),
        PRTMATH (EVAL(FIRST(LST1)), 0, 0, TRUE) EXIT ENDBLOCK,
      PUSH (FIRST(LST1), LST2),
    ENDBLOCK,
    POP (LST1),
  ENDLOOP,
ENDFUN $

% Function CLEAR (LST) is used to free or unbind all bound
variables in the list LST.  It can be used in conjunction with
the function BOUND to restore all user defined variables to
their original unbound status. %

FUNCTION CLEAR (LST),
  LOOP
    WHEN EMPTY (LST) EXIT,
    ASSIGN (FIRST(LST), POP(LST)),
  ENDLOOP,
ENDFUN $

RDS () $
```

## A FOR-loop Construct for muSIMP

**T. B. Robinson** of Research Machines, Oxford, England has done it again! This time he has written an extension to muSIMP which correctly parses the FOR-loop construct found in more conventional computer languages. The following is his description:

A FOR-loop begins with the matchfix operator **FOR** and ends with the delimiter **ENDFOR**. An exit from the loop can be made at any time using a WHEN...EXIT construct. The keywords **FROM**, **TO**, and **BY** can be included after the FOR (in any order). Any or all can be omitted. FROM and BY default to 1; TO effectively defaults to infinity by omitting the loop termination test. The BY and TO elements are evaluated each time round the loop as in Algol. Note that commas are required after the FROM, TO, and BY keyword operands.

The value returned by a FOR-loop strongly depends upon what caused the exit from the loop. Normally exit occurs when the FOR-loop variable meets or exceedes its terminating value. In this case the value of the variable which actually caused the termination is the value returned by the FOR-loop. However, if exit occurred early due to a WHEN...EXIT construct, the value of that construct is the value of the FOR-LOOP. The following are some functions showing the use of various permutations of the FOR-loop statement:

```
FUNCTION FORTEST (),
  FOR I FROM 1, TO 7, BY 2,
    PRINT (I),  SPACES (2),
  ENDFOR,
ENDFUN$
```

```
FUNCTION FORTEST0 (),
  FOR I TO 7,
    PRINT (I),  SPACES (2),
  ENDFOR,
ENDFUN$
```

WHILE <condition>, UNLESS <condition>, and UNTIL <condition> can be used as additional loop termination conditions. WHILE is equivalent to UNLESS NOT with the test being performed at the beginning of the loop. The UNTIL test is performed at the end of the loop.

```
FUNCTION FORTEST1 (),
  FOR I BY 2, WHILE I*I<100,
    PRINT (I),  SPACES (2),
  ENDFOR,
ENDFUN$
```

```
FUNCTION FORTEST2 (),
  FOR I FROM 1, TO 20, BY I,
    PRINT (I),  SPACES (2),
  ENDFOR,
ENDFUN$
```

```
FUNCTION FORTEST3 (),
  FOR I TO -20, BY -1,
    PRINT (I),  SPACES (2),
  ENDFOR,
ENDFUN$
```

```
FUNCTION FORTEST4 (A),
  FOR I TO A,
    PRINT (I),  SPACES (2),
  ENDFOR,
ENDFUN$
```

```
FUNCTION FORTEST5 (A),
  FOR I TO A, UNLESS A>10,
    PRINT (I),  SPACES (2),
  ENDFOR,
ENDFUN$
```

```
FUNCTION FORTEST6 (),
  FOR I UNTIL I*I>100,
    PRINT (I),  SPACES (2),
  ENDFOR,
ENDFUN$
```

```
% File:  FORLOOP.MUS          06/22/84           T. B. Robinson %

% Modified by The Soft Warehouse for compatibility with muSIMP-83 %


MOVD ('APPLY, 'FORLOOP)$

DELIMITER: ADJOIN ('ENDFOR, DELIMITER)$

FUNCTION SIGN (EX1),
  WHEN NEGATIVE(EX1), -1 EXIT,
  WHEN ZERO (EX1), 0 EXIT,
  1,
ENDFUN$

FUNCTION FORPARSE (VAR,
    % Local: % FROM, TO, BY, UNTIL, BODY, ARGLIST, ARGS),
  FROM: BY :1,
  BODY: MATCH (SCAN (), 'ENDFOR),
  BLOCK
    WHEN TO,
      BODY: ADJOIN (LIST (LIST ('>, LIST ('*, LIST ('-,
      VAR, '(EVAL TO)), '(SIGN (EVAL BY))), 0), VAR), BODY),
      ARGLIST: LIST (VAR, 'TO, 'BY),
      WHEN INTEGER (TO),
      WHEN INTEGER (BY),
        ARGS: LIST (FROM, TO, BY) EXIT,
      ARGS: LIST (FROM, TO, LIST ('', BY)) EXIT,
      WHEN INTEGER (BY),
        ARGS: LIST (FROM, LIST ('', TO), BY) EXIT,
      ARGS: LIST (FROM, LIST ('', TO), LIST ('', BY)) EXIT,
    ARGLIST: LIST (VAR, 'BY),
    WHEN INTEGER (BY),
      ARGS: LIST (FROM, BY) EXIT,
    ARGS: LIST (FROM, LIST ('', BY)),
  ENDBLOCK,
  BLOCK
    WHEN UNTIL,
      BODY: CONCATEN (BODY, UNTIL) EXIT,
  ENDBLOCK,
  BODY: LIST ('FUNCTION, ARGLIST, ADJOIN ('LOOP, CONCATEN (BODY,
      LIST (LIST (':, VAR, LIST ('+, VAR,  '(EVAL BY)))))))),
  BODY: LIST ('FORLOOP, LIST ('', BODY), ARGS),
ENDFUN$

FUNCTION ENDFORPARSE(),
  BLOCK
    WHEN SCAN EQ COMMA,
      SCAN () EXIT,
  ENDBLOCK,
  WHEN SCAN EQ 'ENDFOR, FALSE EXIT,
  PARSE (SCAN, 0),
ENDFUN$
```

```
FUNCTION FROMPARSE (),
  FROM: PARSE (SCAN (), 0),
  ENDFORPARSE (),
ENDFUN$

FUNCTION TOPARSE (),
  TO: PARSE (SCAN (), 0),
  ENDFORPARSE (),
ENDFUN$

FUNCTION BYPARSE (),
  BY: PARSE (SCAN (), 0),
  ENDFORPARSE ()
ENDFUN$

FUNCTION MAKEFUN (LEX1),
  WHEN ATOM (LEX1),
    LIST ('IDENTITY, LEX1) EXIT,
  LEX1,
ENDFUN$

FUNCTION UNTILPARSE (),
  UNTIL: ADJOIN (LIST (MAKEFUN (PARSE (SCAN (), 0)), VAR), UNTIL),
  ENDFORPARSE (),
ENDFUN$

PROPERTY PREFIX, FOR,
  FORPARSE (SCAN ())$

PROPERTY PREFIX, FROM,
  FROMPARSE ()$

PROPERTY PREFIX, TO,
  TOPARSE ()$

PROPERTY PREFIX, BY,
  BYPARSE ()$

PROPERTY PREFIX, WHILE,
  LIST (LIST ('NOT, PARSE (SCAN (), 0)), VAR)$

PROPERTY PREFIX, UNLESS,
  LIST (MAKEFUN (PARSE (SCAN (), 0)), VAR)$

PROPERTY PREFIX, UNTIL,
  UNTILPARSE ()$

RDS ()$
```

## Bug in the Matrix Package

A bug was found in the matrix division package. It caused strange things to happen when the "\" operator was used to perform matrix division. It affects only versions 1/14/80, 8/14/80 and 8/23/80 of MATRIX.ARR. Do **not** alter a 12/27/79 or earlier version of the file. On the eighth line of the function STARTBACK (which begins around line 99 of the file) change the call to function "APPLY" to "ADJOIN". Then change the date of the file to 10/06/80.

## Bugs in the Integration Package

Three errors have turned up in the muMATH source file INT.DIF. The errors can easily be fixed using any CP/M compatible text editor. Once corrected the version date given in the first line of the file should be changed to: 11/26/80.

1. Bug 1 occurs when attempting to integrate expressions involving the LOG of a LOG (e.g. INT(LOG(LOG(X)),X). The fix is needed for all versions of INT.DIF dated 8/25/80 and earlier. Change the definition of the following property which occurs around line 112 in the file to:

```
PROPERTY INT, LOG, FUNCTION (EX2, EX3),
  WHEN EX3 EQ #E,
  WHEN FREE (EX3:DIF(EX2,INDET), INDET),
    EX2 * (LN(EX2)-1)/EX3  EXIT EXIT,          % Changed Line %
ENDFUN $
```

2. Bug 2 occurs when constant factors can be removed from an expression but the remaining expression still cannot be integrated (e.g. INT(B*F(X),X)). The fix is needed for all versions of INT.DIF dated 10/30/80 and earlier. Change the next-to-last line of the definition of the function INT1 (which begins on line 76 of the file) to:

```
    EX1: EVAL(EX1),
    WHEN EX3:INT2(), TRGEXPD:-7, EX2*EVAL(EX3)  EXIT,
    EX2 * LIST ('INT, EX1, INDET),              % Changed Line %
 ENDFUN $
```

3. Bug 3 occurs when integrating an expression which is a product divided by a constant (e.g. INT(A*X/3,X)). The fix is needed **only** for the 8/25/80 and 10/30/80 versions of INT.DIF; do **not** change a 7/16/79 version. Change the seventh line of the definition of DRVDIV (which begins on line 43 of the file) to:

```
FUNCTION DRVDIV (LEX1),
  % Fluid vars from INT & INT1: INDET, EX1, EX2, EX3, EX4, EX5 %
  WHEN EMPTY (LEX1),  INT3()  EXIT,
  WHEN (EX4:POP(LEX1)) = INDET,  DRVDIV (LEX1)  EXIT,
  EX5: EX1 / EX4,
  WHEN ZERO (EX3:DIF(EX4,INDET)),
    EX2: EX2*EX4,  EX1: EX5,  DRVDIV (LEX1)  EXIT,   % Changed %
```

## A Note to Users of muLISP-79 and muSIMP-79

An extremely useful pair of functions added to muLISP-80 and muSIMP-80 are the PUSH and POP pair of "stack functions". These provide the analogues in muLISP and muSIMP of the stack operators commonly found in machine languages. They are explained in the new muLISP manual as follows:

> POP - if X is the name of a list, then POP [X] returns the car of that list while setting X to the cdr of the list.
>
> PUSH - if Y is the name of a list and X is an expression, then PUSH [X, Y] will cons X onto the list Y and update Y to point to this enlarged list.

They provide a convenient yet structured extension of muLISP especially when used with the LOOP construct. For example see the definition of MKTOWER in the Tower of Hanoi functions given below.

Unfortunately, however, they were not included in muLISP-79. But have no fear, LISP is an extensible language and as such it is easy to define the "stack functions" in terms of more conventional functions. The only cost in the user-defined stack functions is a loss in execution efficiency. muSIMP-79 users can define analogous functions in muSIMP. The muLISP definitions are as follows:

```
(DEFUN POP (NLAMBDA (NAM)
  (POP1 NAM (EVAL NAM)) ))

(DEFUN POP1 (LAMBDA (NAM LST)
  (SET NAM (CDR LST))
  (CAR LST) ))

(DEFUN PUSH (NLAMBDA (EXPN NAM)
  (SET NAM (CONS (EVAL EXPN) (EVAL NAM))) ))
```

## Dazzler Robot for Blocks World

The Blocks World was created as the domain of discourse for Terry Winograd's thesis Understanding of Natural Language [1972]. The Blocks World has become the basis for much subsequent work and discussion in the field of AI. **John Dunn** of Time Arts Synthesis, 1905 San Ramon Ave., Mountain View, CA, 94043 has implemented in muLISP the low-level primitives which would be necessary for writing a system similar to Winograd's. muLISP makes machine language calls to the **BDAZ animation package**. It is a general purpose graphics utility package designed for use with Cromemco's Dazzler™ graphics board. Currently John is putting the BDAZ package in the public domain. Contact him for details.

The Dazzler "Soft" Robot is a 2D implementation of a robot arm

that will move blocks and wedges from a shelf to a table and back. The arm is semi-intelligent in that it will move blocks out of the way in order to get to the one it needs, and it won't put a block on a wedge.

The program is intended as a set of tools for AI work, not as a finished product. There is no human interface to speak of, only the "hardware" of the robot arm. Functions can be used as they are, as robot primitives for a higher-level LISP program, or they can be used as a guide for building other "soft" robots.

Hardware required:

A Z-80 system with at least 48K of RAM.
Cromemco Dazzler graphics subsystem.

Software required:

muLISP-80 modified with the overlay supplied with BDAZ
BDAZ animation package for the Dazzler, "BDAZ.COM"
The BDAZ screen files for the table & boxes,
"TABLE.DAZ" & "BOXES.DAZ"
The low-level MuLISP/BDAZ interface program "EXTRA.LIB"
The high-level MuLISP/BDAZ interface program "DAZ.LIB"
CPM or CDOS with the TPA located at the normal 100 hex

## The Tower of Hanoi Problem

This is a popular computer problem that should exercise the recursive nature of any good LISP programmer. The game begins with three pegs (say A, B, and C) with a number of disks stacked on peg A. All the disks are of different diameter with the largest on the bottom of the stack and the smallest on top. The object of the game is to move the disks from peg A to peg B using peg C as a spare. The following rules must be observed: i) only one disk can be moved at a time, ii) no disk can ever be placed on a smaller disk. **Read no further if you want to solve the problem for yourself.**

A muLISP solution to the Tower of Hanoi problem is provided in the listing of HANOI.LIB printed below. The function XFER below embodies the heart of the algorithm. XFER moves NUM disks from peg SOURCE to peg DEST using peg SPARE as a spare. This is done by moving NUM-1 disks to the spare peg, moving the remaining disk to DEST peg, and finally moving the NUM-1 disks to the DEST peg by using the original SOURCE peg as the spare. Naturally the algorithm must be called recursively to move the NUM-1 disks. In fact it requires $2^n$ moves to move n disks.

The program displays the three pegs horizontally with the letters of the alphabet representing the disks, the smallest disk A being on "top". The process is started by calling the function HANOI with a positive integer argument equal to the number of disks desired. For example:

```
(HANOI 5)
```

I highly recommend **not** using a hard copy terminal for this! A muSIMP solution to this same problem was provided by **Pierre Schwob** of PRS Corporation.

```
% File:  HANOI.LIB            03/02/81          The Soft Warehouse %

(DEFUN HANOI (LAMBDA (NUM
    %Local: % A B C TAB1 TAB2)
  (SETQ A (MKTOWER NUM ALPHABET))
  (SETQ TAB1 (PLUS (LENGTH (PACK A)) 4))
  (SETQ TAB2 (TIMES 2 TAB1))
  (PRINTTOWERS)
  (XFER NUM (QUOTE A) (QUOTE B) (QUOTE C))
  " " ))

(DEFUN MKTOWER (LAMBDA (NUM ALPHABET TOWER)
  (LOOP
    ((ZEROP NUM)
      (REVERSE TOWER) )
    (PUSH (POP ALPHABET) TOWER)
    (SETQ NUM (SUB1 NUM)) ) ))

(DEFUN XFER (LAMBDA (NUM SOURCE DEST SPARE)
  ((ZEROP NUM))
  (XFER (SUB1 NUM) SOURCE SPARE DEST)
  (MOVE SOURCE DEST)
  (XFER (SUB1 NUM) SPARE DEST SOURCE) ))

(DEFUN MOVE (LAMBDA (SOURCE DEST)
  (SET DEST (CONS (CAR (EVAL SOURCE)) (EVAL DEST)))
  (SET SOURCE (CDR (EVAL SOURCE)))
  (PRINTTOWERS) ))

(DEFUN PRINTTOWERS (LAMBDA NIL
  (TERPRI)
  (PRINHANOI A)  (TAB TAB1)
  (PRINHANOI B)  (TAB TAB2)
  (PRINHANOI C) ))

(DEFUN PRINHANOI (LAMBDA (LST)
  (LOOP
    ((NULL LST))
    (PRIN1 (POP LST)) ) ))

(DEFUN SUB1 (LAMBDA (NUM)
  (DIFFERENCE NUM 1) ))

(DEFUN TAB (LAMBDA (NUM)
  (SPACES (DIFFERENCE NUM (SPACES))) ))

(SETQ ALPHABET (QUOTE (A B C D E F G H I J K L M N O P Q R S T U V
     W X Y Z)))

(RDS)
```