N E W S L E T T E R # 5

September 1981

Aloha from Hawaii! The Soft Warehouse Newsletter provides you with information on new Soft Warehouse products, and software extensions or corrections to existing products. In addition, the newsletter is a medium for the exchange of ideas and application programs within the growing community of **muLISP** and **muMATH** users.

If you would like to subscribe, or extend your subscription to the Newsletter for three issues beyond the expiration number on your mailing label, please send $6 ($10 for orders from outside the U.S. or Canada) by check, VISA, or Master Card to The Soft Warehouse, P.O. Box 11174, Honolulu, Hawaii, 96828, U.S.A. A complete set of back issues is available on request for $15.

APPLE and TRS-80 Versions of muMATH Released

Full versions of muMATH-80 for APPLE II computers equipped with Microsoft's Z80 Softcard and TRS-80 Model I and Model III computers are now available. muLISP-80 is also available for the APPLE. All provide the functions available in the CP/M versions of the software plus special graphics primitives to take advantage of the graphics hardware on these machines. A "native" APPLE version of muMATH, not requiring a Softcard, will be released within a month or two. As a Microsoft dealer, The Soft Warehouse has resumed direct end user sales of muLISP and muMATH.

muLISP and muMATH Reviews

The semi-monthly microcomputer news magazine InfoWorld published a review of muLISP by Jeff Levinsky in the July 27, 1981 issue. Jeff notes that since it is "written entirely in muLISP, muSTAR is easy to modify." As an example, he mentions the addition of a REDO command to muSTAR. A copy of the extension is printed in The muLISPer section of this Newsletter.

The September 1981 issue of BYTE magazine contains a comparative review of muLISP and TLC LISP from The LISP Company. Also reviewed is Cromemco LISP, which is a variant of TLC LISP. All you muLISP users let us and BYTE know what you think!

A well written review of muMATH-79 is published in ACM SIGPC Notes (Association for Computing Machines, Special Interest Group on Personal Computing), Volume 4, Numbers 1/2, Spring/Summer 1981. Dr. David Shochat of Santa Monica College writes, "When I first started in with muMATH, it was the symbolic math which fascinated me most... What excites me most now is the muSIMP language itself."

Floating Point Input for muMATH

The ability to enter floating numbers is made possible by some simple extensions to the muSIMP parser. The source file FLOAT.ARI establishes "." as an infix operator in order to parse expressions of the form X.Y where X and Y are integers. X and Y are passed to the function DOT which returns a rational equivalent of the number.

Unfortunately, there is a problem with the program. Since muSIMP reads in the number following the decimal point as an integer, leading zeros are ignored. Therefore the input 3.08 is converted to 3.8. **Bear this in mind when using this package.**

```
% File:  FLOAT.ARI  (c)        09/11/81           The Soft Warehouse %

%  This package permits numbers to be entered using floating point
   notation.  It can not be used in conjunction with the MATRIX.ARR
   package, because of the conflict in the use of "." for the
   matrix dot product.  If the number following the decimal point
   has leading zeros, the muSIMP scanner will ignore the zeros and
   cause incorrect results. %

FUNCTION FLOAT (EX1, EX2)
  EX1 + EX2/RADIX()^LENGTH(EX2)
ENDFUN $

PROPERTY LBP, ".", 190 $

PROPERTY INFIX, ., COND (
  WHEN INTEGER (EX1) AND INTEGER (SCAN),
    FLOAT (EX1, SCAN, SCAN()) EXIT,
  WHEN SYNTAX () EXIT) $

RDS () $
```

Floating Point Output for muMATH

By modifying the muSIMP print routine it is possible to display numbers in floating point instead of rational notation. The modification to the function PRTMATH necessary to accomplish this is in the file POINT.ARI. This package takes ARITH.MUS as a prerequisite.

The control variable POINT is used to activate printing of numbers using floating point notation. If POINT is a non-negative integer, a maximum of POINT digits will be used to display numbers. Otherwise, numbers will be displayed using rational notation.

If the default value of POINT is FALSE in your version of muSIMP, this package has already been integrated into the system. Its use is available as soon as ARITH.MUS is loaded.

```
% File: POINT.ARI  (c)         09/11/81         The Soft Warehouse %


POINT: FALSE $

% POINT is used as both a flag to activate printing of numbers in
  floating point notation, and as the limit on the number of digits
  to be printed.  Rational numbers are displayed in floating point
  notation if and only if POINT is a non-negative integer.  This
  package has already been integrated into muSIMP if the default
  value of POINT is FALSE instead of itself. %

MOVD ('PRTMATH, 'PRTMATH1) $

FUNCTION PRTMATH (EX1, RBP, LBP, PRTSPACE,
    %Local:% EX2, EX3),
  WHEN NOT ATOM(EX1) AND NUMBER(EX1)
      AND (POSITIVE(POINT) OR ZERO(POINT)),
    EX2: DEN (EX1),                          % EX2: denominator %
    EX1: NUM (EX1),                          % EX1: numerator %
    PRTSPACE: FALSE,                         % PRTSPACE: sign flag %
    BLOCK
      WHEN NEGATIVE (EX1),
        BLOCK
        WHEN LBP > 130,
          PRTSPACE: TRUE,
          PRINT (LPAR) EXIT,
        ENDBLOCK,
        PRINT ('-),                          % print minus sign %
        EX1: -EX1 EXIT,
    ENDBLOCK,
    EX1: DIVIDE (EX1, EX2),
    PRINT (FIRST (EX1)),                     % print integer part %
    PRINT ('.),                              % print decimal point %
    EX3: POINT,
    LOOP
      WHEN ZERO (EX3) EXIT,
      WHEN ZERO (SECOND(EX1)) EXIT,
      EX1: DIVIDE (TIMES(SECOND(EX1),RADIX()), EX2),
      EX3: DIFFERENCE (EX3, 1),
      PRTDIG (FIRST (EX1)),                  % print a digit %
    ENDLOOP,
    WHEN PRTSPACE, PRINT (RPAR) EXIT EXIT,
  PRTMATH1 (EX1, RBP, LBP, PRTSPACE)
ENDFUN $

FUNCTION PRTDIG (EX1),                         % print one char digit%
  WHEN LENGTH (EX1) EQ 1, PRINT (EX1) EXIT,
  PRINT (SECOND (EXPLODE (EX1))),
ENDFUN $

RDS () $
```

Series Approximation of Natural Logarithms

A prospective muMATH user was dubious about the advertised accuracy of the system. Specifically, he wondered if it was possible to approximate the logarithm of 2 to a high degree of accuracy. The result of this challenge is LOGAPX.ALG. The function LOGAPX is based on the series

$$LN(x) = 2 (z + z^3/3 + z^5/5 + z^7/7 + \ldots)$$

where z = (x-1)/(x+1). This series is valid for positive values of x. For the curious, here are the first 100 digits of LN(2) as computed using LOGAPX:

```
@: 0.693147180559945309417232121458176568075500134360255254120
   680009493393621969694715605863326996418687 5
```

Four minutes were required to compute the 105 terms of the series necessary to obtain the desired accuracy. When displaying the results of this function, you will probably want to use the floating point option described elsewhere in this newsletter.

```
% File:  LOGAPX.ALG  (C)      09/11/81    The Soft Warehouse %

% LOGAPX(x,n) approximates the natural logarithm of x using n terms
  of the series

          LN(x)  =  2 (z + z^3/3 + z^5/5 + z^7/7 + ... )

  where z = (x-1)/(x+1).  This series is valid for positive values
  of x.  The rate of convergence of the series degrades with
  increasing size of x. %

FUNCTION LOGAPX (X, N,
  % Locals: % Z, TOTAL, CTR),
    % X IS A NUMBER > 0 %
    % N IS THE NUMBER OF TERMS %
    % TOTAL IS THE LOCAL ACCUMULATOR %
    % CTR IS THE LOOP COUNTER %
    % Z IS THE CURRENT POWER OF (X-1)/(X+1) %
  TOTAL: CTR: 0,
  Z: (X-1)/(X+1),
  X: Z^2,
  LOOP
    TOTAL: TOTAL + Z/(2*CTR+1),
    WHEN CTR = N,
      2*TOTAL EXIT,
    Z: Z*X,
    CTR: CTR+1,
  ENDLOOP,
ENDFUN$

RDS ()$
```

Bug Found in Array Package

Dr. Lynn Martin of St. John's University noticed a strange bug when using the array package. When the expression SIN ([X,Y]) was entered, muMATH returned a FALSE. This problem is corrected by replacing APPLY with ADJOIN in the third from the last line of the function SIMPU. SIMPU is defined around line 100 of the file ARRAY.ARI. The revised line should read

ADJOIN (POP(EX1), MAPFUN(LOP1,EX1)) EXIT,

After making this change, revise the date given on the first line of ARRAY.ARI to 03/17/81. Naturally, if your copy of the file is dated on or after this date, this bug has already been corrected.

Vector Algebra Package

File VECTOR.ARR provides facilities for simplifying dot and cross products of row vectors or of unbound variables declared to represent nonscalars. ARRAY.ARI is a prerequisite for VECTOR.ARR; however, the Optional Column portion of ARRAY.ARI is not used and can be eliminated to save space. ALGEBRA.ARI should also be loaded if you are simplifying dot and cross products that contain unbound variables.

The dot "." is the vector **dot product** operator. The left and right binding powers of "." are 125. Therefore, it has a higher precedence than multiplication and division, but lower than exponentiation.

The tilde "~" is the vector **cross product** operator. The left and right binding powers of "~" are 133 and 134 respectively. Thus, "~" associates right, meaning a~b~c is equivalent to a~(b~c).

The operands of the dot product and cross product operators can be **row vectors** (see ARRAY.ARI documentation) or unbound variables declared to be nonscalar by commands of the form

NONSCALAR (variable, variable, ...)

Simplifications involving such nonscalar unbound variables treat them as entities, without reference to components in a particular coordinate system. However, expressions containing such unbound nonscalars can be expressed in terms of corresponding unbound components by using the functional form

COMPONENTS (expression)

COORDS, initially ['X, 'Y, 'Z], is used by COMPONENTS to replace unbound nonscalar variables such as A by corresponding row vectors such as [A[X], A[Y], A[Z]].

The control variables **VECEXPD** is initially FALSE. When VECEXPD is TRUE, the transformations

```
b~(c~d)  -->  b.d*c - b.c*d
a*b.c~d  -->  b*a.c~d - c*a.b~d + d*a.b~c
```

are automatically applied for nonscalar expressions a, b, c, and d, with **a** ordering ahead of the others.

For examples of the use of VECTOR.ARR, run the demonstration program given at the end of the file. The demonstration is started by making the assignment **RDS:TRUE$** after the initial part of the file has been read in. The algorithms used in VECTOR.ARR are described in "Symbolic Computer Vector Analysis" by David R. Stoutemyer, (Computers & Mathematics with Applications, Vol. 5, pp. 1-9).

1. In vector algebra, there is customarily no distinction between row and column vectors, and "." is commutative but nonassociative. Thus, files VECTOR.ARR and MATRIX.ARR are incompatible and should not both be loaded into the same environment.

2. As in file ARRAY.ARI, row vectors are treated as having implicit trailing zero elements. File VECTOR.ARR supplements this convention by automatically deleting trailing zero components and by replacing the empty vector, [], by the scalar 0. Consequently, row vector operands need not have the same number of components.

3. For the purposes of cross products, any elements beyond the third component of a row vector are ignored.

```
% File VECTOR.ARR  (c)         06/24/81         The Soft Warehouse %

FUNCTION DELLZ (LEX1),
  WHEN ATOM(LEX1) OR NOT ZERO(FIRST(LEX1)), LEX1 EXIT,
  DELLZ (REST(LEX1)),
ENDFUN $

FUNCTION [, LEX1,
  WHEN ATOM (LEX1: DELLZ (REVERSE(LEX1))), 0 EXIT,
  ADJOIN ('[, REVERSE(LEX1)),
ENDFUN $

PROPERTY RBP, ., 125 $
PROPERTY LBP, ., 125 $

PROPERTY RBP, ~, 133 $
PROPERTY LBP, ~, 134 $

PROPERTY SCALAR, ., TRUE $
PROPERTY SCALAR, DIV, TRUE $

FUNCTION NONSCALAR LEX1,
  LOOP
    WHEN ATOM (LEX1), EXIT,
    PUT (POP(LEX1), 'NONSC, TRUE),
  ENDLOOP,
ENDFUN $

NONSCALAR ('[, '~, 'GRAD, 'CURL) $

FUNCTION NONSC (EX1),
  WHEN NAME (EX1), GET (EX1, 'NONSC) EXIT,
  WHEN ATOM(EX1) OR GET(FIRST(EX1), 'SCALAR), FALSE EXIT,
  WHEN GET (POP(EX1), 'NONSC), EXIT,
  LOOP
    WHEN ATOM(EX1), FALSE EXIT,
    WHEN NONSC (POP(EX1)), EXIT,
  ENDLOOP,
ENDFUN $

FUNCTION HASSC (LEX1,
  % local: % EX2, EX3, EX4),
  EX2: EX3: 1,
  LOOP
    BLOCK
      WHEN NONSC (EX4:POP(LEX1)), EX3: EX3*EX4 EXIT,
      EX2: EX2*EX4,
    ENDBLOCK,
    WHEN ATOM (LEX1),
      WHEN EX2 EQ 1, FALSE EXIT,
      ADJOIN (EX2, EX3) EXIT,
  ENDLOOP,
ENDFUN $

FUNCTION CROSS (EX1),
```

```
  FIRST(EX1) EQ '~,
ENDFUN $

FUNCTION INPROD (LEX1, LEX2,
  % local: % EX1),
  EX1: 0,
  LOOP
    WHEN ATOM(LEX1) OR ATOM(LEX2), EX1 EXIT,
    EX1: EX1 + POP(LEX1) * POP(LEX2),
  ENDLOOP,
ENDFUN $

FUNCTION REORD3 (EX1, LEX1),
  WHEN ORDERED (EX1, FIRST(LEX1)), FALSE EXIT,
  WHEN ORDERED (SECOND(LEX1), EX1), POP(LEX1) . FIRST(LEX1)~EX1 EXIT,
  WHEN EX1=FIRST(LEX1) OR EX1=SECOND(LEX1), 0 EXIT,
  EX1~POP(LEX1) . FIRST(LEX1),
ENDFUN $

FUNCTION . (EX1, EX2,
  % local: % EX3),
  WHEN ROW(EX1) AND ROW(EX2), INPROD (REST(EX1), REST(EX2)) EXIT,
  WHEN SUM(EX1),
    POP(EX1), EX3: 0,
    LOOP
      EX3: EX3 + POP(EX1) . EX2,
      WHEN ATOM(EX1), EX3 EXIT,
    ENDLOOP EXIT,
  WHEN PRODUCT(EX1) AND (EX3:HASSC(REST(EX1))), POP(EX3)*(EX2.EX3) EXIT,
  WHEN PRODUCT(EX2) AND (EX3:HASSC(REST(EX2))), POP(EX3)*(EX1.EX3) EXIT,
  WHEN CROSS(EX2) AND (EX3: REORD3(EX1,REST(EX2))), EX3 EXIT,
  WHEN CROSS(EX1) AND (EX3: REORD3(EX2,REST(EX1))), EX3 EXIT,
  WHEN SUM(EX2) OR ORDERED(EX2,EX1), EX2 . EX1 EXIT,
  WHEN NONSC(EX1) AND NONSC(EX2), LIST ('., EX1, EX2) EXIT,
  EX1 * EX2,
ENDFUN $

VECEXPD: FALSE $

PROPERTY *, ., FUNCTION (EX1, EX2, EX3),
  WHEN VECEXPD AND NONSC(EX1) AND CROSS(EX3) AND ORDERED(EX1,EX2),
    EX2*EX1.SECOND(EX3)~THIRD(EX3) - SECOND(EX3)*EX1.EX2~THIRD(EX3)
      + THIRD(EX3)*EX1.EX2~SECOND(EX3)  EXIT,
ENDFUN $

FUNCTION DOT (EX1),
  FIRST(EX1) EQ '.,
ENDFUN $

FUNCTION ~ (EX1, EX2,
  % local: % EX3),
  WHEN ZERO(EX1) OR ZERO(EX2), 0 EXIT,
  WHEN EX1 = EX2, 0 EXIT,
  WHEN ROW(EX1) AND ROW(EX2),
    APPLY ('[, LIST (EX1[2]*EX2[3] - EX1[3]*EX2[2],
```

```
      EX1[3]*EX2[1]-EX1[1]*EX2[3], EX1[1]*EX2[2]-EX1[2]*EX2[1])) EXIT,
  WHEN SUM (EX1),
    POP(EX1), EX3: 0,
    LOOP
      EX3: EX3 + POP(EX1) ~ EX2,
      WHEN ATOM(EX1), EX3 EXIT,
    ENDLOOP EXIT,
  WHEN SUM(EX2), EX2 ~ -EX1 EXIT,
  WHEN PRODUCT(EX1) AND (EX3:HASSC(REST(EX1))), POP(EX3)*EX3~EX2 EXIT,
  WHEN PRODUCT(EX2) AND (EX3:HASSC(REST(EX2))), POP(EX3)*EX1~EX3 EXIT,
  WHEN DOT(EX2) AND (EX3: REORD3(EX1,REST(EX2))), EX3 EXIT,
  WHEN DOT(EX1) AND (EX3: REORD3(EX2,REST(EX1))), EX3 EXIT,
  WHEN CROSS (EX2) AND VECEXPD,
    EX1.THIRD(EX2) * SECOND(EX2) - EX1.SECOND(EX2) * THIRD(EX2)  EXIT,
  WHEN CROSS(EX1) AND VECEXPD OR ORDERED(EX2,EX1), EX2 ~ -EX1 EXIT,
  LIST ('~, EX1, EX2),
ENDFUN $


% * * * * * * * * *  Optional COMPONENTS package  * * * * * * * * * %

FUNCTION COMP (LEX1),
  % fluid: EX1 %
  WHEN ATOM(LEX1), FALSE EXIT,
  ADJOIN (EX1[POP(LEX1)], COMP (LEX1)),
ENDFUN $

COORDS: ['X, 'Y, 'Z] $

FUNCTION COMPONENTS (EX1),
  % fluid: COORDS %
  WHEN NUMBER(EX1), EX1 EXIT,
  WHEN ATOM (EX1),
    WHEN GET (EX1, 'NONSC), APPLY ('[, COMP (REST(COORDS))) EXIT,
    EX1 EXIT,
  APPLY (POP(EX1), MAPFUN ('COMPONENTS, EX1)),
ENDFUN $

RDS: FALSE $  #ECHO: ECHO $  ECHO: TRUE $ % Demo. %

  % Dot Product: %
  [3, T] . [2, #PI];% Cross Product: %
  [3, T, 5] ~ [2, #PI, 7];% You can try some examples for yourself.
  Enter RDS:TRUE$ when ready to resume demo. %
  RDS:FALSE$ % Declaration of nonscalar unbound variables: %
  NONSCALAR (A, B, C, D)$% Simplification of nonscalar variables: %
  (A+B).C - C.B;A~B.C + B.A~C;% Try some examples for yourself.
  Enter RDS:TRUE$ when ready to resume demo. %
  RDS:FALSE$% Simplifications requiring optional transformations: %
  A ~ (B~C) + A.B * C; VECEXPD; VECEXPD:TRUE$ A ~ (B~C) + A.B * C;
  D*A.B~C - A*B.C~D - B*C.A~D;% Enter RDS:TRUE$ to resume demo.%
  RDS:FALSE$% Expanding nonscalar unbound variables into components: %
  COORDS; COMPONENTS (A~B); ECHO:#ECHO $

RDS() $
```

A REDO Command for muSTAR

(Contributed by **Jeff Levinsky**)

The following illustrates how to add a simple REDO facility to the EVAL LISP command in muSTAR. The same idea could be used to add the facility to the EVAL-QUOTE command as well. The original muSTAR code for the EVAL LISP command is:

```
(PUTQQ E EXECUTIVE (LAMBDA ()
  (LINELENGTH LIN-LEN$)
  (LOOP
    (TERPRI)
    (PRIN1 *)
    ((EQ (PRINT (EVAL (READ))) EXIT)) ) ))
```

To implement a REDO command modify the code to read (the changes to be made are in boldface):

```
(PUTQQ E EXECUTIVE (LAMBDA (COMMAND)
  (LINELENGTH LIN-LEN$)
  (LOOP
    (TERPRI)  (PRIN1 *)
    (SETQ COMMAND (READ))
    ((EQ (PRINT (EVAL (COND
        ((EQ COMMAND (QUOTE REDO)) OLDCOMMAND)
        ((SETQ OLDCOMMAND COMMAND)) ))) EXIT)) ) ))
```

A Fractional Factorial Experiment Design Program
(Contributed by **David Dunthorn**)

To explain a bit about what the program is supposed to do, suppose that you have a process which is a function of 4 factors which can be controlled. For a specific choice of each of the four factors, the process produces a value, and you wish to determine the influence of each factor on that value. Now, a two-level experiment design requires that you select what you consider to be two distinct levels for each factor; on-off for a switch, high-low for a temperature, present-absent for a chemical additive, etc.

A two-level factorial experiment for the above would use 16 (=2^4) experiments to run every possible combination of factors. The results could be analysed to determine not only the effect of the factors on the value of the process but also the effect of the interaction of any two of the factors, any three of the factors, or the interaction of all four factors.

A two-level fractional factorial design recognizes that this is more information than is really needed, since fourth order or even third order interactions of factors are rarely significant when compared to the effects of the single factors. A fractional factorial approach to the above problem would be to run a 3 factor factorial experiment with only 8 (=2^3) runs and control the fourth factor so that it would have the same high-low pattern as the third order interaction of the first three factors and thus be "confounded" with it.

Analysing the data from the three factor experiment would thus yield the effect of the fourth factor in place of the hopefully nonexistent third order interaction. The fractional factorial experiment design is not particularly easy to describe. For more information on the subject see the article by G. E. P. Box and J. S. Hunter in Technometrics, Volume 3, Number 3, August 1961, pp. 311-351.

I have written a preliminary version of a fractional factorial design package in muLISP-80. If there is sufficient interest in such a package, I will round out the system with a manual and extend the system to produce a FORTRAN program to analyse the results from the experiment which was designed by the muLISP program. [For more information contact David Dunthorn at CF Systems, 908 W. Outer Drive, Oak Ridge, TN, 37830, U.S.A.]

(This page intentionally left blank)