Prose is poetry

by Bill Musgrave

In the beginning there was raw machine code. Programmers had to think at the arithmetic level; they had to specify each arithmetic operation to be performed to solve their problem. But this was not good.

So programming languages evolved into assembly language. The programmer was freed from having to memorize machine codes for each operation because each now had a mnemonic code. He could also stop worrying about writing absolute code; he could start using symbolic addresses, and if he were lucky he could define and use macros. He saw this was good. Yet the problem level at which the programmer could specify his algorithm was again arithmetic.

Higher level languages—like *Fortran*—appeared next. With *Fortran* the programmer could address his problem at a level above arithmetic—he could now specify the numerical parts of his algorithm explicitly, using algebra. *Fortran* simplified the programming task to the point where many people, not only computer programmers, could program.

With the birth of timeshared *Basic,* more and more people wrote computer programs. Although *Basic* was easier to use than *Fortran,* the programmer still had to reduce his problem to sequences of algebraic expressions: The engineer who needed to integrate some function had to write his own numerical quadrature routine.

A proud prosody

But for the past year or so, Control Data has been offering a new language, called *PROSE,* on its CYBERNET network. *PROSE,* developed by PROSE, Inc., of Los Angeles, is a very high-level general purpose language. It offers the things programmers have come to expect in new general purpose languages—such as vector and matrix algebra. But this is only a starting point. *PROSE* also offers simulation capabilities and calculus operations. Instead of sitting at the algebra level, *PROSE* has risen to the calculus level.

PROSE's ability to address problems on the calculus level makes programming large and small problems easy. Users aren't bothered with the details of the problem's solution; they merely formulate the problem, write it in *PROSE,* and the *PROSE* system does the work.

Programming efficiency is increased and the possibility of errors is decreased. According to Dr. Barnet Krinsky, a senior staff physicist with Hughes Aircraft, "Using *PROSE* I was able to solve a nonlinear optimization problem of some complexity in about three weeks. I estimate that this same problem would have taken up to six months to do in *Fortran.*" Dr. Krinsky pointed out two great advantages of *PROSE.* First was the ease with which the system model was programmed. And second, he found it useful to have a number of optimization techniques from which to choose. He was able to experiment with them and decide which was best suited to his purpose.

```
NO.                          PROSE STATEMENTS
-------------------------------------------------------------------
 1     PROBLEM .CALCULUS.EXAMPLE.PROBLEMS
    [*************************************************************
    *                                                            *
    *    EXAMPLE 1 - DEFINITE INTEGRAL -                         *
    *       CALCULATE THE DEFINITE INTEGRAL OF SQRT (1+X**2) FROM 0 TO 1 *
    *                                                            *
    **************************************************************]
 2          IDENTIFY FOFX AS .FUN
 3          Y=.INTEGRAL (FOFX,0,1,4)
 4          ROW PRINT Y

 5          FUNCTION .FUN(X)
 6          END WITH .SQRT (1+X**2)

 7       END
-------------------------------------------------------------------
INPUT CARDS .....    16

DECK NAME .CALCULU GENERATED ON 10/07/74 BY PROSE VERSION 1.10

PROGRAM SYMBOL REFERENCES .....

BLOCKS (DEFINED=*)
   *.CALCULU     *.FUN

GLORAL VARIABLES FOR BLOCK .CALCULU
   FOFX          Y

***** END OF PREP INPUT DECK

***** PROSE PROGRAM ASSEMBLY SUMMARY

      BLOCKS .....
             .CALCULU    .FUN    .PR2000    .PR2001

      SUBROUTINES .....
             PR0033

      GLOBAL VARIABLES .....
             FOFX          Y

***** PROGRAM ASSEMBLED UNDER PROSE VERSION 1.10
      ENTRY POINT   = .CALCULU
      PROGRAM SIZE  =  137 WORDS
      MINIMUM ASSEMBLY FIELD LENGTH = 33565 OCTAL WORDS
      OPTIMUM  ASSEMBLY FIELD LENGTH = 33565 OCTAL WORDS

VARIABLE VALUES .....
                  Y
      1.147793E+00
```

Black box magic

Like many other languages, *PROSE* can solve explicit algebraic problems. An explicit problem might be viewed as a black box machine into which the user can throw his input, turn a crank, and have the answer pop out at the other end. Problems like this are characterized as having known inputs and unknown outputs.

An implicit algebra problem is somewhat more com-

Reprinted from COMPUTER DECISIONS – December 1974

plicated: A function and its value are known, and the problem is to determine the values of the variables in the function. Solving a quadratic equation is an implicit problem. A user might try to use the black box again, this time putting the known value of the function into the machine through the output hopper. He could twist the crank backwards and hope the unknowns will pop out of the top of the machine. Unfortunately, this probably won't work. If the function is easily invertible, it could be expressed as an explicit problem. But inverting functions is quite often difficult. Here's where *PROSE* is sheer poetry!

```
NO.                         PROSE STATEMENTS
--------------------------------------------------------------------------
         PROBLEM .CALCULUS.EXAMPLE.PROBLEMS
   [*****************************************************************]
    *                                                               *
    *     EXAMPLE 2 - IMPLICIT EQUATIONS-                           *
    *                                                               *
    *         SOLVE THE IMPLICIT NONLINEAR EQUATIONS                *
    *              12.5-3*X1**2-X3=0                                *
    *              3.317-.SIN(X1)-.EXP(X2)=0                        *
    *              1.609-X2*.LOG(X0)=0                              *
    *                                                               *
    *          FOR X1,X2,X3                                         *
    *                                                               *
    *****************************************************************
              ALLOT X(3), Y(3)
              X=.DATA(2,2,2)
              FIND X IN .EQS TO MATCH Y [ TO ZERO ]
              PURGE X, Y

              MODEL .EQS
                      Y(1)=12.5-3*X(1)*X(2)-X(3)
                      Y(2)=3.317-.SIN(X(1))-.EXP(X(2))
                      Y(3)=1.609-X(2)*.LOG(X(3))
              END

         END
--------------------------------------------------------------------------
INPUT CARDS .....    26

DECK NAME .CALCULU GENERATED ON 10/07/74 BY PROSE VERSION 1.10

PROGRAM SYMBOL REFERENCES .....

BLOCKS (DEFINED=*)
   *.CALCULU      *.EQS

SUBROUTINES
     PR1001


GLOBAL VARIABLES FOR BLOCK .EQS
     X              Y

GLOBAL VARIABLES FOR BLOCK .CALCULU
     X              Y

SOLVERS
     AJAX

CONVERGENCE CONDITION ..... UNKNOWNS CONVERGED
                            CONSTRAINTS SATISFIED
                            ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER .........   [INITIAL]          1               2         ...        6
UNKNOWNS
            X (  1,  1)   2.000000E+00   2.294485E+00   2.628441E+00   ...   2.500787E+00
            X (  2,  1)   2.000000E+00   1.342432E+00   1.070884E+00   ...   1.000323E+00
            X (  3,  1)   2.000000E+00   2.678497E+00   3.783681E+00   ...   4.995219E+00
CONSTRAINTS
            Y (  1,  1)  -1.500000E+00   5.809312E-01   2.720562E-01   ...  -2.094964E-09
            Y (  2,  1)  -4.981354E+00  -1.260712E+00  -9.188176E-02   ...   6.219238E-10
            Y (  3,  1)   2.227056E-01   2.863608E-01   1.839781E-01   ...   1.643819E-09

            **********************END OF LOOP SUMMARY**********************
```

Mathematical madrigal

A *PROSE* program is built of various kinds of blocks. The problem block is very similar to its counterpart in other procedure oriented languages. In the problem block, things like storage requirements are specified, variables are initialized, and input/output routines are performed. It is also in the problem block that the user tells *PROSE* what he wants to determine for various model blocks. Model blocks define what the user knows about the problem. A model block might contain an equation to be optimized, a system of simultaneous equations, or the description of a system to be simulated. *PROSE* also has procedure blocks, function blocks, and interrupt blocks.

When a user has an implicit problem *PROSE* lets him specify what he knows about the problem and what he wants to learn about it. The programming language itself selects the proper method for solution and keeps the tedious solution process below the user's level of awareness.

Solving a set of implicit non-linear equations is a simple matter in *PROSE*. In *Fortran*, a programmer would have to figure out how to adjust his approximations best, and then he would have to loop through the equations until the system converged. In *PROSE*, all the programmer has to do is specify the equations in a model block, run some routine set-up procedure in the problem block and ask *PROSE* to solve the model.

A definite lyric style

Solving the equations

$$12.5-3x_1x_2-x_3 = 0$$
$$3.317-\sin x_1-e^{x_2} = 0$$
$$1.609-x_2\log x_3 = 0$$

took eleven lines of code in *PROSE*. The solution was found in six iterations through the solver mechanism. These iterations were controlled by *PROSE* and need not be supervised by the programmer. (Above.)

Calculating a definite integral in *Fortran* or *Basic* would probably be done by either numerical quadrature or a Monte Carlo method. In *PROSE* it's done by invoking a function named, appropriately enough, .INTEGRAL. Integrating a simple function can be done in as few as seven lines. (Opposite.)

Minimizing and maximizing functions is another area where *PROSE* can be expected to excel, since derivatives are used to find maxima and minima. A problem block is written to specify an initial set of values for the variables in the function, then *PROSE* is told to find the values that maximize or minimize the function. The function itself is specified in a model block. It takes *PROSE* seven iterations to minimize a fourth degree polynomial.