Please return to Chris

Computer Museum

Journal

of the HP General Systems Users Group

Volume 2 No. 2 October 1978

Published 4 times yearly

Contributions: Address the Journal Editor

Journal Editor Elias Zabor
HP General Systems Users Group, c/o Hewlett-Packard Company, 5303 Stevens Creek Blvd., Santa Clara, CA 95050, (408) 249-7020

Featured articles

The Journey From Batch to On-Line Processing: One User's Experience

by Milton W. Bowden
Director, Data Processing Services
Allan Hancock College
800 South College Drive
Santa Maria, CA 93454

This presentation might be better titled "IF YOU'RE NOT SURE OF THE ANSWER, MUMBLE CONVINCINGLY." As you will see later during the presentation, I mumbled often.

Allan Hancock is a California Community College located on the Pacific Coast approximately 170 miles north of Los Angeles. Our economy is largely agricultural but we are adjacent to Vandenberg Air Force Base where aerospace activity is relatively high.

I came to the college in 1969 as the data processing manager. At that time we had an IBM 1401 which was used jointly by the administration and instruction. The 1401 stayed with us until 1974 since a purchase agreement with a huge educational discount had been signed one month prior to my employment, and if the machine was disposed of prior to the end of a five-year period, the educational discount (60%) had to be repaid. Had I known about the purchase and iron-clad contract I would probably not have accepted the job. We then resigned ourselves to AUTOCODER and COBOL and control-sequential file access and came up with our student accounting package which has evolved somewhat, but is, in large part, still with us today.

In late October of 1971, we were notified that some vocational education grant money was available. We sat down and hastily put together a project which was subsequently funded and we installed an IBM 1130 in January 1972. This was dedicated to instructional use and stayed with us until June of 1973. Even though the project was statistically a success, I never did view the 1130 seriously for production work.

After we passed the mid-point in our contract payments on the 1401, our monthly payments dropped off enough for us to begin to plan for a replacement system. Phase I of our replacement plan called for an IBM System 3 Model 10. It was installed in September 1973. Phase II called for a memory and disk expansion and the addition of the dual programming feature. Before this could be implemented, the System 3 Model 15 was announced. We became one of the first educational institutions to install a Model 15. This occurred in July 1974.

In September of 1974, we said goodbye to the 1401 and put approximately $40,000 in our pockets in the process. It seems that someone had the perfect home for it. At this point, we were quite pleased with ourselves since we had survived a couple of major conversions and wound up with a super-batch system that supported RPG, FORTRAN, COBOL, and BAL quite nicely. The shock was to come later.

In early 1973, we set a goal of having a terminal oriented system by the fall of 1976. Toward this end we added 3340 disk drives to the Model 15 and started to explore things like CCP, TOTAL, etc. At this time our configuration was a 128K CPU, two 3340 disks, a 2560 MFCM reader/punch, and a 1403 Model 5 line printer. We had an additional 128K of CFI memory free because we agreed to test it for them. Our monthly rent was approximately $6,500. The shock came when we looked at the possibility of adding two terminals and CCP and the other items required to interface this equipment. The monthly rent would have been very close to $9,000. This would have given us two administrative terminals and nothing in the area of instruction. So at this point we resigned ourselves to being happy with what we had.

In June 1975, I visited Hewlett-Packard's exhibit at the National Computer Conference. I saw the HP 3000 CX and quite frankly could not believe either the price or what it was supposed to be able to do. Bear in mind the exercise we had recently gone through and the "we cannot afford it" attitude we had at the time. Over the next few months we did a lot of looking, comparing, and wishing. It was about this time that rumors and information started to leak out out on the HP 3000 Series II. We bit the bullet and obtained permission to go out to bid for a terminal oriented system as a replacement for the System 3/15. There was a proviso that we shop with our current budget and not expect any new money. It took me a month to write the bid specifications, and I must admit that I woke up many nights asking myself if I knew what I was getting into. Here I was with 12 years of DP experience, all of it on nice comfortable, safe batch systems. Now I was busily putting together a process which, if unsuccessful, could ruin what had been a fair reputation and could even cost me my job. During this period of time I did a lot of the mumbling mentioned earlier.

When the bid specifications were finally finished and published, one might think we could relax and wait for the results. The fact that we were actually headed for a system that would put the power and responsibility for creation, maintenance, and use of the information that was their responbility in the hands of the user, was somewhat unnerving. Of course, there was the small item of our department being responsible for providing the software, training, and implementation necessary to make all of this possible. Again–mumble convincingly. Add to this the realization that we could no longer cover up our mistakes or problems, re-runs, etc., so easy to do in a batch environment. Further, consider the security problems associated with sharing a system with three or four hundred inquisitive students. I think you can start to appreciate the acute paranoia I started to experience.

By the time the day arrived for opening of the bids, I was smoking two packs of cigarettes and drinking two plus gallons of coffee each 24 hours. As it turned out, we wound up with bids from IBM (370/115 Model 2), Burroughs

(Burroughs B-1700), and Hewlett-Packard (3000 CX). Hewlett-Packard met all the bid specifications with a couple of minor exceptions and was low, and proposed a five-year purchase plan with annual payments in arrears. This gave us a system that would handle 16 terminals, provide interactive IMAGE/QUERY, six 2640 CRT terminals, two 7260 optical and/or punched card readers, a 600 LPM printer, two 47 megabyte disk drives, plus a beautiful tape drive for system backup. I refer to the tape as "beautiful" because our previous method of backup was to keep a copy of our critical files on the same disk as the file in use. The remainder of our limited backup was in cards, reports, etc. In addition to the Hewlett-Packard equipment we purchased, there was sufficient money to rent from General Telephone three ADDS 520 CRT's, a teletype, and a GE Terminette 1200.

After Hewlett-Packard was awarded the contract, we came to that point in one's professional journey jokingly referred to as "Ulcer Gulch." We had 600 plus production programs that required conversion from the System 3/15 to the HP 3000. About 90% of these programs used indexed sequential files. Approximately 85% were written in COBOL, 10% in RPG and the other 5% in FORTRAN and BAL. We also heavily used IBM's Sort which had sort select with include/omit capabilities. Hewlett-Packard's Sort did not have these capabilities. To get around this, Hewlett-Packard wrote and contributed a front end to their sort which took our sort specifications as they were and gave us the same capabilities we had previously on the Model 15.

Our conversion plan was quite simple; we put our source into data files and wrote a program that would make the obvious changes. We simply wanted the programs to function on the HP 3000. The enhancements and obvious changes were to wait until we could say everything was functional. In September 1976, we began a parallel operation and were successful in selling our purchase accruals on the Model 15 so that we came away with approximately $11,000 to help defray conversion costs. We chose the payment in arrears type of contract so that we had no payment to Hewlett-Packard for the first year. This approach is extremely helpful if you have a cash flow problem.

MORE ABOUT CONVERSION

The nearest HP 3000 that we could use for conversion and testing was located in Fullerton, California which is about 200 miles from our installation. We assigned both our programmers to the conversion project and starting in April 1976 a typical work week was as follows:

1. The source programs associated with a particular application were identified and placed in data files on the Model 15.

   We then processed these source programs against a conversion program which made all the known changes required. Obviously this was not 100% effective, but it certainly reduced the effort required to get a successful compile on the HP 3000.

   All data files associated with a particular application were also copied (in some cases only representative portions were used) to disk.
2. The disk containing source and data was then transported to a friendly service bureau which had a Model 15 with both disk and tape. We then copied both source and data from disk to tape. Allowing for travel time and time at the service bureau, Tuesday was usually shot.
3. On Wednesday morning we arrived at the HP data center with tape in hand and proceeded to load everything into our account.

   The source programs were compiled and, when necessary, corrections made. Since we were performing this work at an HP data center, there was usually an HP System Engineer around to assist with particularly difficult programming or operations problems. Our programmers received a lot of training during this time.

   The data center closed at 5:30 p.m. daily, so in order to optimize our time, we acquired a CRT and acoustic coupler and continued our compiling and debugging from our hotel room over the telephone.
4. By Thursday we usually had our source compiled and stored in our account. The rest of the day was spent testing, recompiling, retesting, etc. until we were satisfied with the results.

   On Thursday evening we returned home to ready ourselves for the next round.
5. Fridays were spent reviewing and revising our plans and schedule for the next week. The first couple of weeks there was the feeling that we would never make it. However, by the end of the third week our learning curve was climbing steadily and more and more was being accomplished in the same time period.

The conversion process required two persons for about two and one-half months or five man-months. Our objective was to convert all our programs straight across without attempting to take advantage of the new capabilities of the HP 3000. This approach assured us of being able to begin our parallel operation on schedule. This was a super conservative approach and I'm not sure I would do it the same way if I were faced with the same decision again. It was necessary, however, to write a few front-end type programs for some of our batch programs. We followed these with specific interactive inquiry and update programs. Again, this is not the most efficient approach, but when you have a staff of seven people (two data entry, two operators, two programmers/analysts, and myself) who have zero experience with on-line applications, it provides an excellent learning experience and is relatively safe.

Almost two years have passed since we started the conversion to the HP 3000, and in all honesty I must state that "the journey from batch to on-line processing" was not nearly as bad as I had imagined. We now have most of our major applications on-line and acceptance by our users has been very good. The large stacks of reports, as a status symbol, have been replaced by CRTs.

One of our pivotal applications is our college catalog and schedule preparation and maintenance system. Since the college catalog is the only official source of information concerning courses offered at the college, we decided first to capture its contents in a machine format that would allow us to maintain it on-line. The master files had to contain all data required to support all applications making reference to the catalog. This includes almost everything on campus. So, rather than repeat course titles, units, hours, grading options, etc., in many files, we have all our programs access the single course master file for this information.

We start the preparation of our schedule with a "Section Request Form" on which the department head enters the number of sections of a particular course that is to be offered. These are entered through a batch program which produces the "Class Schedule Data Form" and assigns a unique four-digit section or ticket number to each offering. During this process, a file is built minus the variable information. The variable information is transmitted to the Data Processing Department via the "Class Schedule Data Form." These are processed through our OpScan 17 optical mark reader. This data can also be entered or changed via a terminal. After all the edits are completed, the schedule is produced by department, by location, by hour of the day, by instructor, and every other conceivable sequence. The schedule runs are accompanied by a "Room Utilization Chart" that shows which classes are attempting to use the same room at the same time. This report is also used by most departments on campus to determine what rooms are available at what times.

We now have some options available in this system which allow us to carry the schedule from one semester to another and simply update the variable data. This particular process has saved considerable time.

Our schedule is printed and distributed as an insert in three local newspapers. This method gets our schedule into virtually every home in our district. We now print and distribute over 60,000 copies of the class schedule less expensively than just the printing cost of 20,000 copies employing our former method.

Hopefully, some of the above may be useful to you. I will be glad to answer any questions you may have. If, for any reason you happen to be in our area, please stop by and chat with us. We have so many ways of doing things, you're bound to like something.

• • •

Structured Design and Segmentation[†]

by Fred Waters
Hewlett-Packard Co.
San Diego Division
San Diego, California

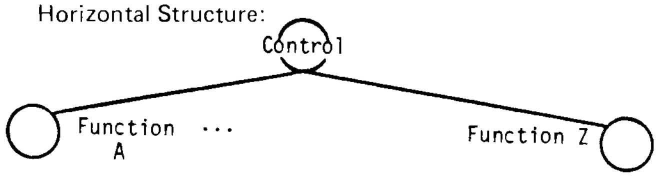
When I was asked to discuss application design, I didn't hesitate in selecting Structured Transaction Processing. Even of greater importance to HP 3000 Users – How do we relate structured programming to a "segmented" environment. Let's discuss these questions in turn.

Artists have an expression – "Form follows function." So too does the software we write. What are the functions? — To provide an efficient computerized solution to a users need? — To reduce solution costs? — To provide an expeditious solution? — Or to provide an easily maintained system? All of these are objectives of our software and functions of how we construct it. Structural programming approaches are given as a tool to assist us in meeting these objectives.

Currently, there are two fundamental "structural" techniques available: (1) A horizontal technique which parallels functions to be performed; (2) Vertical structure which parallels the transaction performed upon.

Examples:

Horizontal Structure:



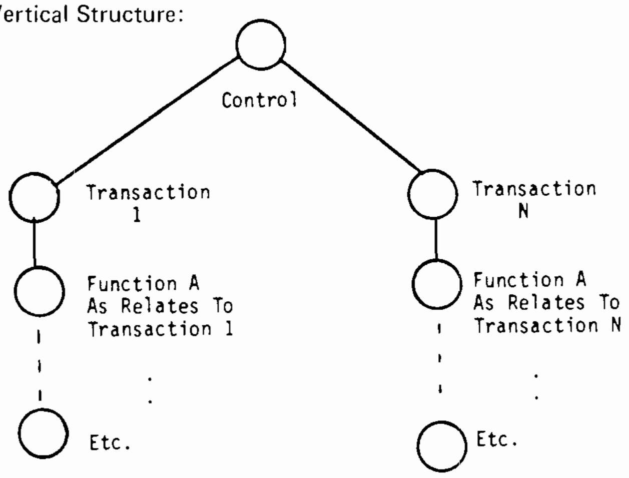
Each transaction "performs" functions A through Z as required. In COBOL this may appear as follows:

```
CONTROL-LVL   Section 0.
   If Transaction-1 then
      Perform Function A,
             •
             •
             •
      Perform Function Z

   Else
     •
     •
     •
Function-A   Section 1.
   If Transaction-1 then
           •
           •
           •
   Else
      (Etc.)
```

† *Originally presented at Southern California Regional Users Group Meeting, March 1st and 2nd, 1978.*

Vertical Structure:


Control
Transaction 1
Transaction N
Function A As Relates To Transaction 1
Function A As Relates To Transaction N
Etc.
Etc.

Each transaction has an "In-Line" flow which is structured. The functions belonging to each transaction are grouped together.

In a COBOL program this might take the following format:

```
CONTROL-LVL   Section 0.
    If Transaction-Type-1  then
      Perform Trans-Type-1
    Else
              •
              •
              •
Trans-Type-1   Section 2.
      Function A
          •
          •
          •
      Function Z
Etc.
```

Typically, what is taught as structured programming ignores vertical structures.

The real world of how or which of these techniques is used is greatly influenced by several internal/external factors:

A. Vendor hardware constraints and design.
B. Operating System Design.
C. Applications Design Objectives.
D. Standards Objectives.
E. Organizational Objectives.

The approach (technique) we chose is probably somewhere between both methods. In making this choice we must apply our understanding of the factors above.

Applying this to a segmented machine and considering memory utilization, then the question of whether function or transaction segmentation is germane. If we decide that a given function is of greater importance, then we should organize functionally. If a transaction type is of greater importance, then, organize along those lines. To help us understand how this applies to the HP 3000, consider a segment size of 6K words as our standard. Also consider an analysis of proposed functions and transactions. To do this, we formulate a function/transaction matrix.

	Transaction			
Function	1	: . .	N	
A	X_{A}/C_1		X_{A}/C_N	S_A
B	X_{B}/C_1	. . . .	X_{B}/C_N	S_B
C	X_{C}/C_1	. . .	X_{C}/C_N	S_C
.	.		.	
.	.		.	
.	.		.	
Y	X_{Y}/C_1	. . .	X_{Y}/C_N	S_X
Z	Z_{Y}/C_1	. . .	Z_{Y}/C_N	S_Z
	TN_1		TN_T	S_T

Where X_A, X_B, ... etc. represents the expected count for that function by that transaction type per transaction

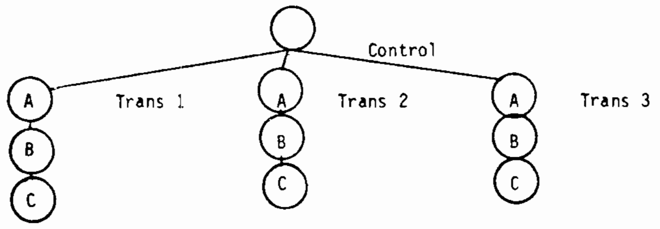
C_N = The count of that transaction type.

The sum horizontally represents the total of these ratios $TN_1 \ldots TN_T$ = The total transaction load per transaction type/some time period. This table allows one to quickly see where most processing time/effort will occur. Then we apply the approximate structure. To help us with segmentation, then one may use a tree chart to locate that area of the structure we wish to segment by.

Let's use the transaction table example:

	Transaction Type				
Function	1	2	3		Time Est.
A	1/200	5/600	10/200	16	2 ms
B	3/200	3/600	1/200	7	1 ms
C	5/200	25/600	2/200	32	4 ms
	200	600	200		1000

Transaction 2 will require the maximum time and has the maximum number of transactions, therefore, optimize on transaction type 2. But Function C is the most heavily used.



A segment table then takes this picture as a median:

Segment 1
Transaction 1 functions A, B
Transaction 3 functions A, B
Segment 2
Transaction 2 functions A, B
Segment 3
Function C

Using this method allows us to construct testing models to verify systems performance as relates to a given application.

● ● ●

Programming For Regression Analysis[†]

by James P. Schwar and Stan J. Perambo
Lafayette College Computer Center
Easton, Pennsylvania 18042

INTRODUCTION

In many areas of scientific research, especially in the social and biological sciences, regression analysis has become the standard tool for determining the nature of the relationship between a given dependent variable and some set of independent variable and some set of independent variables. This is due to the fact that regression analysis, if used properly, makes it possible to describe how one variable varies with a set of other variables. Given an interval scale of measurement for both the dependent and independent variables, regression analysis can therefore be a powerful predictive tool.

Prediction, of course, does not imply causality. To be able to predict the value of variable A by knowing the value of variable B does not imply that B causes A. However, it can perhaps be argued that the ability to predict is an indicator of understanding in the sense that as understanding (i.e., the ability to provide causal explanations) becomes better, regression analysis will undoubtedly continue to be widely used.

Because of the number and complexity of the calculations involved, regression analysis is almost always done on a computer. In this paper, the authors first discuss the general equations and the model for multiple linear regression. Then, two specific problems are mentioned: 1) calculation of the multiple correlation coefficient when the data matrix contains numerous observations that are an exact solution to the regression equation, and 2) the impossibility of performing a regression analysis when the data matrix contains a column of constants. Finally, selected code for the regression procedure in the LISA statistical package is shown, specifically those sections that test for the problems mentioned above.

[†] *Editor's Note: The authors have received a number of inquiries regarding the regression analysis model used in LISA, the statistical analysis program contributed by Lafayette College to the Users Group Library. In response the authors submitted this article which will be of value for users of LISA as well as to anyone interested in programming for regression analysis.*

THE REGRESSION EQUATION

The general multiple linear regression equation is:

$$Y = a_0 + a_1X_1 + a_2X_2 + \dots + a_nX_n$$

where

Y = dependent variable

$X_1 \dots X_n$ = n independent variables

a_0 = intercept

$a_1 \dots a_n$ = regression coefficients

Given a data matrix consisting of N observations (or rows) and n variables (or columns), the regression coefficients are to be computed such that the sum of the squares of the observed Y's less the calculated Y's is minimized. The dependent variable Y and the independent variables $X_1 \dots X_n$ can be replaced by standard (or Z) variables, where

$$Z_0 = \frac{Y-\overline{Y}}{S_Y}$$

$$Z_1 = \frac{X_1-\overline{X}_1}{S_1}$$

$$\vdots$$

$$Z_n = \frac{X_n-\overline{X}_n}{X_n}$$

$$\overline{Y} = \frac{\Sigma Y}{N} \quad ; \quad S_Y^2 = \frac{\Sigma(Y-\overline{Y})^2}{N-1}$$

$$\overline{X}_n = \frac{\Sigma X_n}{N} \quad ; \quad S_N^2 = \frac{\Sigma(X_n-\overline{X}_n)^2}{N-1}$$

Σ is the summation over all N observations. $\overline{X}$ or $\overline{Y}$ is the arithmetic mean and S^2 is the variance. The regression equation in terms of the standard variables is

$$Z_0 = B_1Z_1 + B_2Z_2 + B_2Z_2 + \dots + B_nZ_n$$

where $B_1 \dots B_n$ are called beta weights. These beta weights are the regression coefficients for the standard variables. Beta weights also represent slopes and are simply the following partial derivatives:

$$\frac{\partial Z_0}{\partial Z_1} = B_1$$
$$\vdots \qquad \vdots$$
$$\frac{\partial Z_0}{\partial Z_n} = B_n$$

The regression coefficients are calculated such that

$$\Sigma [Z_0 - (B_1 Z_1 + B_2 Z_2 + \ldots . + B_n Z_n)]^2$$

is a minimum. Performing this summation and setting each partial derivative of this sum with respect to a beta weight to zero the following matrix equation is obtained.

$$\begin{bmatrix} 1 & r_{12} & r_{13} \cdots r_{1n} \\ r_{21} & 1 & r_{23} \cdots r_{2n} \\ r_{31} & r_{32} & 1 \ \ldots r_{3n} \\ \vdots & \vdots & \vdots \quad \vdots \\ r_{n1} & r_{n2} & r_{n3} \cdots \ 1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} r_{01} \\ r_{02} \\ r_{03} \\ \vdots \\ r_{0n} \end{bmatrix}$$

where

$$\begin{bmatrix} 1 & r_{12} & r_{13} \cdots r_{1n} \\ r_{21} & 1 & r_{23} \cdots r_{2n} \\ r_{31} & r_{32} & 1 \ \ldots r_{3n} \\ \vdots & \vdots & \vdots \quad \vdots \\ r_{n1} & r_{n2} & r_{n3} \cdots \ 1 \end{bmatrix}$$ = $\overline{R}$ = matrix of intercorrelations among the independent variables.

$$\begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \vdots \\ B_n \end{bmatrix}$$ = $\overline{B}$ = vector of beta weights

$$\begin{bmatrix} r_{01} \\ r_{02} \\ r_{03} \\ \vdots \\ r_{0n} \end{bmatrix}$$ = $\overline{r}$ = vector of intercorrelations of the dependent variable with each dependent variable

Solving for the beta weights, the matrix

$$\overline{B} = \overline{R}^{-1} \overline{r}$$

results. The regression coefficients for the original regression equation $Y = a_0 + a_1 X_1 + \ldots . + a_n$

are calculated from

$$a_1 = B_1 \frac{S_Y}{S_1}$$
$$\vdots$$
$$a_n = B_n \frac{S_Y}{S_n}$$

and the intercept is given by

$$a_0 = \overline{Y} - a_1\overline{X}_1 - a_2\overline{X}_2 - \ldots . \ a_n\overline{X}_n$$

The multiple correlation coefficient R is determined by taking the square root of the sum of the products of beta weights, times the corresponding elements of the intercorrelation vector, that is

$$R = \sqrt{r_{01}B_1 + r_{02}B_2 + \ldots . + r_{0n}B_n}$$

THE REGRESSION MODEL

The two variable regression model

$$Z_0 = B_1 Z_1$$

shows that B_1 must lie in the range +1 to -1 inclusive and that B_1 is numerically equivalent to the correlation coefficient. For the three-variable regression model

$$Z_0 = B_1 Z_1 + B_2 Z_2$$

the beta weights can be obtained from the following equations:

$$B_1 = \frac{r_{01} - r_{02} r_{12}}{1 - r_{12}^2} \ ; \quad B_2 = \frac{r_{02} - r_{01} r_{12}}{1 - r_{12}^2}$$

The beta weight will be equivalent to the correlation coefficient only if there is no intercorrelation among the independent variables.

One potential problem area in regression analysis is when the data matrix contains a high percentage of observations that gives an exact solution. In this case, if the intercorrelation among the independent variables is very high, the beta weights may be greater than unity and the computed value of the multiple correlation coefficient approaches one. Another difficulty occurs when the data matrix contains a column of constants. The standard deviation for that variable will be zero and the associated regression coefficient is undefined.

COMPUTER PROGRAM

LISA is a Fortran program written to perform general statistical analysis on a data matrix. One of the several statistical options in LISA is multiple linear regression analysis. Regression in LISA follows the techniques discussed previously, namely the computation of beta weights from which, regression coefficients are determined. Figure 1 shows the procedure used for regression analysis. Regression analysis in turn calls the following procedures:

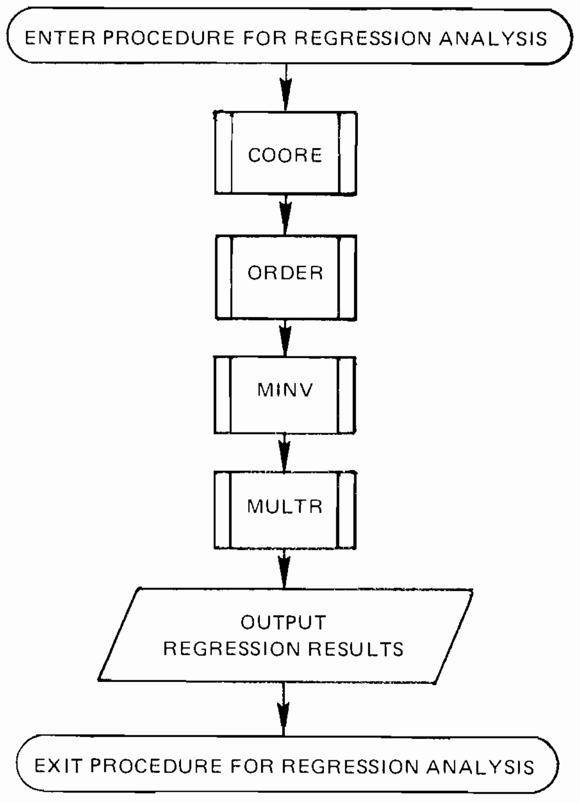

ENTER PROCEDURE FOR REGRESSION ANALYSIS
COORE
ORDER
MINV
MULTR
OUTPUT REGRESSION RESULTS
EXIT PROCEDURE FOR REGRESSION ANALYSIS

Figure 1. Procedure for Regression Analysis Using LISA

COORE calculates the means, standard deviations, sums of the cross products of the deviations from the means, upper triangular portion of the correlation coefficient matrix and the diagonal of the matrix of the sums of the cross products of the deviations from the means. All calculations are for n variables in the data matrix, summed over N observations.

ORDER constructs a subset matrix of intercorrelations among the independent variables and the vector of intercorrelations of the independent variable with each dependent variable, for those dependent variables and independent variables specified.

MINV inverts the matrix of intercorrelations among the independent variables.

MULTR performs multiple linear regression on the data matrix using the variables specified. Output includes the vector containing the regression coefficients; the vector containing the standard deviations of the regression coefficients; vector containing T-values, intercept, multiple correlation coefficient, standard error of the estimate, sum of squares attributable to the regression (SSAR), degrees of freedom for SSAR, mean square of SSAR, sum of squares of the deviations from the regression (SSDR), degrees of freedom for SSDR, mean square of SSDR and the F-value.

The procedure for regression analysis (see Figure 2) checks for a multiple correlation coefficient greater than one. If this condition is true the regression is aborted. Multiple regression analysis runs can be made and procedure COORE is called by the first run only. Procedures ORDER, MINV and MULTR are called during each run since they are dependent on the variables selected for a particular regression run. Procedure MINV checks to see if all elements of the intercorrelation matrix equal unity, and if true, sets the determinant to zero and outputs the message INTERCORRELATION MATRIX CONTAINS ALL ONES. This procedure also tests to determine if a pivot element is zero during inversion of the intercorrelation matrix, and if true, the determinant is set to zero and the message is displayed PIVOT ELEMENT ZERO DURING MATRIX INVERSION. When zero is returned to the regression analysis procedure, the regression run is aborted. The final check performed by MINV is whether the determinant of the intercorrelation matrix is less than 10^{-10}, and if true, the message INVERSE OF MATRIX MAY BE (NEARLY) SINGULAR is displayed. The determinant, however, is not set to zero but remains unchanged so that the regression run does not abort. Procedure MULTR (see Figure 3) checks for beta weights greater than unity and outputs a warning which is indicative of very high intercorrelation among the independent variables. However, if a standard deviation is zero, the multiple correlation coefficient is set equal to two and this value returned to the regression analysis procedure. The message STD DEVIATION IS ZERO - UNABLE TO CALC REGR COEFF is displayed and the regression run is aborted.

Source code and documentation for LISA is available from the Users Group Contributed Library.

References

1. *"Intermediate Correlational Methods,"* Baggaley, John Wiley and Sons (1964).
2. IBM 1130/360 Scientific Subroutine Package-Programmer's Manual.
3. HP 3000 Scientific Library Reference Manual.

```
C.....LISA/3000    SUBR REGRESSION
      SUBROUTINE REGRESSION(*)
      COMMON APPEND,MODE,M,CMAX,FNAME,N,NT,RB,RE,RL
      COMMON TRND,TRENDUSE,IMAXSAVE,JMAXSAVE,OSAVE(8,8)
      LOGICAL APPEND,MODE,TRND,TRENDUSE
      INTEGER CMAX,RB,RE,RL
      CHARACTER*8 FNAME
      CHARACTER*2 REPLY
      DIMENSION XBAR(64),STD(64),D(64),RY(64),ISAVE(64),B(64),
     1SB(64),T(64),DOUT(64)
      DIMENSION RX(4096),R(2080),ANS(10)
C     N=NUMBER OF OBSERVATIONS
C     M=NUMBER OF VARIABLES
      CALL COORE(XBAR,STD,RX,R,D,B,T,15)
      I=1
109   WRITE(6,9002)I
 .                  .
 .                  .
 .                  .
      CALL ORDER(R,NDEP,K,ISAVE,RX,RY)
      CALL MINV(RX,K,DET)
C     IF DET =0 ABORT REGRESSION RUN
      IF(DET.NE.0)GOTO 112
110   WRITE(6,9014)
      GO TO 200
112   CALL MULTR(K,XBAR,STD,D,RX,RY,ISAVE,B,SB,T,ANS)
C     CHECK IF MULTIPLE CORRELATION COEFFICIENT > 1
C     AND IF TRUE ABORT REGRESSION RUN
      IF(ANS(2).LE.1.0)GOTO 113
      WRITE(6,9080)
      GOTO 200
113   MM=K+1
 .                  .
 .                  .
 .                  .
200    I+I+1
 .                  .
 .                  .
 .                  .
      RETURN 1
9002  FORMAT(/1H0,"MULTIPLE REGRESSION......SELECTION",I3/)
9014  FORMAT(1H0,"THE MATRIX IS SINGULAR - THIS SELECTION IS SKIPPED")
 .                  .
 .                  .
 .                  .
9080  FORMAT(1H0,"CANNOT REGRESS - THIS SELECTION IS SKIPPED")
      END
```

Figure 2. Regression Analysis Procedure

```
C       LISA/3000    SUBR MULTR
        SUBROUTINE MULTR (K,XBAR,STD,D,RX,RY,ISAVE,B,SB,T,ANS)
        COMMON APPEND,MODE,M,CMAX,FNAME,N,NT,RB,RE,RL
        COMMON TRND,TRENDUSE,IMAXSAVE,JMAXSAVE,OSAVE(8,8)
        LOGICAL APPEND,MODE,TRND,TRENDUSE
        INTEGER CMAX,RB,RE,RL
        CHARACTER*8 FNAME
        DIMENSION XBAR(64),STD(64),D(64),RX(4096),RY(64),ISAVE(64),
       1B(64),SB(64),T(64),ANS(10)
        MM=K+1
        DO 100 J=1,K
100     B(J)=0.0
        DO 110 J=1,K
        L1=K*(J-1)
        DO 110 I=1,K
        L=L1+I
110     B(J)=B(J)+RY(I)*RX(L)
C       CHECK BETA WEIGHTS & IF GT 1 OUTPUT WARNING
        DO 113 J=1,K
        IF(ABS(B(J)).LE.1.0)GOTO 113
        WRITE(6,9000)
        IF(.NOT.TRND)WRITE(6,9100)(B(J),J=1,K)
        GOTO 114
113     CONTINUE
114     IF(TRND)WRITE(6,9100)(B(J),J=1,K)
        RM=0.0
        BO=0.0
        L1=ISAVE(MM)
        DO 120 I=1,K
        RM=RM+B(I)*RY(I)
        L=ISAVE(I)
C*****CHECK STD DEV FOR DIV BY ZERO & IF TRUE SET R>1 & RETURN
        IF(STD(L).NE.0.0)GOTO 115
        WRITE(6,9200)
        ANS(2)=2.0
        RETURN
115     B(I)=B(I)*(STD(L1)/STD(L))
  .                     .
  .                     .
  .                     .
        RETURN
9000    FORMAT(1H0,"WARNING - BETA WEIGHT(S) GREATER THAN ONE")
9100    FORMAT(1H0,"BETA WTS:",5G12.5/(10X,5G12.5))
9200    FORMAT(1H0,"STD DEVIATION IS ZERO - UNABLE TO CALC REGR COEFF")
        END
```

Figure 3. Procedure MULTR

● ● ●

The Aims and Methods of the Management Learning Project *

by Richard Boot
London Business School
London, England

The overall aim of the Management Learning Project is to encourage and support the use of management educational methods designed to develop the skills and abilities required of managers when operating in situations typified by ambiguity. Inevitably, in such situations, management decision making becomes more than an objective, rational or purely analytical exercise. Individual values, personal judgment and intuition take on a greater significance, as does the ability to be aware of and understand the values and personal judgements of others. It is these aspects of the manager's role that the teaching methods of the Management Learning Project have been designed to develop. It is not intended that those methods should be regarded as alternatives to, or in some way replacements for the various approaches that already exist for teaching the analytical skills required of managers, but rather that they should be seen as complementary to them.

The methods themselves involve the use of a number of computer assisted exercises which provide an opportunity for the manager or management student to build on and learn from his own experience. With this in mind their focus is less on teaching appropriate 'techniques' to be applied in any given decision-making or problem context and more on developing a greater personal awareness of the kinds of assumptions and interpretations made both individually and collectively in such contexts.

In this sense they are likely to represent an addition to the existing repertoire of teachers or trainers already familiar with the use of experiential approaches to learning.

The Methods

The methods themselves are the output of an earlier development project (The Management Decision Making Project) jointly funded by the London Business School and the National Development Programme in Computer Assisted Learning. They can be thought of as falling into two distinct groups: feedback methods and simulations.

The *feedback methods* are based on the principles of repertory grid analysis**. They can be used either in a reflective way to support the self-development of individual managers or in a comparative way to help pairs and/or groups of managers explore in detail their own understanding of each others' points of view in a given problem situation. The main method falling into this "feedback" category is known as "NIPPER."

NIPPER is a flexible system which can be tailored by the teacher/trainer to fit the particular learner needs and learning context he is dealing with. It can be used in a number of different ways. For example, in terms of the individual it can help him reflect upon those past experiences which are relevant to his current situation, be they problems he was faced with, decisions he had to make or whatever. In this way he can become more aware of, or clearer about, the basic assumptions and value judgments he makes, often implicitly, in such situations. Alternatively he might focus on a current problem or decision point and evaluate the options he sees as open to him at the time. In this way NIPPER can help him formulate new options and become clearer about those criteria for choice which are most important to him personally.

With a group of managers faced with the same situation NIPPER makes it possible for that group, amongst other things, to analyze clearly similarities and differences in points of view, identify whose points of view appear most misunderstood, and examine the extent to which assumptions about consensus are borne out in reality.

Finally it can be used to help pairs of individuals explore more fully the extent to which they truly understand each others ideas and views about a given subject.

In addition to NIPPER in the "feedback" category there is "INTERVIEW" which is a self-contained exercise which structures the exchange of views between two people to achieve objectives mentioned for the last use of NIPPER.

Ideally the basic "subject matter" for all these feedback methods would be the learners' real work experience. In some circumstances, however, there will be insufficient shared work experience for the "comparative" uses. In such cases they can be used effectively in programmes involving joint activity, problem-solving exercises, case studies and the like. They could also be used within the context of the simulations.

The *simulations* are of the "behavioural" type which make use of the computer solely to provide a realistic context for the interactions between role-playing participants. This means that learning derives explicitly from these interactions and not, as is often the case, from any "model" of reality programmed into the computer. In other words the simulations are not about "success" in terms of "beating" the computer but rather they provide learners with an opportunity to explore their own and others approaches to decision making, conflict handling, negotiating, etc. Separate simulations have been designed to represent the energy industry (THE POWER GAME), ball bearings manufacture (BALL BEARINGS LTD) and the petfood industry (OFFAL INDUSTRIES).

Full "user" documentation is available for all of these methods.

** The Management Learning Project is funded by the Training Services Agency and located at the London Business School. More information about the project can be obtained from Richard Boot, Director of the Management Learning Project, London Business School, Sussex Place, Regents Park, London NW1 4SA. Tel.: 01 724 1517.*

***See Kelly G.A.* "The Psychology of Personal Constructs" *Norton 1955 and Fransella, F. & Bannister, D.* "A Manual for Repertory Grid Technique" *Academic Press 1977.*

The Aims of the Project

In their development these methods have been used successfully by a large number of managers in a variety of different company and institutional settings. The Training Services Agency, in funding the project, are reinforcing the belief that these methods could represent a valuable addition to the management training and development activities already being carried out in many institutions and organizations throughout the country. With this in mind the official brief of the project is to "maximize the dissemination and effective utilization" of these methods. But what does this mean in practice? It means that for their true value to be realized the methods must become an established part of the repertoire of learning activities of teachers and trainers working in those institutions and organizations.

The role of the Management Learning Project in this is to give full support to those who would like to make use of these methods and, perhaps more importantly, adapt and develop them to suit their own particular applications. This support is available both in terms of technical advice and assistance in transferring and adapting the methods to the users' own computer systems*** and in terms of assistance and advice in integrating the methods into existing training activities, designing new training programmes and the development of appropriate training approaches.

The T.S.A. funding enables the project to provide these methods and all the necessary support for adapting them at virtually no cost to the potential user.

As a final word, it is the longer term aim of the project to help establish a self-supporting network of users of these and similar management development methods within which there would be a free exchange of ideas about applications and developments.

***Where potential users do not have access to their own computer it is possible to make arrangements for them to make use of the London Business School Computer on a timesharing basis.

● ● ●

Total Stack For A Main Program and Subprogram > 64KB? Use An Extra Data Segment

by Jack Howard, HP Neely Sales Region
Computer Systems
System Support Representative
P. O. Box 92105
Los Angeles, CA 90009

Editor's Note: This is Part 2 of a two-part article. The first part appeared in the May/June 1978 Journal issue, Vol. 2, No. 1, (pp. 2-6), and dealt with the extra data segment concept, system performance impact of using extra data segments, and the use of an extra data segment with a COBOL main program.

Part 2 deals with the use of an extra data segment with a main program and subprograms.

Now that we have covered the case of a single program requiring a data segment (stack) greater than 64K bytes, let's look at the situation where a main program and its associated subprograms require a total data stack greater than 64K bytes.

This situation is shown in the printout labeled "Figure 7." Here, "MAIN" compiles successfully, and "SUB," as a subprogram, compiles successfully. However, the greater than 64K byte requirement is detected during the program prep phase (see highlight 17 in Figure 7).

```
PAGE 0001   HEWLETT-PACKARD 32213C.02.00  COBOL/3000 THU, MAR  9, 1978,  2:58

          001000$CONTROL USLINIT
          001100 IDENTIFICATION DIVISION.
          001200 PROGRAM-ID. MAIN.
          001300 ENVIRONMENT DIVISION.
          001400 DATA DIVISION.
          001500 WORKING-STORAGE SECTION.
          001600 01  MAIN-HUGE-FIELD.
          001700       05 FILLER PIC X(10) OCCURS 4000.
          001800 01  MAIN-PARM-FIELD.
          001900       05 MPF PIC X(30) VALUE "FROM MAIN".
          002000 PROCEDURE DIVISION.
          002100 MAIN-PARA.
          002200*
          002300       DISPLAY "PARM SENT BY MAIN = " MPF.
          002400*
          002500       CALL "SUB" USING MAIN-PARM-FIELD.
          002600*
          002700       DISPLAY "PARM RECEIVED BY MAIN = " MPF.
          002800*
          002900       STOP RUN.

     DATA AREA IS %047344 WORDS.
     CPU TIME = 0:00:01.  WALL TIME = 0:00:16.
 END COBOL/3000 COMPILATION.  NO ERRORS.  NO WARNINGS.


 PAGE 0001   HEWLETT-PACKARD 32213C.02.00  COBOL/3000 THU, MAR  9, 1978,  2:59

          001000$CONTROL SUBPROGRAM
          001100 IDENTIFICATION DIVISION.
          001200 PROGRAM-ID. SUB.
          001300 ENVIRONMENT DIVISION.
          001400 DATA DIVISION.
          001500 WORKING-STORAGE SECTION.
          001600 01  SUB-HUGE-FIELD.
          001700       05 FILLER PIC X(10) OCCURS 4000.
          001800 LINKAGE SECTION.
          001900 01  SUB-PARM-FIELD.
          002000       05 SPF PIC X(30).
          002100 PROCEDURE DIVISION USING SUB-PARM-FIELD.
          002200 SUB-PARA.
          002300*
          002400       DISPLAY "PARM RECEIVED BY SUB = " SPF.
          002500*
          002600       MOVE "FROM SUB" TO SPF.
          002700*
          002800       DISPLAY "PARM SENT BY SUB = " SPF.
          002900*
          003000       GOBACK.

     DATA AREA IS %047325 WORDS.
     CPU TIME = 0:00:01.  WALL TIME = 0:00:11.
 END COBOL/3000 COMPILATION.  NO ERRORS.  NO WARNINGS.


  END OF COMPILE
 :CONTINUE
 :PREP $OLDPASS,$NEWPASS
 *** ERROR ***
 ERROR #36 DATA SEGMENT OVERFLOW        <-(17)
 ERR 250 SEGMENTER ERROR
```

Figure 7

A solution to the problem is to change the subprograms to main programs, i.e., separate processes, thereby allocating data stacks of up to 64K bytes for the original main program and for each of what were the original subprograms. An extra data segment provides the parameter passing capa-

Editing Clarification Note: Highlight numbers ($\textcircled{1}$) where they appear are not necessarily continuous numerically; some highlights have been removed in the editing process.

bility ordinarily provided by linkage sections, and process-to-process communication provides the execution control ordinarily provided by CALL and GOBACK statements. (The routine shown in Appendix A of Part 1 contains the extra data segment management routines, as well as the process-to-process communication routines.)

If you are not familiar with parent/child process concepts and communication between processes using Resource Identification Numbers (RINs), at this point it would be beneficial to read the following sections of the MPE Intrinsics Reference Manual, part number 30000-90010:

Section VI	*Resource Management*
Section VII	*Process Handling Capability*
Section VIII	*Data Segment Management Capability*

The basic concept of the use of the extra data segment, for passing parameter data between the 'parent'-and-'child'-process data stacks, is shown in Figure 8. Required communication between the parent-and-child process code, is summarized in Figure 9. Note that path ①of Figure 9 is first time only, i.e., first time parent process code activates child-process code; path ⑥of Figure 9 is last time only.

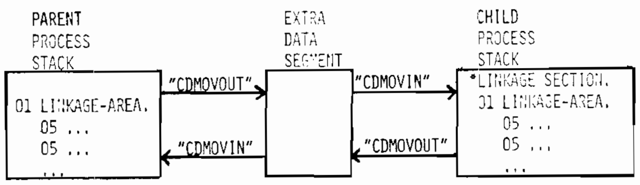


Figure 8. Data Transfer

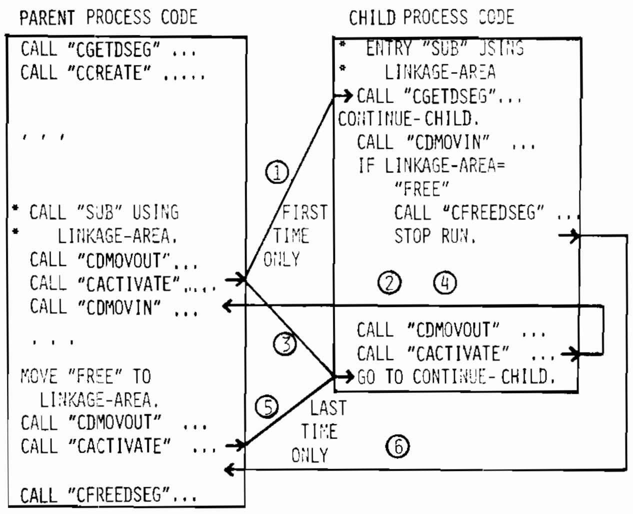
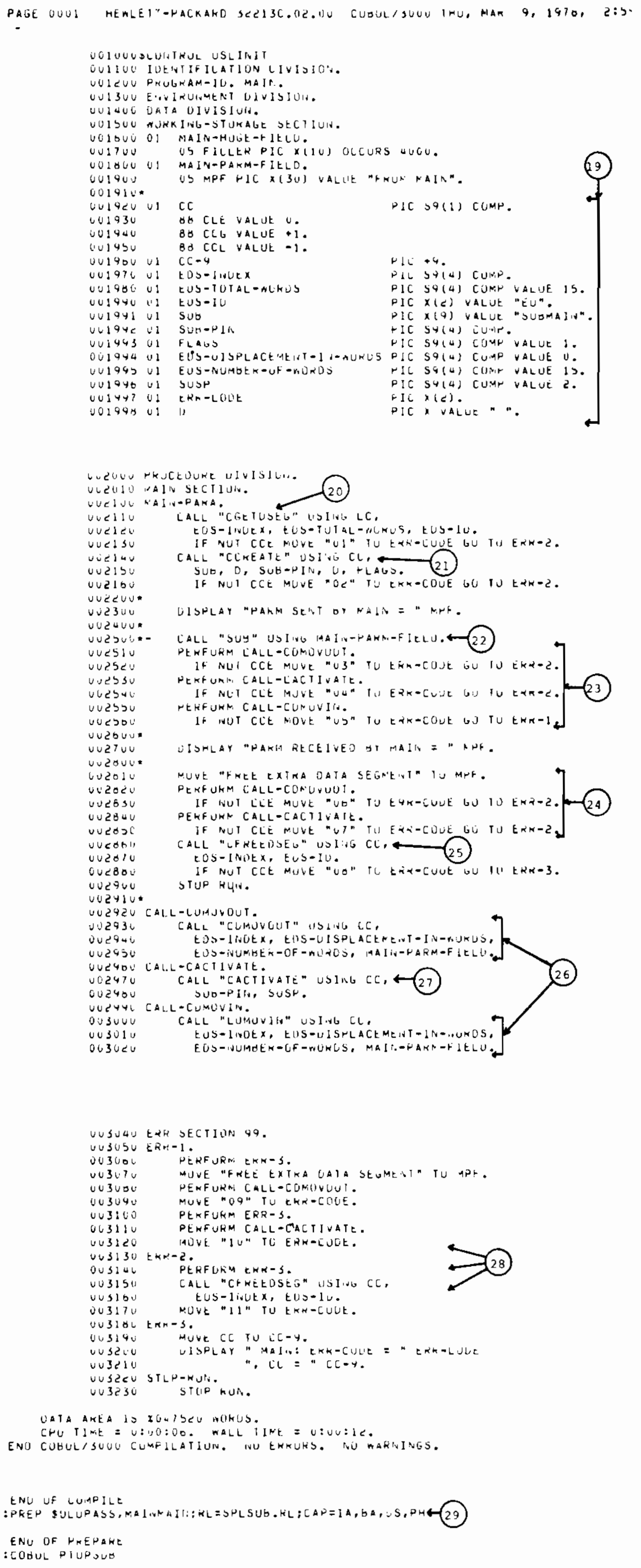


Figure 9. Process to Process Execution Flow

In the following example, the main program "MAIN" and its associated subprogram "SUB" will be changed to two separate processes "MAINMAIN" and "SUBMAIN."

Modifications to program "MAIN" are shown in Figure 10. Explanations of the highlighted items of Figure 10 follow:



```
PAGE 0001   HEWLETT-PACKARD 32213C.02.00  COBOL/3000 THU, MAR  9, 1978,  2:5

$CONTROL USLINIT
 IDENTIFICATION DIVISION.
 PROGRAM-ID. MAIN.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  MAIN-HUGE-FIELD.
     05 FILLER PIC X(10) OCCURS 4000.
 01  MAIN-PARM-FIELD.
     05 MPF PIC X(30) VALUE "FROM MAIN".                      (19)
*
 01  CC                               PIC S9(1) COMP.
     88 CCE VALUE 0.
     88 CCG VALUE +1.
     88 CCL VALUE -1.
 01  CC-9                             PIC +9.
 01  EDS-INDEX                        PIC S9(4) COMP.
 01  EDS-TOTAL-WORDS                  PIC S9(4) COMP VALUE 15.
 01  EDS-ID                           PIC X(2) VALUE "ED".
 01  SUB                              PIC X(9) VALUE "SUBMAIN".
 01  SUB-PIN                          PIC S9(4) COMP.
 01  FLAGS                            PIC S9(4) COMP VALUE 1.
 01  EDS-DISPLACEMENT-IN-WORDS PIC S9(4) COMP VALUE 0.
 01  EDS-NUMBER-OF-WORDS              PIC S9(4) COMP VALUE 15.
 01  SUSP                             PIC S9(4) COMP VALUE 2.
 01  ERR-CODE                         PIC X(2).
 01  D                                PIC X VALUE " ".


 PROCEDURE DIVISION.
 MAIN SECTION.                 (20)
 MAIN-PARA.
     CALL "CGETDSEG" USING CC,
       EDS-INDEX, EDS-TOTAL-WORDS, EDS-ID.
       IF NOT CCE MOVE "01" TO ERR-CODE GO TO ERR-2.
     CALL "CCREATE" USING CC,          (21)
       SUB, D, SUB-PIN, D, FLAGS.
       IF NOT CCE MOVE "02" TO ERR-CODE GO TO ERR-2.
*
     DISPLAY "PARM SENT BY MAIN = " MPF.
*
*-   CALL "SUB" USING MAIN-PARM-FIELD.  (22)
     PERFORM CALL-CDMOVOUT.
       IF NOT CCE MOVE "03" TO ERR-CODE GO TO ERR-2.
     PERFORM CALL-CACTIVATE.                          (23)
       IF NOT CCE MOVE "04" TO ERR-CODE GO TO ERR-2.
     PERFORM CALL-CDMOVIN.
       IF NOT CCE MOVE "05" TO ERR-CODE GO TO ERR-1.
*
     DISPLAY "PARM RECEIVED BY MAIN = " MPF.
*
     MOVE "FREE EXTRA DATA SEGMENT" TO MPF.
     PERFORM CALL-CDMOVOUT.
       IF NOT CCE MOVE "06" TO ERR-CODE GO TO ERR-2.  (24)
     PERFORM CALL-CACTIVATE.
       IF NOT CCE MOVE "07" TO ERR-CODE GO TO ERR-2.
     CALL "CFREEDSEG" USING CC,        (25)
       EDS-INDEX, EDS-ID.
       IF NOT CCE MOVE "08" TO ERR-CODE GO TO ERR-3.
     STOP RUN.
*
 CALL-CDMOVOUT.
     CALL "CDMOVOUT" USING CC,
       EDS-INDEX, EDS-DISPLACEMENT-IN-WORDS,
       EDS-NUMBER-OF-WORDS, MAIN-PARM-FIELD.
 CALL-CACTIVATE.                                  (26)
     CALL "CACTIVATE" USING CC,   (27)
       SUB-PIN, SUSP.
 CALL-CDMOVIN.
     CALL "CDMOVIN" USING CC,
       EDS-INDEX, EDS-DISPLACEMENT-IN-WORDS,
       EDS-NUMBER-OF-WORDS, MAIN-PARM-FIELD.


 ERR SECTION 99.
 ERR-1.
     PERFORM ERR-3.
     MOVE "FREE EXTRA DATA SEGMENT" TO MPF.
     PERFORM CALL-CDMOVOUT.
     MOVE "09" TO ERR-CODE.
     PERFORM ERR-3.
     PERFORM CALL-CACTIVATE.
     MOVE "10" TO ERR-CODE.
 ERR-2.                                        (28)
     PERFORM ERR-3.
     CALL "CFREEDSEG" USING CC,
       EDS-INDEX, EDS-ID.
     MOVE "11" TO ERR-CODE.
 ERR-3.
     MOVE CC TO CC-9.
     DISPLAY " MAIN: ERR-CODE = " ERR-CODE
                  ", CC = " CC-9.
 STOP-RUN.
     STOP RUN.

    DATA AREA IS %047520 WORDS.
    CPU TIME = 0:00:06.  WALL TIME = 0:00:12.
END COBOL/3000 COMPILATION.  NO ERRORS.  NO WARNINGS.


 END OF COMPILE
:PREP $OLDPASS,MAINMAIN;RL=SPLSUB.RL;CAP=IA,BA,DS,PH (29)

 END OF PREPARE
:COBOL P1UPSUB
```

Figure 10. Modified Main Program

Explanations of highlighted items of Figure 10:

19 THESE WORKING STORAGE FIELDS ARE NEEDED BY THE SPL ROUTINES THAT MANAGE THE EXTRA DATA SEGMENT, AND PROVIDE THE PROCESS TO PROCESS COMMUNICATION.

20 THE CALL TO "CGETDSEG" GETS THE EXTRA DATA SEGMENT.

21 THE CALL TO "CCREATE" CREATES A "CHILD" PROCESS TO BE KNOWN AS "SUBMAIN".

```
SUB
  A BYTE STRING CONTAINING THE "CHILD"
  PROCESS'S NAME, "SUBMAIN", TERMINATED BY
  A BLANK.

SUB-PIN
  PROCESS-WIDE "CHILD" PIN (PROCESS
    IDENTIFICATION NUMBER) RETURNED BY
    "CCREATE";
  USED WHEN REFERRING TO THE "CHILD"
    PROCESS IN SUBSEQUENT CALLS;
  MUST BE 1 WORD.

FLAGS
  IF SET TO 1, AS SHOWN, THEN "MAINMAIN" MAY
    BE REACTIVATED AFTER "SUBMAIN" SUSPENDS,
    TERMINATES, OR ABORTS;
  MUST BE 1 WORD.
```

22 THE PREVIOUS SUBROUTINE CALL IS REPLACED BY

23
```
A CALL TO "COMOVOUT" THAT MOVES THE
  PARAMETERS TO BE PASSED FROM WORKING
  STORAGE TO THE SHAREABLE EXTRA DATA
  SEGMENT;
A CALL TO "CACTIVATE" THAT PASSES EXECUTION
  CONTROL TO THE "CHILD" PROCESS "SUBMAIN";
A CALL TO "COMOVIN" THAT MOVES THE RETURNED
  PARAMETERS FROM THE SHAREABLE EXTRA DATA
  SEGMENT TO WORKING STORAGE.
```

THIS SEQUENCE MUST REPLACE EACH CALL TO "SUB".

24 BEFORE "MAINMAIN" TERMINATES, "SUBMAIN" MUST BE ALERTED TO FREE "SUBMAIN"'S CONNECTION TO THE SHAREABLE EXTRA DATA SEGMENT.

25 THE CALL TO "CFREEDSEG" RETURNS THE EXTRA DATA SEGMENT TO MPE; AND, NEED ONLY BE DONE ONCE PER PROCESS.

26 FOR "COMOVOUT" AND "COMOVIN"

```
MAIN-PARM-FIELD.
  WORKING STORAGE FIELD THAT
    CONTAINS THE INFORMATION TO BE
      TRANSFERRED TO THE EXTRA DATA SEGMENT BY
      "COMOVOUT"; OR,
    RECEIVES THE INFORMATION TRANSFERRED
      FROM THE EXTRA DATA SEGMENT BY
      "COMOVIN".
```

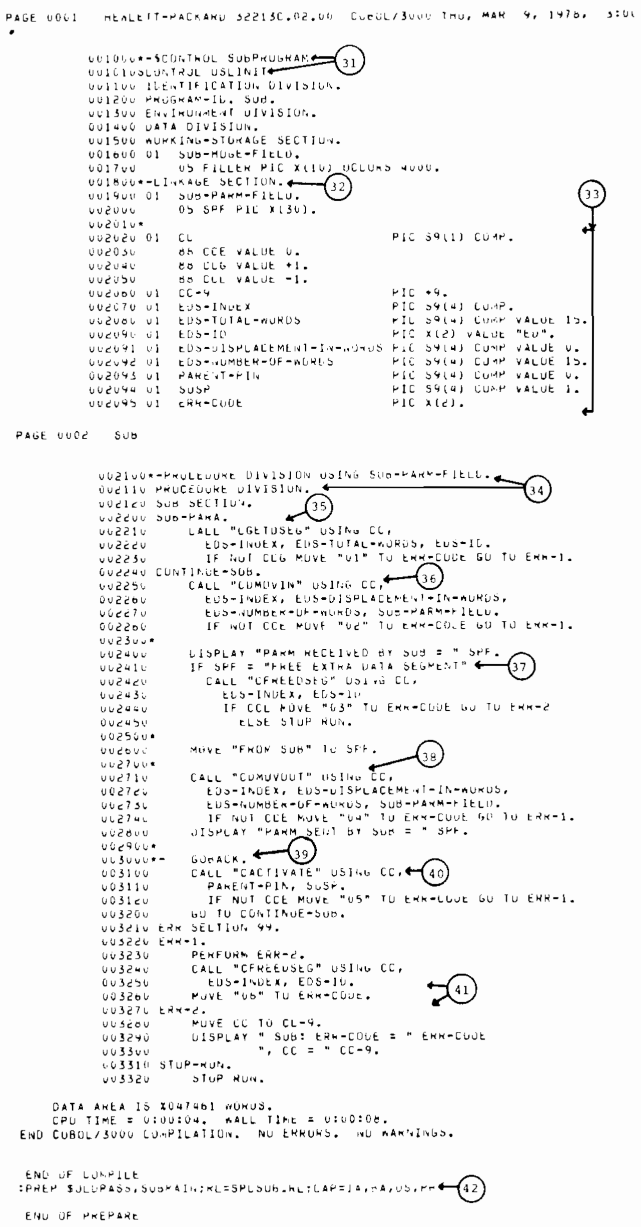
27 FOR "CACTIVATE"

```
SUSP
  IF SET TO 2, AS SHOWN, "MAINMAIN" WILL
    SUSPEND UNTIL THE "CHILD" PROCESS
    "SUBMAIN" SUSPENDS, TERMINATES, OR
    ABORTS;
  MUST BE 1 WORD.
```

28 WHEN AN ERROR OCCURS,
```
  AN ATTEMPT IS MADE TO RETURN THE EXTRA
    DATA SEGMENT (IF PREVIOUSLY GOTTEN);
  ERROR MESSAGES ARE DISPLAYED.
```

29 NOTE THAT THE PREP MUST INCLUDE THE DS AND PH CAPABILITIES. THE USER PERFORMING THE PREP MUST HAVE THE DS AND PH CAPABILITIES. THE GROUP AND ACCOUNT IN WHICH THE PROCESS EXECUTES MUST HAVE THE DS AND PH CAPABILITIES.

Figure 11 shows the modifications to program "SUB-MAIN." Explanations of the highlights of Figure 11 follow:


31
32
33
34
35
36
37
38
39
40
41
42

```
PAGE 0001   HEWLETT-PACKARD 32213C.02.00  COBOL/3000 THU, MAR  9, 1978,  3:00
.
          001000*-$CONTROL SUBPROGRAM
          001010$CONTROL USLINIT
          001100 IDENTIFICATION DIVISION.
          001200 PROGRAM-ID. SUB.
          001300 ENVIRONMENT DIVISION.
          001400 DATA DIVISION.
          001500 WORKING-STORAGE SECTION.
          001600 01  SUB-HUGE-FIELD.
          001700        05 FILLER PIC X(10) OCCURS 4000.
          001800*-LINKAGE SECTION.
          001900 01  SUB-PARM-FIELD.
          002000        05 SPF PIC X(30).
          002010*
          002020 01  CC                            PIC S9(1) COMP.
          002030        88 CCE VALUE 0.
          002040        88 CCG VALUE +1.
          002050        88 CCL VALUE -1.
          002060 01  CC-9                          PIC +9.
          002070 01  EDS-INDEX                     PIC S9(4) COMP.
          002080 01  EDS-TOTAL-WORDS               PIC S9(4) COMP VALUE 15.
          002090 01  EDS-ID                        PIC X(2) VALUE "ED".
          002091 01  EDS-DISPLACEMENT-IN-WORDS     PIC S9(4) COMP VALUE 0.
          002092 01  EDS-NUMBER-OF-WORDS           PIC S9(4) COMP VALUE 15.
          002093 01  PARENT-PIN                    PIC S9(4) COMP VALUE 0.
          002094 01  SUSP                          PIC S9(4) COMP VALUE 1.
          002095 01  ERR-CODE                      PIC X(2).

PAGE 0002   SUB

          002100*-PROCEDURE DIVISION USING SUB-PARM-FIELD.
          002110 PROCEDURE DIVISION.
          002120 SUB SECTION.
          002200 SUB-PARA.
          002210     CALL "CGETDSEG" USING CC,
          002220       EDS-INDEX, EDS-TOTAL-WORDS, EDS-ID.
          002230       IF NOT CCG MOVE "01" TO ERR-CODE GO TO ERR-1.
          002240 CONTINUE-SUB.
          002250     CALL "COMOVIN" USING CC,
          002260       EDS-INDEX, EDS-DISPLACEMENT-IN-WORDS,
          002270       EDS-NUMBER-OF-WORDS, SUB-PARM-FIELD.
          002280       IF NOT CCE MOVE "02" TO ERR-CODE GO TO ERR-1.
          002300*
          002400     DISPLAY "PARM RECEIVED BY SUB = " SPF.
          002410     IF SPF = "FREE EXTRA DATA SEGMENT"
          002420       CALL "CFREEDSEG" USING CC,
          002430         EDS-INDEX, EDS-ID
          002440         IF CCL MOVE "03" TO ERR-CODE GO TO ERR-2
          002450           ELSE STOP RUN.
          002500*
          002600     MOVE "FROM SUB" TO SPF.
          002700*
          002710     CALL "COMOVOUT" USING CC,
          002720       EDS-INDEX, EDS-DISPLACEMENT-IN-WORDS,
          002730       EDS-NUMBER-OF-WORDS, SUB-PARM-FIELD.
          002740       IF NOT CCE MOVE "04" TO ERR-CODE GO TO ERR-1.
          002800     DISPLAY "PARM SENT BY SUB = " SPF.
          002900*
          003000*-   GOBACK.
          003100     CALL "CACTIVATE" USING CC,
          003110       PARENT-PIN, SUSP.
          003120       IF NOT CCE MOVE "05" TO ERR-CODE GO TO ERR-1.
          003200     GO TO CONTINUE-SUB.
          003210 ERR SECTION 99.
          003220 ERR-1.
          003230     PERFORM ERR-2.
          003240     CALL "CFREEDSEG" USING CC,
          003250       EDS-INDEX, EDS-ID.
          003260     MOVE "06" TO ERR-CODE.
          003270 ERR-2.
          003280     MOVE CC TO CC-9.
          003290     DISPLAY " SUB: ERR-CODE = " ERR-CODE
          003300                 ", CC = " CC-9.
          003310 STOP-RUN.
          003320     STOP RUN.

    DATA AREA IS %047461 WORDS.
    CPU TIME = 0:00:04.  WALL TIME = 0:00:08.
END COBOL/3000 COMPILATION.  NO ERRORS.  NO WARNINGS.


 END OF COMPILE
:PREP $OLDPASS,SUBMAIN;RL=SPLSUB.RL;CAP=IA,BA,DS,PH

 END OF PREPARE
```

Figure 11. Modified Subprogram

Explanations of highlighted items of Figure 11:

```
(31) THE SUBPROGRAM $CONTROL STATEMENT IS
     REPLACED BY A MAIN PROGRAM $CONTROL
     STATEMENT.

(32) THE LINKAGE SECTION STATEMENT IS REMOVED
     SINCE PARAMETERS WILL BE PASSED BETWEEN
     "MAINMAIN" AND "SUBMAIN" VIA THE EXTRA DATA
     SEGMENT.

(33) THESE WORKING STORAGE FIELDS ARE NEEDED BY
     THE SPL ROUTINES THAT MANAGE THE EXTRA DATA
     SEGMENT, AND PROVIDE PROCESS TO PROCESS
     COMMUNICATION.

(34) THE SUBPROGRAM PROCEDURE DIVISION STATEMENT
     IS REPLACED BY A MAIN PROGRAM PROCEDURE
     DIVISION STATEMENT.

(35) THE CALL TO "CGETDSEG" LOCATES THE SHAREABLE
     EXTRA DATA SEGMENT PREVIOUSLY GOTTEN BY
     "MAINMAIN".

     NOTE THAT A CONDITION CODE OF CCG IS THE
     "OK" RETURN.

(36) THE CALL TO "CDMOVIN" MOVES THE INCOMING
     PARAMETERS FROM THE EXTRA DATA SEGMENT TO
     WORKING STORAGE.

(37) THIS SENTENCE TESTS WHETHER TO FREE THE
     EXTRA DATA SEGMENT WITH A CALL TO
     "CFREEDSEG".

(38) THE CALL TO "CDMOVOUT" MOVES THE OUTGOING
     PARAMETERS FROM WORKING STORAGE TO THE EXTRA
     DATA SEGMENT.


(39) THE SUBPROGRAM "GOBACK" STATEMENT IS
     REPLACED BY

(40) A CALL TO "CACTIVATE".

     "CACTIVATE" SUSPENDS "SUBMAIN" AND PASSES
     EXECUTION CONTROL BACK TO THE "PARENT"
     PROCESS "MAINMAIN".

     NOTE THAT WHEN "SUBMAIN" IS REACTIVATED,
     EXECUTION WILL BEGIN WITH THE
     "GO TO CONTINUE-SUB" STATEMENT.

     PARENT-PIN
       "PARENT" PIN;
       ALWAYS BINARY 0;
       MUST BE 1 WORD.

     SUSP
       IF SET TO 1, AS SHOWN, "SUBMAIN" WILL
       SUSPEND UNTIL REACTIVATED BY THE "PARENT"
       PROCESS "MAINMAIN".

(41) WHEN AN ERROR OCCURS,
       AN ATTEMPT IS MADE TO RETURN THE EXTRA
         DATA SEGMENT (IF PREVIOUSLY GOTTEN);
       ERROR MESSAGES ARE DISPLAYED.

(42) NOTE THAT THE PREP MUST INCLUDE THE DS AND
     PH CAPABILITIES. THE USER PERFORMING THE
     PREP MUST HAVE THE DS AND PH CAPABILITIES.
     THE GROUP AND ACCOUNT IN WHICH THE PROCESS
     EXECUTES MUST HAVE THE DS AND PH
     CAPABILITIES.
```

Figure 12 shows the execution of "MAINMAIN"; it is executed twice to demonstrate that the extra data segment is obtained from MPE and returned to MPE successfully.

```
:RUN MAINMAIN

PARM SENT BY MAIN = FROM MAIN
PARM RECEIVED BY SUB = FROM MAIN
PARM SENT BY SUB = FROM SUB
PARM RECEIVED BY MAIN = FROM SUB
PARM RECEIVED BY SUB = FREE EXTRA DATA SEGMENT

 END OF PROGRAM
:RUN MAINMAIN

PARM SENT BY MAIN = FROM MAIN
PARM RECEIVED BY SUB = FROM MAIN
PARM SENT BY SUB = FROM SUB
PARM RECEIVED BY MAIN = FROM SUB
PARM RECEIVED BY SUB = FREE EXTRA DATA SEGMENT

 END OF PROGRAM
:EOJ

CPU (SEC) = 21
ELAPSED (MIN) = 2
THU, MAR  9, 1978,   3:00 PM
END OF JOB
```

Figure 12. "MAINMAIN" Execution

Some additional considerations regarding inter-process sharing of extra data segments are worth noting. If the processes sharing extra data segments are concurrently in execution, RINs (Resource Identification Numbers) should be used to control access. That is, before modifying an extra data segment, a RIN directed test should be made to determine if any other process is currently modifying the same extra data segment thereby preventing simultaneous (i.e., erroneous) updating.

● ● ●

Tips and techniques

Principles for Optimizing Performance of On-Line Programs

by Robert M. Green
ROBELLE Consulting Ltd.
#130-5421 10th Avenue
Delta, B.C. V4M 3T9
Canada (604) 943-8021

I apply five general principles in optimizing on-line programs:

- Make each disc access count.
- Maximize the value of each terminal input.
- Minimize the run-time program size.
- Avoid constant demands for execution.
- Optimize for the common events.

FIRST PRINCIPLE: MAKE EACH DISC ACCESS COUNT

Disc accesses are the most critical resource on the HP 3000. The system is capable of performing about 30 disc transfers per second and they must be shared by system processes (spooling, console operator), memory management and user programs. (This rate can be increased to 58 per second under the best circumstances and degrade to 24 per second when randomly accessing a large file.) Another interesting fact is that a 4096 word transfer takes about the same overhead as a 128 word transfer. Therefore it is better to read 4096 words in one transfer than to read 128 words 32 times.

Some of the operations that consume extra disc accesses on the HP 3000 are:

Increasing the number of keys in a detail dataset, thus causing IMAGE to access an extra master dataset on each DBPUT.

Increasing the program stack size by 8,000 bytes, thus causing the MPE memory manager to perform extra swapping disc accesses to find room in memory for the larger stack.

Improperly segmenting an active program, causing many absence traps to the memory manager to bring the code segments into main memory.

Defining a database or KSAM file with overly large blocksize, thus forcing each user terminal to access a large extra data segment that must be swapped in and out of main memory.

NOBUF Disc Accesses

When designing your next on-line application, see if there is some way that a random disc file can be used instead of an IMAGE dataset or a KSAM file. Then open that file with NOBUF and access it via directed reads and writes to specific blocks. Normally when you open a file, the program is assigned an extra data segment to hold the buffer space for the file. Transfers between the file and the program are always done through this extra data segment. When the program requires a record, MPE first checks to see if the record is already in the extra data segment buffers; if so, it is merely transferred from the extra data segment to the user stack. If the block containing the desired record is not in the buffers, MPE issues a read against the disc to bring the block into main memory.

Although this sounds very clever and efficient, it has one major flaw: the extra data segment itself can be swapped. This means that in order to do any file access on a busy system, it may be necessary to read the extra data segment into memory before accessing the data in the disc file. On a heavily loaded system this could cause a large number of unnecessary disc transfers. NOBUF access does away with all this by providing a direct interface between the user program and the disc files. Blocks are transferred to and from the user stack and the disc without any intervening buffer area. NOBUF is the fastest way to use random disc storage from a user program.

The user program must provide its own buffer space in the stack and call for transfers of data via the block number within the file. When multi-record access is used, it is possible to transfer multiple blocks at a time. The user is responsible for determining which block contains the record that he desires and where within the block the record is located. Simple subroutines can be written to handle this transformation.

A typical use for this kind of file is as a data entry transaction file. As the operator enters the data, it is buffered in the stack until a block is full; then the entire block is written to the disc in one operation. For even better throughput and response time, you might try writing the blocks to the disc with the NO-WAIT option; when this is used, MPE overlaps the write operation to the disc with your next print and read from the terminal. Without NO-WAIT your program would be suspended until the disc write could be completed by MPE.

Warning: Be certain that you know when the end-of-file is updated, otherwise you might find that you have an empty transaction file when the system crashes. I suggest that you move the end-of-file to the limit of the file at the start of the day by writing a null entry in the last record position and then closing the file.) When the transaction file is full (or the day ends), a batch program is used to put the transactions into the final IMAGE dataset or KSAM file. This job can be done in low priority or after hours.

SECOND PRINCIPLE: MAXIMIZE THE VALUE OF EACH TERMINAL READ

Each time a program reads from the terminal it is suspended and may be swapped out of memory. When the operator hits the carriage return key, the input operation is terminated and the process must be dispatched again. In order to dispatch a process, MPE must ensure that the data stack and at least one code segment are resident in main memory. If the process is going to access the disc, it may be necessary to make an extra data segment resident also. Unless the computer has enough main memory so that no user segments are ever swapped out, it is desirable to have the process set as much work done as possible before it suspends for the next terminal input (and is swapped out again).

The simplest way to program data entry applications is by prompting for and accepting only one field of data at a time. This is also the least efficient way to do it. The user data stack must be made resident every time the user hits 'return.' (Therefore, the less often the user hits 'return,' the larger your stack can afford to be.) Since it is inefficient, fast response time cannot be guaranteed and the resulting delays are very irritating to operators. They can never work up any input speed, because they never know when the computer is ready for the next input line. If response time and throughput are the only considerations, it is always preferable to keep the operator typing as long as possible before hitting the 'return' key. Multiple transactions should be allowed per line with suitable separators and multiple lines should be allowed without a 'return.'

THIRD PRINCIPLE: MINIMIZE THE RUN-TIME PROGRAM SIZE

The HP 3000 is an ideal machine for optimizing because of the many hardware features available at run-time to minimize the effective size of the program. Even quite large application systems (6000 lines of code) can be organized to consume only a small amount of main memory at any one time. Each executing process on the HP 3000 consists of a single data segment called the "stack" and one or more extra data segments for system storage such as file buffers. Although a process is always executing some code in a code segment, the code does not properly belong to the process, since it can be shared by all processes in the system.

Large programs which are not logically segmented make it harder for the memory manager to do its job and thus cause many disc accesses to be consumed in swapping. In an extreme case, the system can almost be brought to a complete standstill by a very large program executing on many terminals at the same time.

Many more terminals can be supported on a given system if data stack sizes are kept modest (ex: less than 6000 bytes on a 192K byte machine) and the code is properly segmented (ex: all segments about the same size, say 4K bytes on a 192K byte machine). The simplest way to keep the stack small is to make all data variables local (DYNAMIC in COBOL) and to use global storage only for buffers and control values that must be accessed by all subroutines. The reason that this is so effective is that dynamic local storage is allocated on the top of the stack when the subroutine is entered and is released automatically when the subroutine is left. This means that if the main program calls 3 large subroutines in succession, they all reuse the same space in the stack. The stack need only be large enough for the deepest nesting situation.

Since the amount of dynamic stack space that will be required by the program is not known at the start of execution, the 3000 provides methods (both automatic and programmatic) to expand the dynamic area. Whenever a stack overflow occurs, MPE automatically allocates more space (up to a MAXDATA limit). Unfortunately, there is no automatic mechanism for reducing the stack size when that additional space is no longer needed. The user application program can include a check in the mainline and shrink the stack back down to the desired size after returning from an oversize subroutine. (See Appendix for an example.)

The other major way to reduce the size of a data stack is to ensure that constant data items (such as error messages, screen displays) are stored in the code segment instead of the data segment. Since they are never to be modified, there is no logical reason that they must be in the data stack. By moving them to the code segment, one copy of them can be shared by all users running the program. In SPL this is done by including =PB in a local array declaration or MOVE'ing a literal string into a buffer. In COBOL constants can be moved to the code segment by DISPLAY'ing literal strings in place of declared data items. In FORTRAN both FORMAT statements and DISPLAY'ed literals are stored in the code.

FOURTH PRINCIPLE: AVOID CONSTANT DEMANDS FOR EXECUTION

The HP 3000 is a multiprogramming, virtual memory machine that depends for its effectiveness on a suitable mix of processes to execute. Although the sizes of the segments to be swapped have an effect on performance, this is dependent upon the frequency with which memory residency is demanded. Given the same overall configuration and application program sizes, the system supports many more terminals if each one only executes for 5 seconds every 30 seconds than if each one must execute for 60 seconds at a time. Each additional terminal that is demanding continuous execution (in high priority) makes it harder for the operating system to provide proper response time to all other terminals.

Here are some examples of the kind of operation that can destroy response time if performed in high priority:

EDIT/3000, a GATHER ALL of a 3000 line source file.

QUERY, serial read of 100,000 records.

SORT, sorting 50,000 records.

COBOL, compiling on 4 terminals at once.

All of these operations should be done in low priority in batch STREAM jobs. These jobs can even be created dynamically by on-line programs. In this way, the on-line user still requests the high-overhead operation, but the system fulfills the request when it has the time.

FIFTH PRINCIPLE: OPTIMIZE FOR THE COMMON EVENTS

In any application where there is a large variation between the minimum and maximum load that a transaction can cause, the program should be optimized around the most common size of transaction. In any application with a large number of on-line functions, it is likely that a small number of functions are used most of the time. In this case, all optimization efforts should be aimed at the commonly used functions and all others left as is. This is especially feasible on the HP 3000 because of code segmentation and dynamic stacks.

If N is the average number of records in a transaction (i.e., the number of lines on a customer order, maximum is 500), then allow room in your stack for N records. If you only allowed for one record, then there would be unneeded disc thrashing. Alternatively, if you provide room for the maximum number, then the data stack is much larger than actually needed most of the time. Having a larger data stack may cause the system to overload, eliminating the benefits of keeping the records in your stack. It is recommended that room in the stack be allowed for slightly more than the average number and that a NOBUF disc file be used to "page" this area on very large transactions.

Optimizing Case Study: QEDIT

QEDIT is a high-speed, low-overhead source program editor developed by Robelle Consulting Ltd. The primary objective of QEDIT is to provide the fastest possible editing with the minimum possible system load. Other objectives include conservation of disc space, similarity to EDIT/3000 in command syntax, ability to recover the workfile following a system crash or program abort and increased programmer productivity.

QEDIT and the First Principle: Disc Accesses

In order to reduce disc accesses, QEDIT had to eliminate the overheads of the TEXT, KEEP and GATHER ALL commands of EDIT/3000. These three operations have the most drastic impact upon the response time of the other users. QEDIT attacks the problem of KEEPs by providing an interface library that fools the HP compilers into thinking that a QEDIT workfile is really a "card image" file. As a result, it is never necessary to KEEP a workfile before compiling it. Since KEEPs are never used, most TEXTs are eliminated. TEXT is only needed when you want to make a backup or duplicate copy of an existing file. It was anticipated that most users would maintain their source files exclusively in workfile format, so the TEXT'ing of workfiles was optimized (by using NOBUF, multi-record techniques) to be at least 4 times faster than a normal TEXT of a card image file. The GATHER ALL operation is slow because it makes a copy of the entire workfile in another file. QEDIT renumbers up to 12 times faster by doing without the file copy.

Disc accesses during interactive editing (add, delete, change, etc.) were minimized by packing as many contiguous lines as possible into each disc block. The resulting workfile is seldom over 50% of the size of a normal KEEP file or 25% of the size of an EDIT/3000 K-file (workfile). Most QEDIT users maintain all of their source programs in workfile form, since this saves disc space, simplifies operations (there need only be one copy of each version of a source program) and provides optimum on-line performance.

QEDIT always accesses its workfile in NOBUF mode and buffers all new lines in the stack until a block is full before writing to the disc. Wherever possible in the coding of QEDIT, unnecessary disc transfers have been eliminated. For example, the workfile maintains only forward direction linkage pointers, which reduces the amount of disc I/O substantially. Results of a logging test show that reducing the size of the workfile and eliminating the need for TEXT/KEEP reduces disc accesses and CPU time by 70-90%.

QEDIT and the Second Principle: Terminal Accesses

QEDIT allows multiple commands per line, plus multiple data lines per data line input (i.e., you can enter 7 lines of text without hitting 'return'). All interaction with the terminal is done directly through the READX and PRINT instrinsics.

QEDIT and the Third Principle: Program Size

QEDIT is a completely new program, written in highly structured and procedurized SPL. The resulting program file consists of 7 code segments of 1780 words (decimal) each. Only two code segments are required for most editing commands, while the most common function (adding new lines) requires only one code segment most of the time.

QEDIT uses a minimum data stack and no extra data segments. All error messages are contained in the code, isolated in a separate code segment that need not be resident if you make no errors.

QEDIT and the Fourth Principle: Constant Demands

Most QEDIT commands are so fast that they are over before a serious strain has been placed on the host machine. For example, a 2000 line source program can be searched for a string in four seconds. For those operations which still are too much load, QEDIT provides the ability to switch priority sub-queues dynamically. In fact, the system manager can dictate a maximum priority for certain operations such as compiles or TEXT and KEEP commands.

QEDIT and the Fifth Principle: Common Events

The entire design of QEDIT is based on the observation that program editing is not completely random. When a programmer changes line 250, he is more likely to require access to lines 245 through 265 next than he is to lines 670 through 710. This observation dictated the design of the indexing scheme for the QEDIT workfile.

There are many examples of optimizing for the most common events in QEDIT: the blocksize will hold about a screenful of data lines, built-in compiler, fast renumbering command (600 lines per second) in place of a GATHER command, faster TEXTing of workfiles than KEEP files (4 to 7 times faster).

Results of Applying the Principles:

In less than 7 seconds, QEDIT can text 1000 lines, renumber them and search for a string. Commands are 80% to 1200% faster than EDIT/3000, program size is cut in half, and disc I/O and CPU time are reduced by up to 90%.

Programming of QEDIT began in March 1977 and user-site testing in September 1977. At the present time (June 1978) there are 13 QEDIT user installations. QEDIT shows what can be accomplished by applying all of these optimizing principles in the design of one system. In any given application system it may not be possible to take advantage of all five principles, but to whatever extent they can be applied, the resulting system will provide better service than it would have.

REFERENCES: PUBLISHED MATERIAL ON OPTIMIZING

```
[1]  COMMUNICATOR No. 14.
       Page 87, Block/Page mode problems.
[2]  COMMUNICATOR No. 12.
       Segmentation in COBOL
[3]  COMMUNICATOR No. 5.
       Segmentation for Maximum Efficiency
          of System-Type Programs.
[4]  JOURNAL-3000 Vol 1, No. 6.
       KSAM vs. IMAGE
       HP 3000 with Front End Processor
       FORTRAN Optimization
[5]  JOURNAL-3000 Vol. 1, No. 5.
       QEDIT, Quick Program Editing,
           Small Appetite for Computer Time.
[6]  JOURNAL-3000 VOL. 1, NO. 4.
       Using Extra Data Segments.
       Common Programming Errors with IMAGE/3000.
[7]  CONTRIBUTED LIBRARY, Vol I/II
        IDEA Program
        IDEAII Program
        RESP Program
        IDLE Program
        PROGSTAT PROGRAM
[8]  CONTRIBUTED LIBRARY, Vol III
        SOO Program
[9]  CONTRIBUTED LIBRARY, June 1978.
        DBSTAT Program
        DBCHANGE Program
[10] SCRUG MEETING LIBRARY, March 1978.
[11] SCRUG MEETING NOTES, March 1978.
        Extra Data Segments and Process Handling
        Operator Utilities
[12] INTERNATIONAL USERS MEETING, 1977.
       KSAM (see extra data segment size, load times)
       IMAGE  for the advanced User
       Optimizing FORTRAN IV/3000
       RPG/3000 Programming Optimization
       Data Entry Techniques
       Segmentation
       MPE II Measurement and Optimization
       MPE C Measurement and Optimization
[13] INTERNATIONAL USERS MEETING, February 1975
       Software Optimization Through Segmentation
[14] INTERNATIONAL USERS MEETING, May 1974
       Program Performance
[15] CCRUG MEETING MINUTES, May 9, 1978
       IDEA Program
       DBDRIVER Program
```

APPENDIX: SHRINKING THE STACK SIZE

```
$CONTROL LIST,SUBPROGRAM,MAIN=LIBSEG1,ERRORS=9
BEGIN
<<         CHECKSTACK LIBRARY SUBROUTINES
             ROBELLE CONSULTING LTD.
PURPOSE =  CHECK FOR EXCESSIVE DYNAMIC STACK SPACE AFTER
           SUBPROGRAM CALLS AND ADJUST.  THIS PACKAGE CONSISTS
           OF THREE ROUTINES THAT ARE INTENDED TO BE CALLED
           FROM THE MAINLINE OF AN APPLICATION PROGRAM THAT
           CALLS SEVERAL SUBPROGRAMS OF VARYING SIZES.
CONTAINS 3 ROUTINES: CHECKSTACK1, CHECKSTACK2, CHECKSTACK3.
PARAMETERS =  WORKSPACE, 20 BYTES OF GLOBAL WORKSPACE IN THE
  CALLING PROGRAM. THE PROPER COBOL WORKING-STORAGE DEFINITION IS:
        01 CHECK-STACK-SPACE .
           05  PRINT-RESULTS-FLAG  PIC  S9(3) COMP VALUE N.
      *          N=0(NO PRINTOUT),1(ON TERMINAL),
      *            2(ON CONSOLE),3(ON BOTH).
           05  FILLER                  PIC X(18).
HOW TO USE =
  1. ADD THE WORKSPACE TO THE DATA DIVISION OF YOUR PROGRAM
     AND SET THE DESIRED PRINT'FLAG VALUE(SEE STEP 4).
  2.  AT START OF PROGRAM,
    CALL 'CHECKSTACK1' USING CHECK-STACK-SPACE.
       THIS CALL SHOULD OCCUR ONCE AT THE START OF THE
       MAINLINE.  THE PURPOSE IS TO RECORD THE SIZE OF
       THE DYNAMIC STACK AREA BEFORE ANY SUBPROGRAMS ARE
       CALLED.  THIS SIZE IS DETERMINED BY STACK=XXXX IN THE
       :PREP OR :RUN COMMANDS.
  3.  AFTER EACH SUBPROGRAM CALL,
    CALL 'CHECKSTACK2' USING CHECK-STACK-SPACE.
       THIS CALL COMPARES THE CURRENT DYNAMIC STACK AREA
       WITH THE INITIAL SIZE AND IF IT IS OVER 512 WORDS
       LARGER (1024 BYTES), REDUCES IT BACK TO THE INITIAL.
  4.  AT THE END OF THE PROGRAM,
    CALL 'CHECKSTACK3' USING CHECK-STACK-SPACE.
       THIS CALL PRINTS STATISTICS ON STACK USAGE ON
       EITHER $STDLIST OR THE CONSOLE OR BOTH. FORMAT=
         GLOB99  STK99  #OK99  AVG99   #ADJ99    SIZ99
         ^GLOBAL STACK SIZE IN DECIMAL WORDS
                 ^ INITIAL DYNAMIC STACK SIZE
                        ^ NUMBER OF 'OK' SUBPROGRAM CALLS
AVERAGE WORDS OF STACK PER CALL ^
     NUMBER OF TIMES STACK WAS ADJUSTED ^
         AVERAGE SIZE OF THE STACK WHEN ADJUSTED ^

START WITH THE DEFAULT VALUE FOR STACK=(ABOUT 800) AND A LARGE MAXDATA=.
IF ALL OF THE SUBPROGRAM CALLS ARE ADJUSTED (I.E., OK=0), INCREASE THE
STACK= VALUE.  TRY TO FIND A VALUE WHERE MOST OF THE SUBPROGRAMS CAN
EXECUTE WITHOUT HAVING TO SHRINK THE STACK AFTERWARDS, BUT NOT SO LARGE
THAT THERE ARE NO LARGE SUBPROGRAMS TO ADJUST.
>>
```

```
PROCEDURE  CHECKSTACK1 ( BUF ) ;
    INTEGER ARRAY   BUF;
BEGIN
<< DEFINE STRUCTURE/USE OF BUF >>
DOUBLE ARRAY DBUF (*) = BUF;
DEFINE
  PRINT'FLAG    = BUF#        ,INITIAL'SPACE   = BUF(1)#
 ,SHRINK'COUNT = BUF(2)#   ,OK'COUNT           = BUF(3)#
 ,OK'SPACE   = DBUF(2)#     ,SHRINK'SPACE     = DBUF(3)#
 ;
INTEGER Z,Q;

IF NOT ( 0<=PRINT'FLAG<=3 ) THEN
  PRINT'FLAG := 1; <<DEF>>

PUSH (Z,Q);   Z:=TOS; Q:=TOS;

INITIAL'SPACE := Z - Q;
BUF(2) := 0;
MOVE BUF(3) := BUF(2),(7);

END; <<CHECKSTACK1 >>


PROCEDURE CHECKSTACK2 ( BUF ) ;
    INTEGER ARRAY   BUF;
BEGIN
<< DEFINE STRUCTURE/USE OF BUF >>
DOUBLE ARRAY DBUF (*) = BUF;
DEFINE
  PRINT'FLAG    = BUF#        ,INITIAL'SPACE   = BUF(1)#
 ,SHRINK'COUNT = BUF(2)#   ,OK'COUNT           = BUF(3)#
 ,OK'SPACE   = DBUF(2)#     ,SHRINK'SPACE     = DBUF(3)#
 ;
INTEGER Z, Q, STACKSIZE;
INTRINSIC ZSIZE;

PUSH ( Z,Q ); Z:=TOS; Q:=TOS;
STACKSIZE := Z - Q;
IF STACKSIZE > (INITIAL'SPACE + 512) THEN BEGIN
  ZSIZE ( Q + INITIAL'SPACE );
  SHRINK'COUNT := SHRINK'COUNT + 1;
  SHRINK'SPACE := SHRINK'SPACE + DOUBLE(STACKSIZE);
  END
ELSE BEGIN
  OK'COUNT := OK'COUNT + 1;
  OK'SPACE := OK'SPACE + DOUBLE(STACKSIZE);
  END;

END; << CHECKSTACK2>>


PROCEDURE CHECKSTACK3 ( BUF) ;
    INTEGER ARRAY   BUF;
BEGIN
<< DEFINE STRUCTURE/USE OF BUF
DOUBLE ARRAY DBUF (*) = BUF;
DEFINE
  PRINT'FLAG    = BUF#        ,INITIAL'SPACE   = BUF(1)#
 ,SHRINK'COUNT = BUF(2)#   ,OK'COUNT           = BUF(3)#
 ,OK'SPACE   = DBUF(2)#     ,SHRINK'SPACE     = DBUF(3)#
 ;
INTEGER ARRAY P(0:38);
BYTE ARRAY P'(*)=P;
INTEGER TERMINAL;
INTEGER GLOBAL'SPACE;
INTRINSIC PRINT,PRINTOP, ASCII,DASCII,WHO,DATELINE;

IF PRINT'FLAG = 0 THEN RETURN;

IF PRINT'FLAG=2 OR PRINT'FLAG=3 THEN BEGIN
     PRINT IDENTIFYING MESSAGE ON THE CONSOLE >>
  P:="  "; MOVE P(1):=P,(38);
  MOVE P :="CHECK-STACK:";
  WHO(,,,P'(12),P'(21),P'(30),,TERMINAL);
  MOVE P'(39) := "ON";
  ASCII(TERMINAL,10,P'(42));
  P'(20):= P'(29):=".";
  PRINTOP(P, 46,0);
  END;

P:="  "; MOVE P(1):=P,(38);
MOVE P:="GLOB";
PUSH (Q);
GLOBAL'SPACE := TOS;
ASCII(GLOBAL'SPACE,10,P'(4));
MOVE P'(10):="STK";
ASCII(INITIAL'SPACE,10,P'(13));
MOVE P'(19):="#OK";
ASCII(OK'COUNT,10,P'(22));
MOVE P'(28):="AVG";
DASCII(OK'SPACE/DOUBLE(OK'COUNT),10,P'(31));
MOVE P'(37):="#ADJ";
ASCII(SHRINK'COUNT,10,P'(41));
MOVE P'(47):="SIZ";
DASCII(SHRINK'SPACE/DOUBLE(SHRINK'COUNT),10,P'(50)),

IF PRINT'FLAG=2 OR PRINT'FLAG=3 THEN
  PRINTOP(P, 56,0);
IF PRINT'FLAG=1 OR PRINT'FLAG=3 THEN
  PRINT(P,-56,0);

END;   <<CHECKSTACK3
END  <<LIBRARY    .
```

● ● ●

Image Optimization Checklist

by Geoff Walker
Hewlett-Packard Co.
Eastern Sales Region
Paramus, New Jersey

1. Spend time up front on the design. Analyze the application thoroughly. Consider needs vs. wants; batch vs. on-line processing. Consider using a separate transaction data base. Relate data sets to functions.
2. Use "IDEA" (in the contributed library) early in the game to estimate load time, response time, and throughput.
3. Use "DBDRIVER" (in Pub. Sys) on existing data bases to get timings for each Image intrinsic.
4. Minimize the number of paths in the data base. Realize that Query flexibility must be traded off against update speed. Consider batch retrieval for non-time critical reports.
5. Specify the most frequently accessed path as the primary path. Periodically dump and reload the data base to maintain entries on the primary path in physical proximity.
6. Minimize the use of sorted chains. If the application demands sorted chains, keep the chains short. Sort data before loading the data base. Use Sort/3000 on data copied to MPE files for batch reporting.
7. Use a prime number for master data set capacity; don't exceed 80% of capacity in order to minimize synonym chains.
8. Vary the blocksize via the $Control Blockmax parameter to tune the data base for the environment. Consider memory utilization, the number of users on the system, the nature of data base access (serial/chained vs. highly random calculated), and finally disc space utilization.
9. Spread data sets among disc drives to minimize head contention by using MPE Restore with the DEV parameter.
10. When coding, minimize the number of Image calls. Save data for re-use; save record pointers for repeated access.
11. Use the DBGET list options, particularly "*".
12. Reading an entry with write access is faster than reading it with read access. If the application design permits it, consider this both in schema design and in application coding. However, remember that reading a data set with write access updates the "last modified" date in the file label and thus causes the data set to go out on incremental backup tapes.
13. Minimize no-hits when using DBFIND. Don't design a DB access application which will find a match only 3% of the time - instead aim for the highest possible hit ratio.
14. Don't do a DBDELETE & DBPUT when you could do a DBUPDATE instead. Consider flagging records on-line and then doing the actual deletion in a batch processing run.
15. Use the mode 3 "rewind" function of DBCLOSE instead of closing and re-opening the data base.
16. Don't use locking needlessly. Within timing constraints, write code which accomplishes as much as possible while the data base is locked - don't just surround each Image call with lock and unlock calls.
17. Image isn't always the best solution - consider alternate data management methods: KSAM (INDEX), or MPE Files.

• • •

About HP's New VIEW/3000

by Jutta Kernke
Hewlett-Packard - General Systems Division
5303 Stevens Creek Blvd.
Santa Clara, CA 95050

This new product was introduced by Hewlett-Packard in September 1978 and is the result of the implementation of new advances in data entry technology and suggestions gathered from more than 600 current DEL/3000 users. DEL/3000 is the Data Entry Library introduced by HP two years ago; VIEW/3000 is not an enhanced version of DEL/3000, but a completely new product.

Whereas DEL/3000 offered a basic set of data entry capabilities to improve user productivity in the development of on-line transaction processing, VIEW/3000 takes advantage of a new design approach and provides much more powerful forms-creation and data handling facilities than DEL/3000.

Data entry applications have traditionally been designed using a high-level programming language such as COBOL. Data Editing was accomplished through the use of user-specified edit routines. This process was very costly in system development and overall productivity. With VIEW/3000, screen formats are created by simply drawing them in an interactive fashion on the CRT and selecting edit routines from a standard set. Implementation does not require programming effort or extensive training.

VIEW/3000 can help users implement straightforward interactive data entry tasks more easily and efficiently, and can facilitate the development of more complex terminal-oriented applications through the use of a high-level program interface.

Designed both as a stand-alone source data entry facility that can be implemented without programming effort and as a "front-end" to transaction processing applications, VIEW/3000 provides four important features:

- A FORMS DESIGN FACILITY utilizing most HP 264X terminals, allows the creation of interactive screens from "fill-in-the blanks" menus and the use of function keys. Simple edits are accepted by standard defaults and comprehensive data editing and validation can be specified by the use of a free-form field of definition language.

- A SOURCE DATA ENTRY FACILITY that allows immediate on-line entry and modification of data through forms created with the Forms Design Facility.

  This facility controls the flow of forms to a terminal, edits and validates the input data, and records that data in a special file. The same facility allows the terminal operator to "browse" through this file and to modify already entered data as desired. This entire process is completed without the need to write a single program, and it makes data collection easier to implement on an HP 3000 computer system.
- A DATA REFORMATTING FACILITY to change the format of entered data to meet the input requirements of existing application programs.
- A PROGRAM INTERFACE which aids efficient and easy implementation of forms oriented interfaces for transaction processing applications. This library of high-level procedures is available to provide a simple programmatic interface between an application program on the HP 3000 computer system, the terminal, the forms and edits created by the forms design facility, the entered data, and the data file.

With the availability of VIEW/3000, Hewlett-Packard will discontinue sales of DEL/3000.

In accordance with standard policy, although DEL/3000 will not be sold to new customers, Hewlett-Packard will provide support of the product for five years, until January 1, 1984. This is important for existing users who wish to continue to use DEL/3000 for new applications development, or for those who choose not to use DEL/3000 for new development but wish to continue to use their existing DEL/3000 applications; both Comprehensive Software Support and Software Subscription Service are offered. If you choose to convert your existing applications to VIEW/3000, and no longer desire support for DEL/3000, you may at any time cancel your software contract for DEL/3000.

For those users who are currently developing applications with DEL/3000 and choose to begin developing new applications exclusively with VIEW/3000, Hewlett-Packard will provide a full trade-in value for the DEL/3000 initial payment of $300, or $900 if you had purchased the prepaid option, to be credited toward the $1500 list price, of VIEW/3000. The value of this trade-in applies to the list price of VIEW/3000 prior to discounts, and extends until April 1, 1979.

The right to use VIEW/3000 requires HP's standard software purchase agreement for either 12 or 48 months.

Hewlett-Packard's number one objective is customer satisfaction. The standard support policy, which guarantees the availability of support services for five years after discontinuing marketing a product, protects the software investment you may have made with DEL/3000. At the same time, by offering the full trade-in value of DEL/3000 for VIEW/3000, Hewlett-Packard presents an opportunity to take advantage of the additional benefits of VIEW/3000.

If you have any questions concerning either DEL/3000 or VIEW/3000, contact your local HP sales representative.

● ● ●

A One Line Program

For those of us interested in the power of SPL, here is a one-line program,

by Ross Scroggs
Alter * Ability
154 Laidley Street
San Francisco, CA 94131

```
00001000  00000 0   begin
00002000  00000 1
00003000  00000 1   logical array
00004000  00000 1       str'(0:35);
00005000  00000 1
00006000  00000 1   byte array
00007000  00000 1       str(*)=str';
00008000  00000 1
00009000  00000 1   intrinsic
00010000  00000 1       dascii,dbinary,print,readx;
00011000  00000 1
00012000  00000 1   print(str',-dascii( dbinary(str,readx(str',-72))
00013000  00007 1                   *dbinary(str,readx(str',-72)),10,str),%40)
00014000  00025 1
00015000  00025 1   end.
 PRIMARY DB STORAGE=%002;    SECONDARY DB STORAGE=%00044
 NO. ERRORS=0000;            NO. WARNINGS=0000
 PROCESSOR TIME=0:00:00;     ELAPSED TIME=0:00:21

 END OF COMPILE

 END OF PREPARE

-24
125
-3000

 END OF PROGRAM
:save $oldpass,oneline
:
```

● ● ●

User-Defined Command Listing Program

by Pete Fratus
Senior Programmer
Futura Systems, Inc.
P. O. Box 3485
Austin, Texas 78764

When the System Manager supplies one main User-Defined Command (UDC) file in PUB.SYS for most system users, it sometimes becomes necessary to know who has SETCATALOG on that file. This program lists the UDCs set by each user on the system.

```
CMD
  10 FILES COMMAND.PUB.SYS
  20 DIM P$[4],N$[26]
  30 ON END #1 THEN 999
  40 LET I=2
  50 READ #1,I;P$,N$
  60 LET J=NUM(P$[2;1])
  70 LET K=NUM(P$[4;1])
  80 IF K=1 THEN GOTO 110
  90 LET I=I+1
 100 GOTO 50
 110 PRINT LIN(1),N$[1;16]
 120 LET J=J+1
 122 READ #1,J;P$,N$
 130 PRINT TAB(16),N$
 140 LET J=NUM(P$[2;1])
 150 IF J<>0 THEN GOTO 120
 160 LET I=I+1
 170 GOTO 50
 999 STOP
```

```
The program reads COMMAND.PUB.SYS and prints the following list:
```

```
MARIA    SYS
                   UDC06.PUB.SYS

USER     YKLRN
                   UDC06.PUB.SYS

BILLC    SYS
                   UDC05.PUB.SYS

RETA     SYS
                   UDC06.PUB.SYS

MANAGER  UFUT
                   UDC06.PUB.SYS

KENT     SYS
                   UDC06.PUB.SYS

USER     SYS
                   UDC06.PUB.SYS

MANAGER  YIGA
                   UDC06.PUB.SYS

MGRNC    WANTADS
                   UDC02.BOOK.WANTADS
                   UDC06.PUB.SYS

MANAGER  WFUT
                   UDC06.PUB.SYS

                         . . .
```

● ● ●

HP 3000 Software: Keeping Current

by Tom Simon
HP General Systems Division
5303 Stevens Creek Blvd.
Santa Clara, CA 95050

You have all probably heard of the Software Status Bulletin, the COMMUNICATOR 3000, and the Installation Tape Note files. You may also know that all of these publications deal with developments — particularly in connection with software — in the HP 3000 environment. Now, what you may not know is which of these offer you the most usable information; that is, which are prepared from a perspective similar to yours. To help you decide which publications fit your information needs, this article provides a brief description of each publication's orientation and of how it is related to the others.

Note Files

Users of HP 3000 computer systems receive periodic updates to the Multiprogramming Executive Operating System (MPE) and certain program and diagnostic files through the Installation Tape (IT). This magnetic tape contains the cold load version of MPE, the system and subsystem program files in the account SYS, and the necessary diagnostics in the support account. Some of these files are referred to as Note or "noon" (* - explanation at end of article) files which consist of written descriptions of how the current version of MPE differs from the previous version.

COMMUNICATOR 3000

The COMMUNICATOR exists primarily to provide users with a printing of the Note files a week or two in advance of their release through the Installation Tape. The purpose of this early release is to help users anticipate the effect which the updated MPE could have on their operations. Thus, the schedule for publishing the COMMUNICATOR approximates that of the updates to MPE, preceding them slightly.

The COMMUNICATOR also contains articles of general interest to HP 3000 users: programming tips, announcements of new products, and descriptions of changes (updates, new editions, new publications) to manuals. Occasionally, COMMUNICATORs are published when there are no corresponding MPE updates. These issues contain only articles. And because of the demand for back issues of the COMMUNICATOR, some articles from early issues may be revised and reprinted.

Software Status Bulletin

The Software Status Bulletin (SSB) is published twice each month and documents known problems with HP 3000 supported software. Of the three publications mentioned here, it contains the most current information regarding the status of problems (and work-arounds) with HP software. For keeping track of particular problems, or for checking if some difficulty you've encountered has been verified as a software bug, the Software Status Bulletin is the place to look.

Software problems submitted by HP 3000 users, field system engineers, software developers, and quality assurance engineers are assigned Software Maintenance Request (SMR) identification numbers. Those Software Maintenance Requests which are verified to document actual software bugs are also referred to as Known Problem Reports (KPR) and are listed in the Software Status Bulletin. The process goes something like this:

```
Software Problem
     I
     I-----> Software Maintenance Request
                  I
                  I-----> Known Problem Report
                               I
                               I-----> Software Status Bulletin
```

These verified software problems are listed in the Software Status Bulletin by their Software Maintenance Request number and by product, and are designated as open or closed. An open problem is one which has no current fix (a work-around may be indicated); a closed problem has a fix which will be available in a specified version of the software product. After being published as a closed problem for two successive Installation Tape distributions, a Software Maintenance Request is deleted from the bulletin. Open problems are listed until resolved.

Note that not every problem submitted is listed in the Software Status Bulletin, only new and verifiable problems. Duplicates, misunderstandings, and so forth, are not entered. However, a response is made to each problem report submitted, indicating whether the bug has been verified and, if so, what is being done about it.

How They Fit Together

Since the COMMUNICATOR, the Installation Tape and the Software Status Bulletin are all concerned with HP 3000 software, it's not surprising that any Software Maintenance Request may appear in all three. Open Software Maintenance Requests, which are documented in the Software Status Bulletin, may also be referenced in the "Known Problems" section of the Installation Tape Note files and, consequently, in the printing of these Note files in the COMMUNICATOR. Software Maintenance Requests noted as closed in the Software Status Bulletin should also appear in the Note files on the Installation Tape and in the COMMUNICATOR. In these cases, the software fixes will be described briefly under the "Corrective Software Changes" section of the Note files.

```
                                  . o  Installation Tape Note files
                                 .
                                .        - under "Corrective Software
                               .           Changes"
                              .
                             .
                            .
Verified Software -->. . . . o  COMMUNICATOR
Maintenance Requests .
                      .          - via advance printing
                       .           of Note files
                        .
                         .
                          .
                           . o  Software Status Bulletin

                                 - description of the fix
                                   (closed) or of the problem
                                   (open) and, possibly, of a
                                   work-around
```

Which One .... And When?

In case all of this has only confused you more, try this explanation. The Note files are written by lab personnel and are designed to tell you what's new with software on the HP 3000. You will receive these files on the Installation Tape (which also contains the encoded changes) and, shortly before, in the COMMUNICATOR. The COMMUNICATOR will also give you additional information concerning changes to HP 3000 software and, generally to the HP 3000 systems. Lastly, the Software Status Bulletin, prepared by lab and support personnel, provides the most current information available about software problems, new and old, and about fixes for those problems.

How To Get Them

All three of these publications are supplied to users with Hewlett-Packard's Comprehensive Software Support Service (CSS) and Software Subscription Service contracts. The COMMUNICATOR may be ordered separately, on an individual issue basis (subject to availability) and through subscription. Order forms are available in the COMMUNICATOR itself and through the Software Subscription Center of the Computer Systems Group (**).

```
+-------------------------------------------------------------+
|    * Note files are designated on the Installation Tape     |
|      by the format N00Nyyz, where:                           |
|                                                             |
|      yy = last three digits of the product number;          |
|       z = current version of product.                       |
|                                                             |
|           (e.g., COBOL is HP32213, version B;               |
|            its note file is N00N213B)                       |
|                                                             |
|      With little effort, "n0On" translates "noon".          |
|                                                             |
+-------------------------------------------------------------+
|                                                             |
|   ** Subscription Supervisor - COMMUNICATOR 3000            |
|      Software Subscription Center                           |
|      P.O. Box 61809                                         |
|      Sunnyvale, CA  94088                                   |
|                                                             |
+-------------------------------------------------------------+
```

• • •

Article Reprints

By Editor

"Minicomputer System Power Requirements," reprinted from *Mini-Micro Systems*/July 1976;

What's Happening to UPS (Uninterrupted Power Supply), reprinted from *Modern Data*/July 1975;

Protecting Minicomputers From Power Line Perturbations, reprinted from *Computer Design*/June 1976,

are the titles appearing in a DELTEC Corporation publication, Bulletin 124, 8/76.

If interested, Deltec Corporation is located at 980 Buenos Ave., San Diego, Calif. 92110. Telephone (714) 275-1331.

• • •

Still More Literature on Computer Security

By Editor

The development, implementation, and self-audit of a computer security plan requires time and effort. Since every facility has unique conditions and requirements, no single plan or set of specific steps can be developed to apply universally. However, there are a number of reference documents which may prove of use in the development or self-checking of the plan. The following publications are particularly recommended:

- National Bureau of Standards, Guidelines for Automatic Data Processing – *Physical Security and Risk Management,* NBS, Washington, D.C., 1974, 92 pages.
- American Federation of Information Processing Societies, *Security System Review Manual,* AFIPS Press, Montvale, N.J., 1974, 109 pages.
- Canadian Institute of Chartered Accountants, *Computer Control Guidelines,* CICA, Toronto, 1970, 135 pages.

- Factory Mutual System, *Loss Prevention Data - Electronic Computer Systems,* 1971, 7 pages.
- James Martin, *Security, Accuracy, and Privacy in Computer Systems,* Prentice-Hall, Englewood Cliffs, N.J. 1973, 626 pages.

Other steps which may prove helpful in self-auditing include the check of the entire security plan against those of other corporate facilities and the use of simulated conditions to test security and recovery plans.

Internal Audit

In an audit of the computer center by members of an audit staff, the manager of the computer center may be expected to provide a written computer security plan. Typically, the auditors will examine the plan to assess its consistency with these guidelines. In addition they will use procedures and techniques to test the elements of the plan to be sure that they adequately deal with security aspects and are being carried out in an appropriate manner. While these guidelines form a basis for the audit of the center, auditors exercise judgment in reviewing any areas not fully covered in the document.

● ● ●

Computer Security Conference

By Editor

Computer Security Institute will hold its annual conference and exhibition in New York City at The Statler Hilton Hotel, November 6th through 8th. For additional information, contact Mr. John C. O'Mara, at Computer Security Institute, Five Kane Industrial Drive, Hudson, MA 01749, Telephone (617) 562-7311.

● ● ●

Contributed Library Software Spotlight

DBREBILD: A Data Base Restructuring Package

A Report
by Jason M. Goertz
Computer Systems Programmer
Whitman College Computer Center
Walla Walla, WA 99362

This issue's software spotlight deals with one of several Image Data Base Utilities that are available on the HP 3000 User's Group Library. Submitted by Linford Hackman of Vydec, Inc., DBREBILD is a data base restructuring tool that is intended to supplement the HP utilities DBLOAD and DBUNLOAD, in addition to providing several functions which allow certain changes in data base security without the necessity of complete purging and rebuilding of the base.

DBREBILD has 5 entry points, each providing a different function:

UNLOAD - This function reads data from the data base and writes it to a magnetic tape, with no linkage or accounting information. This is similar to DBUNLOAD, except that the data set and item names, which are used by the LOAD entry point, are also written to the tape.

LOAD - This function reads a tape created by the UNLOAD entry point, and writes it to the data base. No account checking is done, so the destination base need not have the same name as the source base. (Copying of data bases, however, will be described later). The function of LOAD is similar to DBLOAD, except that all items are referenced *by name.* In this point lies the versatility of DBREBILD. New data items and data sets may be inserted anywhere in the schema, not necessarily at the end of the data set or data base description, as required by DBLOAD. (The rules of IMAGE, however, still cannot be violated. For example, masters must still appear before details, etc.) If a data item length is changed, the necessary truncation or justification is performed. New data items are initialized, and new data sets are ignored, allowing new manual masters to be loaded before the existing details they are linked to. Also, the order of existing data sets may be changed.

COPY - This allows copying of one data base to another, i.e., across group or account boundaries. The bases must be identical, however, and the user running the program must be the creator of the destination base, as well as have Privileged Mode and System Manager capabilities.

ALTSEC - Allows the user to alter the security (read/write levels) of any data item or set, by directly accessing the root file. The user must be data base creator and have SM/PM capabilities.

ALTPASS - Allows the creator of the data base to change passwords of the user levels, also by directly accessing the root file. Again, SM/PM capabilities are required for the user running the program.

Using DBREBILD, in conjunction with DBLOAD and DBUNLOAD, a great amount of versatility in changing data base structure can be realized. Certain operations, such as changing security levels and passwords, may now be done interactively, rather than by a complete unload, purge, rebuild, and load. The most important thing that can be said for DBREBILD, however, is that it works. Our site uses IMAGE extensively, and we have used DBREBILD numerous times to change our data base in ways that would be impossible by any means other than writing programs to pull the data out, rebuild, then write more programs to load the data back in — a time-consuming and laborious chore, at best. Any questions on this utility, or others in the library, may be addressed to:

Jason Goertz
Whitman College
Walla Walla, Washington 99362
(Telelphone) (509) 527-5417

The Clearing House

Controller Permits HP 3000 Computer Link to IBM 1403 Printer

From:
Roger E. Holmes & Associates
7900 Cowan Avenue
Los Angeles, CA 90045
(213) 670-5450

A printer controller developed by Spur Products Corp., Los Angeles, has made possible the mating of a Hewlett-Packard HP 3000 minicomputer to an IBM 1403 printer.

The data processing system was installed at executive offices of the Aerospace Group of VSI Corp., Culver City, Calif., when the company decided to replace its central IBM 360 computer with an HP 3000 minicomputer complemented by satellite computers at remote locations.

The Spur controller is plug compatible with the HP 3000, as well as the 1403 printer it was designed to operate.

● ● ●

$975 00

DBREFORM: A Database Utility for IMAGE/3000 Users

DBREFORM is designed to work with Hewlett-Packard IMAGE/3000 software to allow data base changes not provided with existing IMAGE/3000 utilities.

Changing a data base schema is very straightforward; however, except for relatively minor structural changes, moving data from the old design to a new design has required complex programs which are usually discarded after one use. Also, there are cases where a data base is loaded from a file rather than from the keyboard and the same conversion problems arise.

Another consideration, especially for the remote terminal user, is the requirement for the use of magnetic tape and operator intervention. Given sufficient disc space, the terminal user can control all DBREFORM functions from the remote terminal. However, in the case of very large data sets, or when loading data from another computer, use of magnetic tape is a DBREFORM option.

When loading data sets from files, there has been the necessity of editing and/or translating input data. DBREFORM has nineteen editing/translation subroutines which operate on specified fields of the input record, allowing a data set load in a single pass.

Use DBREFORM in place of complex data base redesign and data transfer programs which are used once and then discarded.

DBREFORM is specifically designed to provide at least the following functions:

1. Load a data set from an MPE file
   (With input editing, e.g., EBCDIC and packed decimal data to ASCII or binary)
2. Offload a data set to an MPE file
   (Chained or Serial mode)
3. Delete a data set from a data base
4. Add a data set to a data base
5. Re-name data sets, change master to detail or vice versa
6. Re-name data items within a data set
7. Re-order data items within a data set
8. Merge data items within a data set
9. Merge several data sets into one data set
   (can overcome QUERY limitation re: retrieval from multiple data sets)
10. Re-name a data base and/or move from one account to another

(Some applications require multiple passes)

One-time license fee is $975 per CPU including documentation. Return ORIGINAL and one copy of completed license agreement to:

ERIN ENTERPRISES
P. O. Box 201
Chantilly, Virginia 22021

DBREFORM package will be sent postpaid on a 1600 bpi tape.

90 DAY LIMITED WARRANTY

● ● ●

UNIVAC-1004 Emulator for Remote Job Entry

"CSM-1004-E": A UNIVAC-1004 Emulator for Remote Job Entry from an HP 3000 II to a UNIVAC-1100 Computer System. (CENTRO SPERIMENTALE METALLURGICO – CSM ROMA)

1. The "CSM1004E" lets you transfer data between HP 3000 II computer system and a remote UNIVAC-1100 computer system in a full multiprogramming environment over the public telephone (switched) network or a private leased line.

2. You can transfer data at rates of up to 4800 bits per second, depending upon your choice of modem. Higher transmission rates can be supported for certain applications. However, such determination must be made on particular specifications.

3. The emulator, which runs under the control of the HP 3000 multiprogramming executive operating system (MPE/3000), makes your HP 3000 computer system appear to the remote processor either as a UNIVAC-1004 or UNIVAC-9200.

4. The emulator is more flexible than the UNIVAC-1004 and the UNIVAC-9200 in that it allows you to use greater variety of input/output devices, including disc and magnetic tape.

5. You invoke the emulator with the ":RUN CSM1004E;PARM=LDN-CHAN" MPE command as described in "UNIVAC-1004 Emulator Reference Manual."

6. You can operate the emulator in either the job (batch) or session (interactive) mode. The session mode allows you to perform interactive commands such as: HALT, HALT-GO-VOICE, START, PUNCH-ABORT, PRINT-ABORT, etc. (See UNIVAC-1004 PROTOCOLL).

7. Provided that your HP 3000 computer system has more than one synchronous single-line controller, several people may use the emulator concurrently. The number of concurrent users is limited by the number of synchronous single-line controller which are available. Prior to invoking the emulator, you can specify which synchronous single-line controller you wish to use through the "PARM" parameter in the ":RUN" command.

8. The emulator transmits in XS3-UNIVAC code. It performs automatic conversion from UNIVAC-CARD code or ASCII-CARD code (only a 64 character subset available in XS3-UNIVAC code).

9. The emulator receives XS3-UNIVAC and converts to ASCII-SUBSET code.

10. The emulator transmits performing short-record truncation, and compacts blank elements, that are obviously restored in output.

11. The emulator works in half-duplex; however, as far as files are concerned, it can alternate input and output blocks, thus increasing transmission efficiency.

12. Comparisons have been made between the performances of the "CSM1004E" and of the previous facility we used in connecting HP 3000 with UNIVAC-1100 computers. This facility was the "IBM 2780/3780 EMULATOR" of the HP 3000, coupled with a HANDLER-2780 (property of INTEMA S.P.A.) operating on the UNIVAC-1100 computer system. In these comparison tests, the "CSM1004E" halved both CPU and elapsed times. However, these tests were made without taking advantage of the possibilities described in point 11). In normal routine work, when these possibilities are used, HP 3000 teleprocessing elapsed time appeared to be reduced to a much greater extent.

13. Among other things, the 2780-HANDLER previously used required about 10 K-words in the UNIVAC-1100 CPU: the "CSM1004E" is controlled by EXEC8, needing no UNIVAC extra-memory. Also, HP 3000 memory space requirements are strongly reduced.

The "CSM1004E", also provides automatic accounting of UNIVAC time and resources used.

For more details, please contact:

Mr. Vanni Giovanni
Centro Sperimentale Metallurgico SPA
Via di Castel Romano
10747 Roma (Italy)
Telephone: 0039 6 6495340
TELEX 62173

• • •

Financial Planning and Accounting Packages

Foresight Systems, Inc. offers five systems for use with the HP 3000 Systems:

Foresight, a financial planning and Management Reporting System;

Infonational's General Ledger with automatic budgeting and responsibility accounting system;

Infonational's Accounts Payable System with vendor analysis, check reconciliation and purchase order;

Infonational's Accounts Receivable/Sales Analysis System;

Infonational's Fixed Asset Accounting and Control System.

Each package includes installation by technical staff, user training by a consultant, complete user documentation, and upgrades.

Also available are maintenance agreements, plus technical, training and consulting resources, and a user group applications library.

For additional information, contact:

John Gewecke
Foresight Systems, Inc.
1901 Avenue Of The Stars
Suite 585, Century City
Los Angeles, CA 90067
Tel.: (213) 277-2722

• • •

TSPOOL – Remote Printer Subsystem

TSPOOL is a subsystem which enables a remote keyboard terminal to be utilized in much the same way as a line printer. Output files may be queued up by output priority and printed in turn with any number of copies specified. Additionally, TSPOOL is able to perform most of the carriage control functions reçognized by the line printer. This enables the user to print compiler listings, special forms, etc., on a hard-copy keyboard terminal and have the output appear just as it would on the line printer. Most existing programs can switch output from the line printer to a terminal via TSPOOL without modification. Any number of terminals can use TSPOOL simultaneously and each may have its own characteristics such as page size, etc.

TSPOOL is available by mail for the mere cost of getting it to you (i.e., mag tape cost, mailer, postage).

You will receive a magnetic tape containing the source code, object code, IMAGE data-base schema and an EDITOR file containing complete user documentation. The subsystem is written entirely in SPL and is for use on HP 3000 Series II and subsequent HP 3000 models. It most likely will run on cx and pre-cx models but has never been tried. The magnetic tape containing all files will be recorded at 1600 bpi.

TSPOOL is being made available to all interested parties on a non-profit basis and is neither supported nor guaranteed in any way. Questions, however, may be directed to the author:

Terry Branthwaite
12413 S.E. 173rd Place
Renton, WA 98055
Tel.: (206) 271-3495 (Evenings)

To receive a copy of TSPOOL, please send $15.00 to the above address stating clearly your name and address. Allow at least two (2) weeks for delivery.

● ● ●

Computerized Hiring and Rental System

During the latter part of 1977 and the beginning of 1978, Freightways Data Centre Ltd., designed, wrote and implemented an on-line computer system for Transport Container Pool Ltd. Although the system has been initially used for the business of container hirage it can be modified to suit general hirage. The following gives some indication of the system capabilities.

SYSTEM OBJECTIVES:

1. *UP-TO-DATE INFORMATION*
   Up-to-date knowledge of all items. Eesentially the whereabouts and status are the key attributes to be recorded. The information must be accurate and available at short call.
2. *CONTROL*
   Control of all item movements. (For containers this helps ensure a logical succession of trips.) Illogical or unconfirmed movements are held by the system and upgraded to confirmed when the 'missing' information is entered.
3. *PERMANENT/CASUAL HIRE*
   Facilities for permanent/casual hire are included in the system. Allowance is made for permanent hire items to be used on casual hire. (Allowance is made for documentation of movement of permanent hire items.)
4. *DEBTORS*
   The system caters for standard debtors requirements such as invoice/statements, credit control, trial balance, hirer information, debtors control.
5. *ITEM ANALYSIS*
   Analysis of item movements providing utilization, status and hirer usage.
6. *REVENUE REPORTS*
   Revenue reports by depot, item type, representative, customer.
7. *ENQUIRY*
   The facility to 'interrogate' the system asking for details of the information. (e.g., item information, hirer information.)

The system is written in COBOL using structured programming techniques as a multi-user package. IMAGE and QUERY are used extensively. The system is currently operating on a HP 3000 on a timesharing basis utilizing to leased telephone lines. The company using the system has two Hewlett-Packard display terminals and a DEC LA180 printer.

The system is available for purchase or lease.

For further information, please write to:

The Manager
Freightways Data Centre Ltd.
Private Bag
Penrose
New Zealand

● ● ●

FOR SALE:

Grumman Controller for IBM 1403-N1 Printer. Can be used with standard HP interface and cable.

Contact:

Ernie Broudy
VSI Corporation
Culver City, California
Tel.: (213) 838-2131

● ● ●

Journal of the HP General Systems Users Group
c/o Hewlett-Packard General Systems Division
5303 Stevens Creek Blvd.
Santa Clara, CA 95050
U.S.A.

ADDRESS CORRECTION REQUESTED

Linford Hackman
VYDEC, Inc.
9 Vreeland Road
Florham Park, NJ 07932