Modifications to the APL/1130 System Part 2

J. F. Clementi
I. B. M. (Australia)
95 North Terrace
Adelaide, 5000, South Australia

B.G. James
School of Mathematical Sciences,
The Flinders University of South Australia,
Bedford Park, 5042
South Australia

R. P. Fletcher
School of Mathematical Sciences,
The Flinders University of South Australia,
Bedford Park, 5042,
South Australia

In Part 1 of this series we indicated that two immediate projects to be pursued were:

(a) Introducing a 'Paging' routine for the console typewriter for operator convenience in separating output;

(b) The implementation of two new system commands to allow printing of alphanumeric plus several special characters on the 1132 printer or printing of the full character set on the console typewriter.

Both of these projects have been completed.

(a) The system counts lines as the console typewriter successively skips to a new line and after typing the sign-off message skips to the beginning of the next page. This is accomplished by passing control after the printing of the sign-off message to a 'Paging' overlay which is disk resident. The overlay then returns control to the system which in turn searches for the next sign-on card.

(b) The two new system commands to control printing are:

(1))PRINTER N

and (2))CONSOLE N where N is any non-negative integer less than or equal to 32,767.

The)PRINTER N system command causes everything which would normally be printed on the console typewriter to be printed on the 1132 line printer thus speeding up output. The characters available on the line printer are:

A to Z, 0 to 9, = , . - * ( ) ' and + .

An attempt to print any other character will result in a $ (Dollar sign) being printed in its place, with the following exceptions:

(1) [ ] are converted to ( ) respectively to allow tracing of function execution on the line printer.

(2) The negative sign ¯ is converted to a minus sign - .

Execution of each printer command causes a skip to a new page with the following message printed out at the top of the page:

PRINTOUT NUMBER N OF USERNAME

Where N is as before and USERNAME is the name by which you are known on the system. This heading is followed by a blank line after which your printing begins.

The Command:

)CONSOLE N

Restores the system to normal console mode of input and output. At present

it is not possible to use the)PRINTER N system command from the console typewriter.

A couple of facts:

(1) If an error occurs while in printer mode the information regarding the error is printed on the printer.

(2) A final)CONSOLE N command is unnecessary as the sign–off routine assures that the system is in console mode. If the final)CONSOLE N command is omitted the following occurs at sign-off time:

)OFF (on the Printer)
SIGNED OFF (on the Console)

In addition to the above extensions, two more extensions have been made. They are:

(a) A patch suggested by Steve Raucher has been implemented,
and (b) the system has been modified to accept optical mark read (OMR) cards from a 2501 card reader with the OMR feature.

(a) Normally when the system detects an error in card mode on our system, it would flush cards until a)OFF system command was read. The system would then sign–off, page, and search for the next sign-on card. With the suggested patch, the flushing of cards is suppressed when an error is detected. This means of course that some meaningless printout will occur, but the advantage of having a complete reading of the cards to check for mis-punchings or mis-markings (depending on whether or not you are using the OMR system) has more than outweighed any disadvantage arising from meaningless printout. In addition by having a complete listing of a program, logic errors can often be discovered before the program runs for the first time, thus saving machine time.

(b) The optical mark read (OMR) APL System is being used by the South Australian Education Department on there 1130 system which is equipped with a 2501 card reader with the OMR feature. A photostat of the card design being used is included in this article. A study of the card will show that some rather useful operators are not on the card. This situation arose because of a space problem on the card and the reluctance to decrease the number of characters per card to less than 19 or 20. Marking a CT in column 20 allows for continuation to the next card. Each student is issued with a chart indicating for each missing operator the marks that he must make to get the desired operator.

No difficulties have so far been experienced in the use of the card. The cards. are printed in two colors with two different color distributions being used to test student preference. Marking instructions are printed on the back of each card.

Except for the mode of card input, the two APL systems are identical. A simple patch deck converts from one system to the other to allow for interchange of programs and workspaces.

Currently under way are two projects:

(a) The implementation of a)NEWPAGE system command. When this command is implemented it will cause skipping to a new page on the line printer. After the first)PRINTER command each successive)PRINTER

system command will cause a skip of 1/3 of a page.

(b) An attempt is being made to restore the) PCH and)PCHS system commands which were removed to allow for the punching of ⍴, ⍳ and ∇ on an 029 Key Punch. When implemented these commands will be accepted in card mode. Before a card is punched, it will be checked to see that it is blank. An attempt to punch a non-blank card will terminate punching.

Three projects which are planned but not underway at present are:

(i) Printing of message when a program is terminated by the operator;

(ii) Printing of the current date for each sign-on;

(iii) Extension of the 1850 word (i.e., 925 characters, numbers, etc.) core limit, perhaps by an automatic overlay procedure: exceeding this 925 character limit gives the WS FULL message.

correspondence

To The Editors, Quote-Quad

The following functions are being experimented with on the UMASS–APL system at the University of Massachusetts. We would appreciate comments from the reader about the usefullness of these functions and any changes that should be made. Note that we use I-beam funcitons in place of the function names specified for test purposes only. We feel that all I-beam functions should be masked from the user in user-defined locked functions. These functions are available on the experimental APL system only.

The function

TRAP arg

returns the right arguement as its value. The right arguement must be a one element integer object or a null. After the TRAP function is executed any errors (i.e. DOMAIN) will cause execution to resume at the line specified by the right arguement of the TRAP function. A null removes any trap set previously by the TRAP function. No other levels of functions are affected. If an error occurs, and the right arguement specified a non-existent function line, a return will be made for the function. If an error occurs on the line specified by the right arguement, the normal diagnostic is provided (infinite loops can still occur, but they can be traced as with other infinite loops). The diagnostic 'SYSTEM ERROR' can not be trapped. If an error occurs in a locked function with no trap specified, the SI vector will be un-raveled to the first un-locked function (as is the case now) or to the first function with a trap specified (at which point the trap will occur).

The function

ERROR arg

requires a one element positive integer object as its arguement. Execution of the function causes execution to be terminated as though an error had occurred on that line. The arguement specifies the diagnostic to be printed.

Argument Value	Diagnostic
0	nil (manuel interrupt
1	SYSTEM
2	SYNTAX
3	DOMAIN
4	LENGTH
5	VALUE

6	RANK
7	INDEX
8	DEPTH
9	LABEL
10	SYMBOL TABLE FULL
11	WSFULL

The function

WHY

is a niladic function whose value is a two element vector. The first element of the result specifies which error occurred last (for values see the ERROR function) and the second specifies the line on which the last error occurred.

James H. Burrill, Jr.
APL Group
University Computing Center
University of Massachusetts
Amherst, Mass. 01002

```
      ⍝EXAMPLES

      )SIV
      ∇F[⎕]∇
    ∇ Z←A F B
[1]   Z←A÷B
    ∇
      ∇TEST[⎕]∇
    ∇ Z←R TEST S;A
[1]   ⍝THE NEXT LINE SETS THE TRAP AT LINE L1
[2]   A←8⌶L1
[3]   ⍝THIS LINE MAY CAUSE A DOMIAN ERROR
[4]   Z←R F S
[5]   →0
[6]   ⍝GET THE INFO ON THE ERROR
[7]   L1:A←⌶0
[8]   'A TYPE';A[1];'  ERROR OCCURRED ON LINE';A[2]
[9]   ⍝THE NEXT LINE TURNS OFF THE TRAP
[10]  A←8⌶⍳0
[11]  ⍝THE NEXT LINE INFORMS THE USER OF THE ERROR.
[12]  A←9⌶0
    ∇
      1 TEST 1
  1
      1 TEST 0
DOMAIN ERROR
F[1] Z←A÷B
       ^
      ∇F
[2]   ⍫
      1 TEST 0
A TYPE  3  ERROR OCCURRED ON LINE  4
TEST[12] A←9⌶0
           ^
      ∇TEST
```

```
[13]  ∇
      1 TEST 0
A TYPE  3  ERROR OCCURRED ON LINE  4
1 TEST 0
^
      1 2 TEST 1 2 3
A TYPE  4  ERROR OCCURRED ON LINE  4
1 2 TEST 1 2 3
^
      )SI
TEST[12] *
F[1]  *
TEST[4]
      )PACK
      )SI
      1 TEST 0
A TYPE  3  ERROR OCCURRED ON LINE  4
1 TEST 0
^
      )SI
      ∇END OF EXAMPLE
```

An Implementation Note Concerning Literal Constants

It is necessary to warn a beginner using APL/360 that if he neglects to complete a pair of quotes around a literal constant, he may push RETURN and be unable to elicit any response from the system. He will be unable to escape from this situation until he provides the missing quote. Special treatment of quote marks in the input stream thus adds a complexity to the system that even the rank beginner is forced to acknowledge. An implementation alternative is presented below which should circumvent this problem. In offering these ideas to others it is hoped that airing reasons for both approaches will bring implementers to an agreement and avoid inconsistencies between implementations.

It appears that the APL/360 treatment of unmatched quote marks offers a way of entering a carrier return character and also allows the literal data to be entered on separate lines on the paper (although the characters entered are presumably treated otherwise by the system as a single line). A better way of entering a carrier return in a literal constant might be to represent it as C overstrike R. This does not allow a literal vector to be typed on several consecutive lines, but it is not clear that it is desirable to do so. If the literal data is being entered on several lines, how far back can the user backspace to make corrections? Allowing him to revise the entire input would conflict with the principle of visual fidelity, but being unable to do so would be restrictive, especially when modifying lines of a function.

Another carrier control character is the backspace. On the UMASS–APL implementation a backspace is represented by B overstrike S. The treatment of carrier return and backspace characters is thus consistent. The alternative of having the backspace character available only as the value of a variable in a public library workspace seems comparatively undesirable.

This treatment of the carrier control characters has been implemented on our system by Jim Burrill and costs very little to the system. It should be remembered that these carrier control characters should be printed as overstrikes when they are displayed as part of a function or in an error report under immediate execution mode. The special handling to do this is negligible

with our system. This allows us to easily edit a line containing either of these characters, and circumvents a treatment that would obtrude on users who do not wish to be concerned with carrier control characters in their literal constants.

Clark Wiedmann
University Computing Center
University of Massachusetts
Amherst, Mass. 01002

Editors, Quote–Quad,

1. In the promotion of APL, one key feature has probably not been emphasized enough and adequately publicized. It is the APL tracing, a very flexible and powerful feature, unique for APL, and not available for CALL 360 BASIC, FORTRAN, and PL/I.

In using APL, I usually have a number of trace functions in each workspace. For example:

```
      ∇TRACE X

[1] T∆ALPHA←⍳X
[2] T∆BETA←⍳X

      .....
      .....
      .....

[ ] T∆OMEGA←⍳X

      ∇
```

ALPHA, BETA, and OMEGA are function names. After entering TRACE 100, all functions will be in the tracing mode. By entering TRACE 0, all tracing will be halted.

The APL tracing feature is especially effective in debugging or understanding large programs coded by other programmers in other computing centers. With tracing, one can grasp immediately the logic and flow of an unfamiliar program and narrow down the range and area of possible errors and bugs. Recently I have looked into a 4500-card FORTRAN program and I badly missed the efficient tool of APL tracing.

Several years ago, the Bell Laboratories carried out an experiment, comparing the productivities of two similiar programming groups, one using FORTRAN and one using ASSEMBLER. The result was that the FORTRAN group was much more productive than the ASSEMBLER group. Many of your readers would probably agree that APL programming requires much less time than other terminal languages. For many, the APL tracing feature saves a lot of time in debugging.

2. In your January 15, 1971 issue of QUOTE–QUAD, you published a determinant function submitted by Mr. Wesley C. Kranitz. The following is a determinant function developed and popularly used at the IBM Scientific Center at Philadelphia which might be of interest to your readers.

```
      ∇DETRM[□]∇
    ∇ R←DETRM M;I;J;O
[1]   J←(⍴M)[O←(⍳0)⍴⍳1]
[2]   →4×(=/¯2↑⍴M),R←⍳0
[3]   J←1⍴⍴M←(1 1 ↓M)-(1↓M[;O])∘.×(1↓M[O;])↑1⍴M
[4]   I←J⌊(M[O;]≠0)⍳1
[5]   →3×(0≠R←×/R,(¯1*1≠I)×1⍴M[;O,I]←M[;I,O])∧1≠J
    ∇
```

Sincerely,
C. C. Tsao
IBM, Armonk
New York, 10504

To the Editors, Quote–Quad

We recently purchased a Hewlett-Packard 7201–A graphic plotter. This plotter, unlike most others, is language and software independent, and works extremely well with our APL/360 system. Commands for the plotter are as follows:

"PLTL"	–	plot the following points in line mode
"PLTP"	–	plot the following points in point mode
"PLTT"	–	terminate plotting

The only formatting needed is to scale the X and Y values between 0 and 9999, restructure the two vectors into an Nx2 matrix (using lamination), and drop off the decimal part (using ceiling or floor). Plotting is accomplished by simply typing "PLTL" or "PLTP", and then printing out the Nx2 matrix of scaled points. To terminate the plot, the user types "PLTT". Terminal printing can also be suppressed during plotting, if desired. The example program that follows will plot B=sin A:

```
B←1+10A←.1×⍳100
'PLTL'
⌊(800×A),[1.5] 4000×B
```

(the curve is now plotted)

```
'PLTT'
```

I have also written, with the help of Warren Juran at Proprietary Computer Systems, a complete conversational package that plots vector data (similar to 1 PLOTFORMAT PLOT), evaluates, scales, and plots user-defined functions (in rectangular or polar coordinates), and solves and plots curve-fitting problems using quad-divide. We find that these routines make the plotter and APL system attractive and useful to even the most elementary computer user.

We compared this plotter with the TSP-system, and found that the HP takes less CPU time, requires absolutely no software (unlike the TSP), and has about forty times the plotting resolution. There are, however, two disadvantages to the HP compared with the TSP. The first is plotting speed–the TSP has sacrificed resolution for speed and can plot 3.2 times faster (about 215 points per minute). The other disadvantage is analog output–the TSP delivers analog output capable of running any analog X–Y recorder or storage oscilloscope. We are in the process of developing, however, a modification to the HP that would allow this flexibility.