# PROPOSAL FOR FINAL CRMS APL INTERFACE

APL Subcommittee

David Redell

August 6, 1974

## 1. Function Definition Mode

As in APL\360, function definition mode allows the editing of one function at a time, and is entered by typing:

a) The header of a new function

b) A '∇' followed by the name of an existing function

In case (a) a new function is being defined (i.e., appended to the end of the program). If a function by that name already exists, it is an error; but if a global variable already exists by that name, it will be automatically erased and replaced by the new function. The syntax of the function header is checked immediately. In case (b) an old function is being modified, hence the function must exist or an error is signaled.

Once function definition mode is entered, system commands are not recognized until it is left by typing a closing '∇'.

In function definition mode, the most frequent form of command is the line-entry command which consists of one or more line numbers, an APL statement, and a carriage return. The first line number is one suggested by the system, but the user may type in one or more additional line numbers. In all cases, the last of these line numbers is the one which determines which line is being entered. If the APL statement in the command is not empty, it becomes the contents of the indicated line (i.e., replaces the line if it existed, or is added to the function otherwise) and the system suggests that the next line be entered. If the APL statement is missing from the command, however, rather than entering nothing as the contents of the line, the system suggests the indicated line number again.

Examples:

[5] (CR)

[5]

[5] $A \leftarrow \rho B$ (CR)

[6]

[5] [8] (CR)

[8]

[5] [8] $A \leftarrow \rho B$ (CR)

[9]

A line-entry command can also be used to modify a line, because the system always provides the original contents of the suggested line as the line-collector's "template."

Examples:

suppose that line 7 is

$$Z \leftarrow (+/N) \div \rho N$$

then

[7] $X$ (P^C) $\leftarrow (+/N) \div \rho N$ (CR)

changes it to

$$X \leftarrow (+/N) \div \rho N$$

and

[7] (◊^C) < [20] (P^C) $Z \leftarrow (+/N) \div \rho N$ (CR)

defines line 20 to be a copy of line 7

In addition, a line may be deleted by entering as its contents the single character "line-feed" (which is an illegal character in any other context).

Example:

[5] (LF) (CR)

[6]

deletes line 5

Finally, a line may be inserted between two existing lines by giving it an appropriate non-integral line number.

Example:

[5] [5.1] $I \leftarrow I + 1$ (CR)

[5.2]

inserts the new line between lines 5 and 6

The fractional part of a line number may only be 2 digits long. After a non-integral line number is entered, the next suggested line number will be one greater in its last non-zero digit (e.g., after 5.87 would come 5.88, 5.89, 5.9, 6). Line numbers are converted into consecutive integers when function definition mode is left.

The size of the mapping table matching logical line numbers to physical line numbers during function definition mode limits the fraction to two digits and the integral part to 600. (The table is a vector of 16-bit half-words.)

## 2. Workspaces

As in APL\360, the user is allowed several workspaces holding programs and data. The workspace "*CONTINUE*" is always loaded when starting up and always saved when leaving.† The commands )*LOAD CONTINUE* and )*SAVE CONTINUE* do nothing, unlike in APL\360. There may be )*PLOAD* and )*PSAVE* commands which load and save only the functions but not the data in a workspace.

To provide a repeatable initial state for experiments and other saved programs, the command )*START* is provided, which acts like )*CLEAR* on the data in the workspace, but leaves the stored functions untouched. In particular

)*START* <immediate statement>

will initialize the workspace and execute the immediate statement in a fresh new process.

A workspace consists of a program (text, code, and debugger segments), one data segment per process, and a "state" segment for reconstructing internal tables in ARS. Thus, in CRMS APL, a workspace can hold an entire experiment consisting of several processes, each with its own stack and global variables. It also holds any mailboxes created by the processes. Saving the workspace saves the entire statem which is helpful for later debugging, although in general, resuming execution after reloading the workspace will not be possible since the connections to terminals and files will have been saved. (In one sense, this is an advantage, since it allows leisurely perusal of crashed experiments without tying up terminals.)

---

†In fact, *CONTINUE* is itself the working copy in memory.

There are certain (infrequent) times when, for implementation reasons (cluttered symbol table or global variable are overflow) a )*START* must be done which would otherwise be unnecessary. This will be automatically followed by a global recompilation when execution is next attempted (to reclaim symbol table and/or global area space).

## 3. Execution

In general, starting and resuming execution (i.e., immediate statement containing a branch, a return, a call on a user-defined function, or *)GO* ) cause recompilation of all functions which need it. If any errors are detected during compilation, execution is not allowed. A function needs recompilation:

a) if it has been modified

b) if it had any errors when last recompiled

c) if any global name has been erased (including a global variable overriden by a newly defined function)

d) if any function has had its header modified

e) a *)START* has been done and recovery of global variable and/or symbol table space is needed

Note that in cases (c) and (d) only the body of each function need be recompiled. Note also that the global symbol remains valid in all cases except (e), which is the only time a total recompilation is needed.

If the stack is empty (following *)START*, *)CLEAR*, or '→'), an immediate statement executes in the context of the global variables (if any) which exist at that time. If an error breakpoint or etc. occurs, a message is printed and the state may be examined. *)GO* resumes execution exactly where it left off.

If the stack is not empty, an immediate statement executes in the context of the appropriate function in the stack. Normally, this is the top function but the *)IN* command may be used to select a different one.† (This is most useful for examining local variables of functions

---

†This can be implemented by saving higher frames in an external segment and putting them back before resuming execution.

in the stack.) If an error occurs during the immediate statement, it is absorbed and the stack is left in the state it was in before the immediate statement started (i.e., multi-level debugging is not provided).

Whenever an error occurs, all process are stopped until execution is resumed. This allows debugging and experimenter-process I/O to use the same terminal. It also allows the APL process data segments to be swapped out during debugging, thus saving memory space.

While the program is stopped at an error (breakpoint, etc.), inactive functions and the top function on the stack may be edited. No function further down in the stack may be modified unless all higher functions are removed. Furthermore, after the top function is edited, the *)GO* command is no longer applicable; only a branch may be used to resume execution just as in APL\360.

## 4. Commands

Most APL\360 commands are retained in the same or slightly changed from:

| | |
|---|---|
| *)OFF* | (works like *)CONTINUE* in APL\360) |
| *)CLEAR* | (unchanged) |
| *)LOAD* | ( " ) |
| *)COPY* | (works only for functions) |
| *)PCOPY* | ( " " " " ) |
| *)SAVE* | (unchanged) |
| *)DROP* | ( " ) |
| *)WSID* | ( " ) |
| *)LIB* | ( " ) |
| *)FNS* | (not alphabetized) |
| *)VARS* | ( " " ) |
| *)ERASE* | ( " ) |
| *)DIGITS* | ( " ) |
| *)WIDTH* | ( " ) |
| *)SI* | ( " ) |