Working Document
David Redell
July 22, 1972

CRMS APL/SIMPLE

Integrated Language Processing System:

Design Considerations

I. Introduction.

An interactive language system consists of several components, which provide various facilities for program development (e.g. editing, debugging). Existing language systems generally take one of two approaches regarding the organization and interaction of these components:

A. Modular Approach:

Each component of the language system is a separate and independent program. This has the advantage of allowing the user to freely choose a language and compiler, and to use these in conjunction with the standard (or the user's own) editor and debugger. The problem with this generality is often a kind of "lowest-common-denominator" effect, due to a lack of uniform conventions regarding the format of the object program and its data, and the correspondence of these with their source-language counterparts.

B. Integrated Approach.

The various components of the language system are considered to be parts of an integrated package, which processes a single language. Detailed conventions are established for the format of the program, allowing the user to operate at a high level at all times, ignoring low level details (machine instructions, octal dumps, etc.) which obscure the user's view of his program. A major problem with the integrated language system is the specialized orientation of its components, which generally precludes their use individually, or in a system for another language.

The CRMS language system attempts to compromise between these two extremes. Two integrated language systems are provided, one for APL and one for SIMPLE. By careful parallel design, it has been possible to establish very similar conventions for program formats in the two languages, with the result that many components of the two systems can be shared.

The integrated-system approach is particularly crucial in the case of APL. The primary users of the META-APL language will be experimenters and their assistants, whose progress must not be impeded by irrelevent details of machine language debugging and so forth. The original APL/360 system provides a set of tools for source-language-level program development; META-APL, while differing in its command language interface, provides a similar (but rather more powerful) set of tools.

A further advantage of the compatibility of the APL and SIMPLE language systems is the relative ease with which an APL programmer can learn to use the SIMPLE system. The more sophisticated experimenters may eventually wish to program their own versions of various support routines (e.g. input/output managers for special devices) which run on the SIMPLE processor. They will be aided in this by the compatibility of the two language systems, and by the flexibility of the operating system, which allows graceful replacement of such routines on a per-user basis.

II. Facilities Provided.

An integrated language system smoothes over the traditional divisions between "editor", "compiler", and "debugger". Nevertheless, these divisions are still rather useful, both as guidelines for system organization, and as categories for discussion of the facilities offered.

A. Editing.

The editing facility allows the user to create and update source programs by locating, inserting, deleting, replacing, and modifying lines of text. In keeping with the function/global block structure of APL and SIMPLE programs, the editor considers the source text to consist of a sequence of blocks. Thus, line addresses are triples: (program,block,line#). At any given time, the language system considers some one line to be the "current" line, at which point insertions, replacements, and other changes can be made. Various commands shift attention to a new current line. Most important are the context-search features, which scan the text (either current block or entire program) for a line containing a desired substring. The current line and context search features of the editor are useful more generally in the language system (e.g. for locating the desired line for insertion of a breakpoint).

B. Compiling.

The compilers for APL and SIMPLE present a uniform internal interface to the other components of the integrated language system. The user of the system interacts with the compiler somewhat indirectly, via commands to the other components. There are essentially two user actions which invoke the compiler:

1. Program Modification.

   Editing of the source text causes automatic recompilation of the modified block (or, when necessary, the entire program) when execution is begun or resumed.

2. Immediate Statements.

   Statements typed in by the user for immediate execution are compiled in the context of the currently active function and

executed individually, whereupon control returns to the user at his console.

C. Debugging.

Probably the most complicated part of the language system, at least from the user's point of view, is the debugging facility. Some complexities here are intrinsic (e.g. multiple activations of recursive functions) but, as far as possible, purely implementation dependent complications are hidden. All addresses in the object-code are translated back into source line addresses, using a table constructed by the compiler. Addresses in data areas are translated back into symbolic names, using the compiler's symbol table. Breakpoints, statement tracing, and function tracing are all available, and are always referred to in terms of the source text, rather than object-code addresses.

D. Command Interface.

All commands to the system are interpreted by a single "front-end" module, which calls upon the editing, compiling, and debugging modules to perform the required actions. This localization of command interpretation is designed to unify the command language and ease modifications as they become necessary. Eventually, a macro-command facility will be provided to further ease the development of programs using the integrated language processing system.