select all of it, copy it, open the original document, and paste into it. In the same way, append makes transferring pieces of information from several sources easier because it allows one to add text either to the end of an existing file or to the end of the clipboard.

Being able to keep several windows (documents) open simultaneously in WordPerfect is convenient. In WordPerfect I divide my manuscript into separate documents while I edit and then retrieve them when I have finished. My reference list is opened as a separate document while I write my paper. That way I can add or delete a reference from my paper and my reference list without having to scroll back and forth through a single document. I keep better track of any changes this way.

The automatic backup feature on WordPerfect, which saves one's creative work at regular intervals, is a nice touch. MacWrite (version 4.6) does not have this insurance, and I have been caught out more than once with unexpected power outages.

WordPerfect is ideal for book writing. The spell, thesaurus, and mark text features of WordPerfect along with the two headers and footers that can be placed on every page or every odd or every even page show that WordPerfect was created with this purpose in mind. The mark text command allows one to mark words and then use them to generate a list, table of contents, or an index. The ability to format a document into several columns is another useful feature, especially for producing a newspaper or newsletter.

When files need to be reordered or cleaned up or a "lost" document must be found, the file management feature is there to help. With file management files can be retrieved, copied, or printed. One can delete, rename, look into, or display information about a file or folder or display all the files that contain a specified word.

The manual that comes with WordPerfect is well laid out and well written. It is difficult to describe everything a novice needs to know to use WordPerfect to its full potential, but the manual does a very good job.

I like having all the extra features in WordPerfect even if I rarely need them for my work. However, in my opinion, WordPerfect is not an "outwrite" winner over MacWrite. MacWrite is better than WordPerfect when it comes to pasting graphs into a document. With MacWrite, one can scroll through the graph, much as one scrolls through the text of a document. Using WordPerfect version 1.0, I had difficulty in seeing the graph on the screen if the graph occupied one whole page. With WordPerfect version 1.0.1, viewing the graph on the screen is much less of a problem than with version 1.0 but still not as good as with MacWrite.

The indent (wordwrap) feature on WordPerfect is not as versatile as wordwrap in MacWrite. In reference lists where the second and subsequent lines are often indented, I have never been able to do this as easily with WordPerfect as with MacWrite.

In WordPerfect, the ruler is more awkward to use than in MacWrite. The WordPerfect ruler always appears at the top of the page, but it is the ruler for the line the cursor is on. Each line can have a different ruler setting, but one can only observe the ruler setting for a particular line of text by placing the cursor on that line. In MacWrite the ruler always appears in the text above the section of text it controls, so it is easy to see where changes in format occur. The ruler in WordPerfect is unobtrusive and tabs are more easily set, but the ruler on MacWrite is easier to understand and it is much easier to determine where changes in the ruler setting have been made.

WordPerfect is available for a variety of computers and is compatible among them. I transferred a file created by WordPerfect for the Macintosh version 1.0.1 to an IBM-AT. The text transferred accurately, but the wordwrap for each line had to be corrected and changes in style in the original text were lost in the exchange.

Being able to transfer files (or documents) between computers is useful for a lab like ours with Macintosh, IBM, and VAX computers. One can start a document on one type of computer and finish it on another, and if a professional-looking printout is needed, WordPerfect files can always be transferred to the computer with the best printer.

I recommend WordPerfect version 1.0.1 as a powerful, user-friendly word-processing package. The number of features in MacWrite (version 4.6) is modest compared with WordPerfect, but then, so is the price.

ASYST 2.1

KENNETH L. RATZLAFF†

Instrumentation Design Laboratory, The University of Kansas, Lawrence, Kansas 66045-0046

Received December 12, 1988

ASYST, in a nutshell, is a *scientific software development environment* based on the FORTH programming language, but greatly extended to meet the needs of the laboratory scientist. This review will attempt to describe ASYST, compare it with other methods of software development, and discuss performance and applicability for scientific problems.

FORTH-type languages are not familiar to most scientists, and consequently a summary of this class of languages is in order. In FORTH, software commands (or *words*) are equivalent to the procedures and subroutines of other languages, but very important differences exist. First, when a single word is defined and entered into the computer, the text is not retained, but it is immediately compiled and becomes a resident part of the system. At any time thereafter when the ASYST prompt allows command entry, that word can be executed just as would be any interactive command that is part of the native language. Successive new words may be defined, using old words (defined by ASYST or by the programmer) as building blocks, until entire programs are written as a single word. This has the powerful advantage of keeping many compiled procedures available for immediate execution at any time.

To string together these words, FORTH-type languages use a Stack (similar to but not the same as the CPU stack) onto which parameters are placed and accessed via Reverse-Polish notation (RPN). Consequently, the style of programming is substantially different from those to which scientists are customarily exposed; because code does not read linearly,

† BITNET: RATZLAFF@UKANVM.

FORTH can be difficult to learn, even for experienced programmers. If comments are not included generously, the code can be difficult to read even for other FORTH programmers.

Whereas FORTH is typically a low-level control language with few data-processing tools, the strength of ASYST is the richness of its collection of words (the dictionary). Delivered as a set of one to four modules, the ASYST dictionary includes words for standard mathematical operations, matrix operations, FFT, statistical functions, 2-D and 3-D graphics capabilities, and data acquisition and control using RS-232, GPIB, digital, and analog I/O. The data acquisition modules include drivers for a very broad collection of GPIB and analog I/O boards. A debugger and a barely adequate editor are also part of the package.

In each case where the ASYST developers anticipated that a user might need a relatively high-level function, a word was developed, written in assembly language for high performance. Consequently, a procedure such as a matrix inversion is fast if it is performed by using a word from the ASYST dictionary. In opposition to tendency toward typed languages such as Pascal, ASYST words accept any numeric data type, analyzing the type "on the fly"; this is convenient, although it slows execution.

To understand ASYST's relevance, it may be useful to compare the ASYST environment with others used for developing laboratory applications.

One option might be a preprogrammed package that requires only that the user select parameters from menus. These packages are available for tasks such as data processing and statistics or data acquisition. The user is limited to performing the tasks in the way prescribed by the software author; for example, for data acquisition, one may be limited to oscilloscope emulation and simple spectrum analysis. Furthermore, these packages do not support a wide range of tasks.

A second option is the use of a conventional compiler such as FORTRAN. Many analog I/O boards come with drivers that can be linked into the program. Libraries of programs can be consulted for special functions. Program development is tedious because of the time-consuming edit/compile/link/test syndrome.

A third option is the use of integrated environments such as Turbo Pascal or Turbo C (Borland International). These differ from other compilers in that the edit/compile/link/debug is all accomplished automatically from within a single program. Substantial libraries of functions including scientific and graphics support are available, but support for analog I/O boards must come from the board vendor or be written by the user.

Compared with these alternatives, ASYST has the richest functionality, combining support for I/O, statistics, and graphics with computational capabilities comparable to other languages used by scientists. No other known environment covers this range.

When writing an application in ASYST for the first time, one cannot help being impressed with the ability that it also provides for a user to interactively manipulate data. Operations can be written, reedited, and used while keeping the data set in memory; consequently, one can try different data-processing or graphics schemes just as one would if operating manually. While not useful for routine computer work, this is a very powerful mode for working up unique sets of laboratory data or debugging a specialized hardware interface.

There are several areas that might be perceived to be shortcomings in performance and/or capability.

First, although the richness and performance of the words in the native dictionary have been underscored, performance when words must be strung together is seriously degraded. For example, the tutorial poins out that two 5000-element arrays may be added in a single operation rapidly; we measured 0.98 s. If the same operation is performed by using a loop and an index variable, the operation requires about 17 s, a procedure that would be necessary, for example, if a nonstandard test was required within the loop. Incidentally, the same loop programmed in Turbo Pascal requires 1.01 s, leading one to the conclusion that high speed is only achieved when the standard dictionary is used.

Second, operations that were not anticipated by the developers can be rather cumbersome. For example, the procedure that is required when one wishes to scale the labels on a contour plot is quite complex compared with the simple procedure to use the array indices as labels. In the same vein, it can be difficult to include procedures that are coded in assembly language. Vectoring hardware interrupts to an ASYST word is not possible, eliminating some very useful real-time control methods.

Third, the documentation, although apparently massive (four large binders), is frequently insufficient; besides adding detail, cross-referencing would have been extremely helpful.

Finally, several error-handling problems appeared, including the vexing problem of having the computer occasionally freeze when procedures were improperly coded; in several cases, the computer had to be powered down before it would properly reexecute ASYST. (We were informed that an incompatibility with DOS was at fault in some of those cases and could be fixed in DOS 3.3.)

In view of these problems, the "Hotline" (free for only 60 days) was very helpful. The engineer was helpful in providing fixes for several problems. She gave forthright responses when something could not be done. Only once was it suggested that we should not want to do what could not be done.

The other problems that might be considered with ASYST are ones of taste rather than deficiency. The FORTH style of organization and RPN syntax make much larger demands on the programming team with regard to documentation. While the FORTH community is undoubtedly the most "evangelistic" in its support of a language, there are numerous examples of groups that have switched from FORTH-type languages because of the maintenance difficulties. ASYST does not get around this problem.

This difference between ASYST and other conventional languages should be considered carefully. A group contemplating a switch to ASYST should bear in mind that to achieve the promised power and performance there is a much higher price to pay in training and programming discipline than would be true for a switch to languages such as Pascal or C.

Finally, this reviewer objects to the extensiveness of the copy protection. The protection device is a block that plugs into the parallel port, probably the protection method that is most considerate of the user. However, ASYST copy protection applies not only to the machine used for development but also to every machine that uses the program that is developed. Recently new types of site licenses have become available that make it possible for groups to work together on a software project at a more moderate cost. Nevertheless, at the high cost of this software, a low-cost or royalty-free method of distribution of the compiled application software should be made available.

In summary, ASYST requires a very high cost both in training and for the software. However, it is an extremely flexible system for a very wide variety of interactive data acquisition and processing operations. It is limited somewhat by its moderate performance, inadequate error-handling, and difficulty in handling unforeseen circumstances.