THE UNIVERSITY OF ARIZONA
TUCSON, ARIZONA 85721

DEPARTMENT OF COMPUTER SCIENCE

The Icon Newsletter

Number 24 — June 13, 1987

Welcome to New Subscribers

We've added many new names to our Icon Newsletter subscription list in recent months. If this is the first copy of the Icon Newsletter that you have received, welcome!

A typical issue of the Icon Newsletter includes information about new and existing implementations, reports on applications of Icon, questions and answers from our mail, a programming corner, and announcements of new publications.

The Icon Newsletter is published aperiodically, three or four times a year. At present, subscriptions are free, and we hope keep it that way.

Tabulation of Questionnaires

We have now tabulated the questionnaires returned from Icon Newsletter #22 (686 in all, more than 34% of those sent out). When interpreting the information that follows, keep in mind that it comes only from persons on our mailing list last fall. There is no reason to suppose that it is representative of Icon users in general. For example, it tends to emphasize personal-computer users. Most personal-computer users of Icon probably are on our mailing list, while the opposite probably is true of large multi-user computers.

Also, many persons received this questionnaire in response to a request for information about Icon and had no prior experience with it. Such persons were not then Icon users and were unable to answer many questions.

1. *What are your interests in Icon?*

applications:	498
language design:	345
implementation:	286
teaching:	172

Other interests that were mentioned overlapped significantly with the responses to Question 3 and are included there.

2. *Do you have access to Icon?*

yes: 512 no: 148

What version?

Version 2:	23
Version 5:	433
Version 6:	276

We asked persons to specify the systems on which they used Icon. Many responses were incomplete and ambiguous (our fault for not being more specific). It appears that 298 responders are using Icon on computers running MS-DOS, 94 on VAXes running UNIX, 106 on on other kinds of computers running UNIX, and 80 on VAXes running VMS, with the rest scattered over a variety of computers and operating systems.

3. *Do you use Icon?*

yes: 423 no: 232

What is the level of your use?

high: 80 medium: 97 low: 223

Note: The question was misformatted, making the replies potentially ambiguous. Where the replies were not clearly marked, we tried to interpret them from other information, and we split the difference when there was no basis for interpretation. In any event, the results shown above are subject to question.

What are your applications?

The responses here were many and diverse. The following list includes almost all of them, although some have been paraphrased: AI research, BASIC substitute, CAD/CAM applications, DNA and protein sequencing, DOS batch alternative, SNOBOL4 substitute, algorithm design, algorithmic research, analysis of German literature, analysis of word forms, analysis of writing by disabled children, arbitrary-precision arithmetic, audio software, authoring software, automated testing, automatic program generation, automatic programming, avionics development tools, awk substitute, bibliographical data base processing, bit-mapped graphics, business applications, chess editor, class work, command interpreters, compiler design, computational lexicography, computational linguistics, computer algebra, concordances, cross-reference generation, cryptography and cryptanalysis, customer systems, data base conversion, data

base front ends, data base management, data flow generation, discrete event simulation, document maintenance, document preparation, educational software, experimental psychology, expert systems development, file processing, filters, front-end interfaces, fuzzy set applications, games, genealogies, genetic map specification, grammar processing, historical research, idea management, image processing, industrial automation, intelligent human interfaces, interpreters, label processing, labor tracking systems, language implementation, library automation, library maintenance, list processing, logic programming, mailer software, matrix manipulation, menu systems, music composition, natural language interfaces, natural language processing, natural language translation, networking, office automation, office reports, one-shot applications, operations research, order entry systems, parsing, preprocessors, process control, processing musical scores, programming in the Humanities, programming language translation, prototyping, psycholinguistics, quantitative language analysis, real-time information retrieval, report writing, scientific data analysis, scripts, sed substitute, shell substitute, simulation and modeling, software implementation, software tools, source-code decomposition, statistics gathering, stock market research, symbolic mathematics, syntactic analysis, tables of contents, teaching, test data generation, text utilities, typesetting applications, writing tools.

How do you rate your level of competence as a programmer?

The choice of classification was left to the responders; we have interpreted the responses according to the scale given below. Of course, the results are highly subjective.

In general:

high: 367 medium: 205 low: 23 novice: 9

In Icon:

high: 40 medium: 145 low: 198 novice: 174

The Icon Newsletter

Madge T. Griswold and Ralph E. Griswold

Publishers and Editors

The Icon Newsletter is published three or four times a year, at no cost to subscribers. For inquiries and subscription information, contact:

The Icon Project
Department of Computer Science
Gould-Simpson Science Building
The University of Arizona
Tucson, AZ 85721
U.S.A.

What features of Icon do you find the most difficult to understand or use?

Co-expressions came in high with 54 citations, followed by generators and goal-directed evaluation with 39. Various aspects of string scanning came in a close third with 38 citations. 22 persons cited the syntax of Icon, ranging from the large number of operators to problems with precedence and associativity. 12 persons cited backtracking. Other features mentioned included failure, pointer semantics, tables, and csets.

4. *Do you use the Icon program library?* Responses were tabulated only for persons who indicated they use Icon.

yes: 151 no: 165

5. *Do you need Icon for a computer for which it presently is not available?.*

Almost every computer we'd ever heard of was mentioned, as well as many that were new to us. The most frequently mentioned computers were the Macintosh (68), the IBM 370 (36), the Amiga (19), the Atari (10), the Prime (7), and the DG MV series (6). Note that the Macintosh, Amiga, and Atari implementations became available after the questionnaire was distributed.

6. *What extensions or enhancements to Icon would you like?*

Folks had a lot of fun with this one. The phrasing varied a lot, but the main needs clearly are for an interactive debugger, interfaces with other programming languages (everything from COBOL to Forth), the ability to access system-specific capabilities like graphics and screen handling, and more sophisticated input and output.

Other general aspects mentioned included a faster implementation and a compiler or at least stand-alone object files.

Here is a sampling of requests for language features: a data abstraction mechanism, bit strings, block structure, complex output formatting, control over table organization, conventional arrays, customized character sets, deletion of entries from tables, dynamic linking, indirect referencing in the style of SNOBOL4, named constants, object-oriented constructions, patterns in the style of SNOBOL4, regular expressions in string scanning, reverse string scanning, shared variables among co-expressions, system calls, virtual data structures.

Several persons commented that Icon already is a very large language and that adding to it would detract from its general quality. One person suggested removing (unspecified) features.

7. *Is the available documentation for Icon adequate for your needs?*

yes: 465 no: 49

If not, what additional material would you like?

There is a clear need for more in-depth material, ranging from case studies of large programs to algorithms cast in

Icon — a book on advanced Icon programming techniques, if you will. Of the 49 'no' responses, 22 fell in this category. Other needs mentioned were a book on the implementation (since published), a teaching text, a tutorial, and a reference manual.

8. *What material from the Icon Newsletter do you find most useful?*

The programming corner was the clear winner here, with 209 citations. Next came implementation news with 110. Questions and answers from our mail was third with 45. It is worth noting that many persons said everything was useful, although we did not tabulate the number of these remarks.

The least useful?

The programming corner got 17 citations here, 10 of which specifically found puzzles and arcane techniques annoying. Two wags suggested the only feature of the Newsletter that is not useful was the page numbers.

9. *What additional material would you like to see in the Icon Newsletter?*

Here the largest number of mentions went to descriptions of applications of Icon (47 citations), with more material in the programming corner second (35). The sentiment there was for greater depth and 'more code!' (we thought we detected heavy breathing in spots). There were many other specific suggestions, most of which we will try to accommodate (although we pass on the person who wanted color printing).

10. *Would you be willing to pay a nominal subscription fee for the Icon Newsletter?*

yes: 484 no: 98 maybe: 18

Applications of Icon

As noted in the tabulation of returned questionnaires above, one point of interest for many persons is what applications there have been for Icon. Here are two reports from commercial organizations.

Prototyping at AT&T Information Systems

Owen R. Fonorow, Member of the Technical Staff at AT&T, provides the following description of their use of Icon for software prototyping:

Icon has been used by AT&T Information Systems, Computer Technology Laboratory, Software Technology Group to prototype a large UNIX application. Most of the system has now been converted to C, although a substantial amount of code remains in Icon. The system is held in high regard by our 'customers' (UNIX development projects) judging from the low number of complaints we have received since the December 1986 production release.

Prototyping in Icon provided, among other things, a way to have a completely functional system test suite before one line of C code was written. We also avoided writing English high/low level design documents, and instead used the Icon code as the 'blue print' for our developers to use when converting the system to C.

The issue our management faces now is whether it is 'cost effective' to turn the rest of the system into C. The conversion itself doesn't cost much. The problem is that once the code is converted to C, maintenance costs increase and our ability to make changes in the code is reduced. Here is a quote from a report on the subject:

Finally, the question: "Why prototype in Icon and convert to C later?" has probably occurred to the reader. Part of this question ("Why prototype in Icon?") is best answered in the development history. The five Icon releases (0.0 through 0.4) were produced within nine months, while the single C release spanned six months. The second portion ("Why convert to C later?") must be answered by management. The purpose of this study has been to provide information to help answer that question.

Test Generation at Tartan Laboratories

Bill Wulf, Chairman of Tartan Laboratories Incorporated, provides the following information:

My Christmas project (I treat myself to a coding orgy every year) was a rather sophisticated automatic test-generation system. It's about 15K lines of Icon, and took about a month; frankly, I doubt if I could have done it at all in the Pascal/Ada family of languages.

The system is called TG (Test Generator, imaginative, huh?). It generates a suite of ~300,000 tests for our Ada compilers. Unlike the ACVC tests, they are not designed to test language features so much as the code-generator and run-time system -- in other words, the new components when we retarget/rehost to a new system. As such it is complementary to the ACVCs.

It's fast. I was surprised, frankly, but also quite pleased. The standard mode of operation is to regenerate tests as needed rather than burn the disk space to save them. Right now, our Ada compile speed is respectable, but TG is like a factor of 20 faster — so regenerating the tests adds only ~5% to testing time. Moreover, there is pretty fine-grained control on which tests are generated, so a developer can (re)generate only those tests that pertain to a new (or changed) part of the compiler, tests that failed last time, etc.

It's probably not like many Icon programs — specifically almost no string scanning except in the command line analysis, and a bit in substituting variants of a test into a template. I do make heavy use of iterators, tables, and string names for procedures. Garbage collection, of course, simplifies my life enormously.

Note: If you have an interesting application of Icon, send us a description of it for inclusion in a future Icon Newsletter.

Report from a Conference

Jerry Nowlin, Member of the Technical Staff at AGS Information Services, currently working as a consultant to AT&T Bell Laboratories, sends the following report from a meeting he attended:

I attended the second half of an International Telegraph and Telephone Consultative Committee (CCITT) conference from January 19 through January 23. This conference was a full meeting of CCITT Study Group X. Study Group X is made up of four separate Working Parties. I was invited to give a special presentation on the Icon programming language on January 20 by the chairman of Working Party 2.

The objective of Working Party 2 is to define standards for Systems Support Environments (SSE) for telecommunications systems. The chairman felt that the Icon programming language would be of interest to the members of his working party and to the members of

the other working parties in Study Group X.

During the week I was in Geneva, I attended CCITT Working Party 2 sessions and listened to the presentations on various models being used to define System Support Environments. It takes a great deal of patience and diplomacy to get a consensus from a group of 30 to 40 people from a number of different countries. The formulation of international standards is a long drawn-out process. An actual working session lasts for four years and is composed of several working parties, rapporteurs, and experts meetings per year.

The final two days of this meeting were taken up with plenary sessions. In these sessions the chairmen of the four working parties explained the progress their groups had made during the working party meetings and presented the papers their groups had generated for the entire study group to discuss. The plenary sessions were conducted in English and simultaneously translated into French, German, Russian, Italian and Chinese. It was an interesting experience.

My presentation on Icon was at an after-hours session. While there were no official translations during the presentation, two delegates from China attended and brought their own translators. The presentation consisted of about 30 view graphs and a 50 minute talk followed by a question and answer session. The presentation was attended by about 50 CCITT delegates. There were many requests for further information on Icon after the presentation. In addition to giving copies of the language directly to several attendees, I was able to provide information on ordering further copies of the language and documentation to all interested delegates.

Implementation News

New Implementations

Several new implementations are available from the Icon Project (use the form at the end of this Newsletter to order them):

The Amiga: Rob McConeghy recently completed an implementation of Icon for the Amiga [1]. It requires at least 512K of memory and runs under Version 1.2 of the operating system. Executable files are available now. Source code and the Icon program library will be available later.

The Atari ST: Executable files for the Atari 520 ST and 1040 ST implementation of Icon done by Rick Fonorow and Jerry Nowlin are now available on a distribution diskette that includes Version 7.0 of Jerry Nowlin's public-domain shell, ASH. Source code and the Icon program library will be available later.

The Macintosh: Source code for the MPW implementation of Icon is now available.

MS-DOS: Cheyenne Wills has provided several new functions for the MS-DOS implementation of Icon that allow users to access operating-system and hardware features. These include console input functions, generation of hardware interrupts, storage allocation outside of Icon's regions, access to environment variables, file positioning, and so forth. These new functions take a fair amount of space and are available only for the LMM implementations. To accommodate the variety of user requirements, the LMM distribution of MS-DOS Icon now contains four versions of the Icon run-time executor: fixed versus expandable memory regions, each with and without the additional functions. This distribution now comes on two diskettes.

UNIX PC: Dave Slate has completed an implementation of Icon for the UNIX PC. This implementation has been tested on the AT&T 7300 and also should run on the 3B1. The Icon program library is included. Source code is not available on UNIX PC diskettes, but can be obtained from the basic UNIX distribution or in XENIX format as described below.

XENIX Icon Source Code: Source code for the UNIX implementation of Icon in XENIX *tar* format diskettes is now available. Since configuration files for all supported UNIX systems are included, these diskettes may be useful for transferring Icon source code to systems other than XENIX.

Summary of Existing Implementations

Some interested persons are not aware of all the implementations of Icon that are available. This showed up in responses to the questionnaire, where several persons indicated they needed an implementation that was not available, when in fact the implementation was readily available.

For reference, here's a list of the implementations that are presently available (all are Version 6): The Amiga, the Atari 520 and 1040 ST, computers running MS-DOS 2.0 or higher (IBM compatibility is not required), the Macintosh with 512K or more memory running MPW, and VAX running VMS 4.2 or higher.

There are UNIX[2] implementations for many different computers. The presently supported systems are the Amdahl 580 (UTS), the AT&T 7300 and 3B1 (UNIX PC), the AT&T 3B2/5/15/20, the Codata 3400, the Diab[3], the Gould Powernode, the HP 9000 (HP-UX), the IBM PC/XT/AT (PC/IX and XENIX), the IBM RT PC (ACIS and AIX), the Masscomp 5500, the Microport[3], the Motorola 8000[3], the Plexus P60, the PDP-11 (Version 7 and 2.9bsd), the Pyramid 90x, the Ridge 32, the Sperry 7300[3], the Sun Workstation, and the VAX-11 (4.*n*bsd, Ultrix, System V, and 9th Edition[3]).

Stand-Alone Icon for the Mac

The response to the MPW implementation of Icon on the Macintosh has been less than enthusiastic. Some folks are reluctant to pay for a programming environment just to be able to run Icon. Others just don't want to use a different environment from the one they usually use. Others probably bogged down in the process of getting MPW.

[1] Another implementation for the Amiga by Scott Ballantyne and Gary Sarf is available for downloading from BIX; see listings/amiga there.

[2] UNIX is a trademark of AT&T Bell Laboratories.

[3] These implementations are fairly recent and are not supported in our current UNIX distribution. If you are interested in one of these, contact the Icon Project.

Bob Alexander, who implemented Icon under MPW, contributes the following discussion of the problems involved in producing Icon as a stand-alone application on the Mac:

One solution to the problem of needing MPW to run MacIcon is a public domain mini-shell. It would have to accept and parse commands, load and execute programs in such a way that it can regain control when the program is finished, and provide a run-time environment expected by MPW tool-type programs such as MacIcon (mainly some sort of console). Also helpful would be support for environment variables and the ability to execute commands from a disk file (icont for Mac depends on the latter capability). Unfortunately, I suspect that it's anything but simple to replicate the MPW environment.

Another possibility would be to make each of the Icon tools a double-clickable Mac application. The MPW tool libraries seem to already provide a sort of very primitive console capability. They seem to be oriented toward doing the minimum required to allow error messages to be visible. Stand-alone capabilities of MPW tools are totally undocumented (at least in Version 1.0). The Mac finder passes (via a different mechanism from that used by C's argc and argv) a list of data files that were selected at the time the application (i.e. icon tool) was invoked. There is no way to pass options short of displaying a dialog box with option selections. Same for environment variables.

I don't immediately see an easy way to make icont work in this environment. (There probably *are* ways, but I don't really understand what they are.) Each tool would have to be invoked individually. Files that are selected at the same time as the tool have to be in same window, which equates to same folder (an inconvenience).

For run time, it might be annoying to users to always see a dialog box asking for the memory region sizes, etc., even though all they would have to do is click an OK box. With some additional effort, it could be rigged so that once options have been specified for a program, they could be saved with the icode file (in the Mac's resource fork). The dialog box could be skipped, but could be displayed if the users wanted it, which they could indicate by holding the option key (or something) as they double-clicked (this method is used by some other Mac programs). Also, for run time, arguments would have to be parsed and passed to the program.

The Mac user interface is really neat for many uses, but it falls short for program development. That's probably why Apple decided to make their deluxe development environment (MPW) command-line oriented. There is one development environment called LightSpeed C that Mac folks have taken quite a liking to, but I'm not very familiar with it. It's user interface is supposedly very Mac-like. One thing I'm sure of is that a lot of effort went into its creation.

Documents Related to Icon

Implementation Documentation

The book *The Implementation of the Icon Programming Language*, by Ralph E. Griswold and Madge T. Griswold, first in the new Princeton Series in Computer Science, is now available from the Icon Project as part of a package of implementation documentation. In addition to the book, the package contains a list of changes to the implementation that have been made since the book was written, information that is useful in using the book in conjunction with the source code listings, and corrections to errors in the book.

The price of the package is $40, which includes shipping in the United States. Persons who already have the book and just want the supplementary documentation can obtain it at no cost as described in the next section.

Technical Reports

Three new technical reports related to Icon are now available:

TR 87-2 *A Recursive Interpreter for Icon*, by Janalee O'Bagy. This report describes a new model for the implementation of generators and goal-directed evaluation. For more information, see 'Implementing Generators and Goal-Directed Evaluation' in the section on research in progress.

TR 87-6 *Programming in Icon; Part II -- Programming with Co-Expressions*, by Ralph E. Griswold. This is the second in a series of reports that treats various aspects of Icon in some depth and with an emphasis on programming techniques.

IPD29 *Supplementary Information for the Implementation of Version 6 of Icon*, by Ralph E. Griswold. This report supplements the Icon implementation book as described in the preceding section.

Single copies of these reports are available, free of charge. To get copies, simply list the report numbers as given above on the order form at the end of this Newsletter and write 'free' in the price column. There is no charge for shipping.

Back Issues of the Icon Newsletter

In response to numerous requests, we have reprinted back issues of the Icon Newsletter. They are available on a per-issue basis or as a complete set. See the order form at the end of this Newsletter. *A word of caution:* The first issue consists only of a brief announcement of the inception of the Newsletter. Other early issues contain material that is now obsolete and only of historical interest. On the other hand, here's your chance to be the first person on your block with a complete run of this elusive serial.

From our Mail

How about revising the Icon programming language book to bring it up to date with Version 6?

We have told the publisher that the book needs revising and that we are prepared to do it. The decision is, however, the publisher's, not the authors'. The publisher considers economic matters as well as ones of content.

What about a French translation of the Icon book?

We'd love it. Contact the publisher if you are interested in undertaking such a project.

Can't you advertise? Icon is worth it.

We're not a commercial organization and we don't have an advertising budget. In fact, we can't handle much more 'business' than we have now.

Would you please send me a list of all Icon documents?

That's a tall order. Over the years, there have been literally hundreds of documents related to Icon, ranging from books through articles in journals and technical reports. Of course, much of the early material on Icon is out of date (and out of print). We provide the most relevant material in our distribution packages and announce new documents in these newsletters. There also is a bibliography of documents related to the SNOBOL, SL5, and Icon programming languages, which lists about everything. If you'd like a copy of this bibliography, you can get it, free of charge, by asking for TR 85-13 on the order form at the end of this Newsletter.

Is there an Icon interest group in the United Kingdom?

Not as far as we know. If you'd like to start one, we'd be happy to place a notice in the next Newsletter.

The Icon programming language book is not readily available in the United Kingdom. Is there some problem?

There is no specific problem that we know of. However, relatively few bookstores maintain a large inventory of computer books, so it often is necessary to place an order. This may take some time, and some bookstores may not welcome special orders. This book is available from the Icon Project as noted on the order form at the end of this Newsletter. However, the delay and cost of shipment probably does not make this an attractive alternative.

Isn't $15 a bit much for a diskette of public-domain software?

The cost reflects more than just the diskette. All our diskettes are accompanied by printed documentation, and mailing is provided at no additional cost in the United States. In addition to these costs, we have to recover the costs of hardware and software that is devoted solely to producing distribution material. Anything that is left over after that goes to support new distributions and documentation. We don't (and aren't permitted to) make a profit.

Is there an MS-DOS pop-up window utility that can be used with Icon?

Yes there is — Flash-Up Windows from the Software Botling Company. You load it into RAM and issue commands from a running Icon program to manipulate windows and menus. A caveat: it requires IBM hardware compatibility. The package lists for $90. The best price we've seen is from Catspaw, Inc. See the section titled 'A SNOBOL's Chance' for their address.

Why can't you just distribute one version of the Icon source code that will compile on all the computers for which Icon has been implemented?

Although the basic source code for all implementations of Icon is the same, there are many additional components of the Icon system that are specific to individual operating systems. The differences between operating systems vary all the way from directory structure to path syntax, and each system has a number of system-specific files that augment the basic source code. In the case of UNIX, which has many variants, we have made it possible to configure any variant from the same distribution. However, the Icon distribution for UNIX systems cannot be used to build a VMS version of Icon, and vice versa. Similarly, one cannot build an MS-DOS system with only the files provided in the UNIX distribution. While it would be possible to build a 'super system' from which all implementations of Icon could be configured (we, in fact, have such a system), it would be very large, complicated, and the configuration process itself would be system-specific. Since most users want Icon only for a single system, we have decided to tailor our distribution in this way and not burden everyone with unnecessary size and complexity.

Are you going to produce a LMM MS-DOS Icon with 80286 code, or do I have to do it myself?

We have no plans to add a 80286 (or 80386) version of MS-DOS Icon to the already large number of versions that we distribute. There are many possibilities along these lines and we do not have the resources to pursue them. What we try to do is give priority to providing systems that the largest number of persons can use. The source code is available, so that persons such as yourself can create tailored versions.

Are you doing an implementation of Icon for OS/2?

We're sure there will be a lot of interest in Icon for the new IBM Personal Computer/2 and the facilities it will offer through OS/2. We have no present plans for such an implementation, however — we don't have the necessary hardware or software. If someone would like to provide them for us ... .

I have Icon source code for MS-DOS Icon but seem to be missing the file dos.mac. *Would you please send it to me?*

The file dos.mac is used in the assembly-language portions of Icon (the co-expression context switch and arithmetic overflow checking) when building Icon under Lattice C. If you are using the Lattice 3.2 C compiler, you should have the file. If you are using another compiler, you may have to provide different assembly-language routines. You can skip the assembly-language portions of the implementation altogether by adding

```
#define NoCoexpr
#define NoOver
```

to config.h.

I obtained Icon on a data cartridge to install on our Altos 1086. We were unable to read the cartridge on the 1086 or on an IBM RT PC (admittedly, our RT cartridge drive is a non-standard one). Can you help me?

There should be no trouble reading the cartridges we distribute on an RT PC with a standard cartridge drive or on a Sun Workstation. Beyond that, it's really an experimental question, since there are a variety of cartridge formats and incompatibilities are common. Most persons use cartridges for local backup rather than for transferring information between different computers.

I am interested in a version of Icon for MS-DOS that produces stand-alone load modules (e.g., .COM *or* .EXE *files). Is such a version in the works?*

We understand the need for producing stand-alone executable files from Icon programs. The present implementation emphasizes portability (which is why there is an MS-DOS version at all) but does not lend itself to the production of stand-alone executables. We are working on several approaches to overcoming this limitation, but it is presently premature for us to make any projections about availability.

What is the Icon Project?

We sometimes wonder what image the Icon Project evokes. Is it one of a software house within a university, complete with professional programmers working at rows of workstations, secretaries busily answering telephones, and clerks assembling implementation packages? Is it an image of a dark cubbyhole in the basement of an ancient academic building, with only the glow of a PC illuminating the face of a bleary-eyed hacker? Is it one of a professor's desk littered with books, diskettes, and unopened mail?

You would have to visit us to find out what the Icon Project really is like, but perhaps we can dispel some misconceptions, by describing briefly the Icon Project's origins and activities.

For more years than we care to admit, we have been designing and implementing high-level programming languages. This work began at Bell Laboratories in 1962 (There! We admitted it.) and moved to The University of Arizona in 1971. The first programming language was SNOBOL, followed by SNOBOL3, SNOBOL4, SL5, and finally Icon. Implementations of all of these languages are in the public domain and copies have been distributed to all interested persons.

Over a period of time, these activities became larger and more complicated. At some point a few years ago, it seemed like a good idea to attach names to these activities, so that persons would have recognizable organizations to contact and also so that we could more easily identify and segregate incoming mail. We chose two names: the SNOBOL4 Project and the Icon Project.

So, in part, these names are just labels. What do these projects really do?

It's easy to dispense with the SNOBOL4 Project. It serves only to send out occasional copies of documents and program material related to SNOBOL4. About once a year, it publishes a bulletin that contains recent implementation news and other topical information. The SNOBOL4 Project is not dead, but it is not initiating anything new.

The Icon Project is more active and complicated. It serves primarily to disseminate information and program material about Icon. Unlike the SNOBOL4 Project, the Icon Project is actively producing new material. Icon is an evolving and changing language while SNOBOL4 (at least at The University of Arizona) is not.

As mentioned earlier, both SNOBOL4 and Icon are byproducts of an ongoing research program. The word *byproduct* is important — the research program is not a part of the project, although there is a close relationship between the two and the results of research are reflected in new versions of Icon and its implementation.

The other important point is that the Icon Project is not an organization in the formal sense. It has no official status at The University of Arizona and no employees of its own, nor does it have any officially assigned space, although it has all of these resources in an unofficial way.

What then, does the Icon Project actually do? It distributes documentation and program material. It answers telephone inquiries and responds to electronic mail. It supports an electronic bulletin board and other forms of electronic access. It publishes this Newsletter. And so on. In short, it provides a conduit between the Icon user community and the research program that spawned Icon.

The Icon Project is not large. It does not have the resources of a commercial operation. It cannot solve programming problems that users may have, nor can it guarantee software support (although it tries to do so). It cannot produce new language features on demand nor can it mount major development programs. It does make Icon available in a way that is affordable to most persons and it tries to do it in a professional way.

This is just part of the story of the Icon Project. In the next Newsletter, we'll talk a little about the persons who are associated with the Icon Project and what their roles are.

Research in Progress

Icon is a byproduct of research related to the design and implementation of high-level programming language facilities that focus on non-numerical computation. That research program is an on-going one. Here are brief descriptions of four current research projects at the University of Arizona:

Implementing Generators and Goal-Directed Evaluation

Implementation techniques for traditional languages like Pascal or C are well developed and understood. But this is not true of Icon and other languages with novel expression semantics. The goal of this research is to find elegant and efficient models for implementing generators and goal-directed evaluation. The models should be sufficiently general to apply to languages with similar properties, such as C with generators, the SNOBOL4 pattern-matching mechanism, and so on.

Implementations of traditional languages typically consist of a compiler that translates source code into assembly language for a given machine. The existing implementations of Icon are different in that the translator emits not assembly code, but rather code for a 'virtual' machine; an interpreter is then written to interpret the virtual machine code. Thus there are three points of interest in the implementation of Icon: the design of the virtual machine; the interpreter or compiler for the virtual machine; and static analysis techniques used for improving generated code.

This research focuses on the last two issues. Currently, a new interpreter has been developed that simplifies the implementation of control flow for generators and goal-directed evaluation. Based on the methods used in the interpreter, a compiler is now under development. The compiler is unusual in that it generates C code and not assembly code. The compiler makes use of information about expressions given by a static analyzer, such as whether generators are present and how much space an expression requires during evaluation, in order to improve its generated code. — *Janalee O'Bagy*

Type Inference

Variables in Icon are untyped. That is, a variable may take on values of different types as the execution of a program proceeds. In the following example, x contains a string after the read, but it is then assigned an integer or real, provided the string can be converted to a numeric type.

```
x := read()
if numeric(x) then x +:= 4
```

In general, it is impossible to know the type of an operator's operands at translation time, so some type checking must be done at run time. This type checking may result in type conversions, run-time errors, or the selection among alternate operations (for example, the selection of integer versus real addition). In the current implementation of Icon, all operators check all of their operands. This incurs significant overhead.

Much of this run-time type checking is unnecessary. An examination of typical Icon programs reveals that the type of most variables remains consistent throughout execution (except for the initial null value) and that these types can often be determined by inspection. Consider

```
if x := read() then
   y := x || ";"
```

Clearly both operands of || are strings, so no checking or conversion is needed.

The purpose of the type inference project is to explore various methods for gathering, at translation time, information about the types of values variables may have at run time. This information will allow the translator to omit type-checking code in some places where it is not needed, thus producing more efficient code. Different type inference schemes offer trade-offs in terms of the quality of information they gather and in the cost of gathering that information. — *Ken Walker*

Pattern Matching in Real Time

Motivated by a desire to provide the power and flexibility of SNOBOL4 and Icon to the application area of communications programming, this work undertakes to analyze and suggest modifications to Icon's current goal-directed evaluation mechanisms in order that these mechanisms might execute with real-time response. In this context, real-time response is defined as response that is bounded by a small constant (any operation that might invoke Icon's garbage collector does not, for example, meet this criterion).

The challenge of providing goal-directed pattern matching inside a real-time environment is to bound execution times of built-in language primitives by small constants and to facilitate analysis of user-defined language capabilities in terms of worst-case response time. This research focuses on three distinct areas:

Integration of Pattern-Matching Hardware — The use of special-purpose hardware frequently speeds the matching of common patterns. Research efforts here are dedicated to providing a clean interface between existing language mechanisms and available hardware.

Control Structures — Icon's goal-directed evaluation traverses the search tree of possible solutions in a depth-first manner. The goal of this research is to provide language mechanisms that allow programmers to prune the search tree without violating the general paradigm of goal-directed evaluation or complicating an understanding of a program's behavior.

Data Structures — In many real-time applications, pattern matching is performed on a potentially endless stream of characters arriving from some remote source. This research attempts to determine whether special data structures might facilitate the representation of this stream of characters and simplify pattern matching operations on the data represented by the stream. — *Kelvin Nilsen*

A New Language?

Icon is called a 'high-level language' because it supports many operations that have no close correspondence at the machine level. Some of these operations involve Icon's expression-evaluation mechanism, which is completely different from the evaluation mechanism of the underlying machine. The purpose of high-level languages is to free programmers from the details of the machines they are working on and to allow them to express higher-level abstractions clearly. Icon has proven very useful in this area, as have a number of quite different languages.

Other languages of interest are FP and Prolog. FP is a functional programming language. Programs in FP consist of a set of function definitions; there are no procedures, only functions. Prolog is a logic programming language, where programs consist of logical assertions and questions. FP and Prolog are different from Icon in that programs in these languages are more *declarative*. That is, programs in FP and Prolog do not specify *how* to get the answer, they specify *what* the answer is.

There are a lot of problems that are easier to solve in Icon than in FP or Prolog, but there are a lot of other problems that are easier to solve in FP or Prolog than in Icon. This research is aimed at finding ways to combine the advantages of the three types of language in a way that is simple and powerful. The goal is to make a programming language that is good for solving a very large class of problems. The language will contain some aspects of Icon's evaluation mechanism, some of FP's notation for defining functions, and *unification*, Prolog's very powerful form of assignment. — *Dave Gudeman*

The Icon Project

For general information and ordering, call (602) 621-6613 and ask for Beth. For technical help, write to

The Icon Project
Department of Computer Science
Gould-Simpson Science Building
The University of Arizona
Tucson, AZ 85721
U.S.A.

or send electronic mail to:

icon-project@arizona.EDU

or

... {allegra, ihnp4, noao}!arizona!icon-project

To get program material electronically, download from our BBS, (602) 621-2283, or FTP from /usr/ftp/icon on arizona.EDU.

The Icon Program Library

Judging from questions that we have received, we haven't done a very good job of describing the Icon program library. The library consists two parts: programs and collections of procedures. The programs are diverse and include everything from demonstrations and games to text-processing utilities. One of the more elaborate and interesting programs in the library is a random sentence generator that takes grammatical specifications as input and produces randomly-selected sentences from the corresponding language. The uses of this program range from the generation of test data to poetry. There also are more mundane programs, such as ones for formatting mailing labels and sorting address lists by ZIP codes. Persons developing Icon programs may find the Icon cross-reference and procedure sorting programs of interest.

The collections of procedures provide a way of extending the built-in repertoire of Icon. They include a math library, procedures for doing bit operations, radix conversion, performing complex arithmetic, operating on strings and structures, — you name it.

Several persons who responded to the questionnaire indicated they found the Icon program library valuable in learning to program in Icon, although one person noted, fairly, that the programs presently are not well documented.

Material in the Icon program library comes from many authors and is a kind of communal enterprise. Contributions are welcome. We require material in machine-readable form. Adequate documentation is required; sample data or test programs should be provided also. The final decision on the inclusion of contributions rests with the Icon Project. In the past we have accepted most contributions, although lack of adequate documentation frequently delays inclusion.

Electronic mail, for persons who have access to it, is probably the easiest way to submit material. We also can handle most magnetic media from UNIX, MS-DOS, and Atari ST systems.

We presently are accumulating material for a new version of the Icon program library. This is going slower than we hoped, and it looks like the new release will be late this calendar year. There still is plenty of time to submit new material.

Programming Corner

As indicated by responses to the recent questionnaire, the programming corner is one of the most popular parts of the Icon Newsletter. It seems, however, that everyone has different ideas about what it should contain. Some want simpler programs. Others want more complicated ones. Some want larger programs. 'Puzzles' are the most controversial; some readers are turned off by them, while others thrive on them.

Obviously we cannot please everyone simultaneously. Also, the size of the programs that we can present here is limited by our publication format, not to mention the time and effort it takes to explain larger programs. We will continue to present program material here that covers a range of difficulty — and hope that our readers will be tolerant of those things that don't suit their particular interests.

Correction

An initial clause was omitted from one of the procedures in the program for dealing bridge hands in the last Newsletter. The beginning of the procedure display should have looked like this:

```
procedure display()
   local layout, i
   static bar, offset

   initial {
      bar := "\n" || repl("-",33)
      offset := repl(" ",10)
      }
```

The procedure works properly without the initial clause enclosing the assignments to bar and offset, but the assignments are performed every time display is called, rather than just on the first call.

It's worth noting that this kind of error is fairly common, as is the less benign error of omitting the static declaration while using the initial clause.

Processing Command-Line Options

In the last Newsletter, there was a discussion of using command-line arguments to Icon programs to provide options that could be used to control program execution. For example, the program for dealing bridge hands might have options for specifying the number of hands to deal as well as the seed for random-number generation. Using the UNIX style for specifying options, the program might be called as follows to produce 10 hands with an initial seed of 17:

```
deal -h 10 -s 17
```

Of course, there is nothing sacred about the UNIX style for communicating options, although we've found it handy in the Icon program library to have a relatively uniform method.

In the present Icon program library, every program that has command-line options handles them in its own way. Bob Alexander has provided a procedure that avoids this duplicate code and provides more uniformity to the handling of options. It is patterned after the UNIX getopt facility and is called as getopt(arg,optstr), where arg is the argument list as passed to the main procedure and optstr is a string of allowable option letters.

If a letter is followed by ":", the corresponding option is assumed to be followed by a string of data, optionally separated from the letter by space. If instead of ":" the letter is followed by a "+", the parameter is converted to an integer or if a ".", to a real. If optstr is omitted, any letter is assumed to be valid and to require no data.

The procedure getopt returns a list consisting of two items. The first is a table of the options specified, where the entry values are the specified option letters. The assigned values are the data following the options, if any, or 1 if the

option has no data. The table's default value is null. The second item in the returned list is a list of remaining parameters on the command line (usually file names). A "–" that is not followed by a letter is taken as a file name rather than an option.

If an error is detected, stop() is called with an appropriate error message. After calling getopt() the original argument list, arg, is empty.

Not every program that has command-line options needs all the generality that this procedure provides. For example, the program for dealing bridge hands in the last Newsletter needs only the table in the first element in the list returned by getopt:

```
link getopt
        .
        .
        .
procedure main(args)
   local s, hands, opts
        .
        .
        .
   hands := 5

   opts := getopt(args,"h+s+")[1]
   hands := \opts["h"]
   &random := \opts["s"]
        .
        .
        .
```

The option string indicates that the allowable option names are h and s and that both require integer arguments. The default values of hands and &random are changed only if the corresponding values are non-null (that is, they are specified on the command line).

Here's the code for the procedure itself:

```
procedure getopt(arg,optstr)
  local x, i, c, otab, flist, o, p
  /optstr := string(&lcase ++ &ucase)
  otab := table()
  flist := []
  while x := get(arg) do
    x ? if ="–" & not pos(0) then
      while c := move(1) do
        if i := find(c,optstr) + 1 then
          otab[c] := if any(':+.',o := optstr[i]) then {
            p := "" ~== tab(0) | get(arg) | stop(...)
            case o of {
              ":": p
              "+": integer(p) | stop(...)
              ".": real(p) | stop(...)
              }
            }
          else 1
        else stop(...)
      else put(flist,x)
  return [otab,flist]
end
```

In the actual procedure, the calls to stop contain appropriate error messages. The text was elided here to make the program fit into the space available. With apologies to Bob, we also shortened an identifier here and there and rearranged things somewhat toward the same cause.

Efficient Programming in Icon

Since many of Icon's features have no direct correspondence in the underlying architecture of the computers on which Icon is implemented, there often is no way of knowing, from the language itself, whether a particular operation is efficient or inefficient. For example, we have been asked several times whether computing the size of a string (*s) is fast or slow.

The ultimate source of such information is in the implementation itself. Persons who are interested in implementation techniques may want to study the implementation book or even the source code. Of course, an Icon programmer should not have to be an expert in the implementation of the language in order to know how to program efficiently. While a general treatment of this subject is complicated and extensive, a few specific suggestions go a long way toward providing the information needed most frequently.

This section of the Programming Corner is devoted to such issues. More material on efficient coding techniques will appear on a more or less regular basis in future Newsletters.

The size of an object: Let's start with the answer to the question above: 'Is determining the size of a string fast or slow?'. The answer to this one is simple: It is fast. The reason is that the size of a string is stored as part of the string value and is immediately accessible. (This is in contrast to C, where it is necessary to count the characters every time the size of a string is needed.)

In fact, the sizes of all objects are stored with them and are quickly available. For objects that may change in size, such as lists, sets, and tables, the size is updated whenever it is changed. So *x is fast, regardless of the type of x.

Don't get carried away with this, however:

```
*s = 0
```

is slower than

```
s == ""
```

simply because the former requires two operations.

Augmented assignment: Icon provides augmented assignment operations such as

```
i +:= 1
```

as shorthand for

```
i := i + 1
```

In fact, there are augmented assignment versions of all binary operations except assignments themselves, although many are rarely used.

Augmented assignments are not just abbreviations; they are more efficient that the non-augmented forms, since the variable to which the assignment is made is only referenced

once. With just a simple variable like an identifier, the difference is minor. With a computed variable, such as a table reference, the difference may be quite significant.

For example, if t is a table, it is better to write

```
t[x] +:= n
```

than to write

```
t[x] := t[x] + n
```

While the amount of time it takes to look something up in a table is hardly obvious on the face of it (but will be discussed in a subsequent Newsletter), it does not take a lot of imagination to realize that it must depend on the size of the table and possibly what's in it — and if the table is huge, the time to find something in it may be considerable.

Operations on lists: Since lists can be accessed both by position and as deques (stacks and queues in combination), it is worth thinking about alternative possibilities when performing list operations. Consider, for example, the problem of circularly rotating the elements in a list left by one. The typical code for this for a Pascal-like language, cast in Icon is:

```
first := a[1]
every i := 1 to *a - 1 do
   a[i] := a[i + 1]
a[*a] := first
```

Looping though the list like this is certainly straightforward, if a bit tedious. If you want to get more idiomatic, it can be rewritten more concisely. But what about thinking of the list as a deque and writing

```
put(a,get(a))
```

That is, take an element off the left end of the list and put it on the right end.

Certainly the second approach is more concise, but what about efficiency? To begin with, it seems plausible that the looping approach takes an amount of time that is approximately proportional to the size of the list[4]. But what about taking an element off one end of a list and putting it on the other end? Does Icon go through the looping approach internally? Or does it do something cleverer and more efficient? (This question might lead another: How would *you* go about implementing lists with both positional and deque access?)

The answers to some of these questions are given in some detail in the Icon implementation book. Suffice it to say here that Icon uses a fairly sophisticated method for implementing lists that balances the costs of the different kinds of access. Both put and get are fast. In fact, get(a) is slightly faster than a[i]. Furthermore, the rotation above, using put and get, is essentially independent of the size of the list.

[4]This is true if the list is constructed all at once. The situation is more complicated if the list is built by push() or put(). See the Icon implementation book.

To get an idea of the difference between the looping and deque methods, here are comparative timings for rotating a list created by list(n), where n has the values 100 and 1000:

n	100	1000
loop	35.76	356.81
deque	0.16	0.16

The moral should be clear: When you're programming in Icon, think in Icon.

Pattern Words

We recently received a list of available publications from Aegean Park Press, which publishes a wide variety of material related to cryptography. Among their current offerings are two books of 'pattern words'. The pattern word for a word is obtained by replacing all instances of the first letter of the word by A, all instances of the second letter by B, and so on. For example, proposals has the pattern word ABCACDEFD. (Can you find another word with the same pattern word?)

So, here's a small problem: Write an Icon procedure patword(s) that returns the pattern word corresponding to the value of s. For simplicity, you can assume that s may contain characters other than letters, that upper- and lowercase letters are distinct, and that there are (say) no more than 26 different characters in s. Think about different programming techniques, keeping in mind the moral given at the end of the preceding section. Strive for a solution that is both efficient and elegant. We'll leave the definition of elegant to you; our ideas on this will be included with solutions, which will appear in the next Newsletter.

Using Pipes — for UNIX Users

One of the most elegant and powerful features of UNIX is the pipe, which allows the output from one program to be fed as input to another program. Some other operating systems 'fake' this by writing all the output of the first program to a temporary file, and then using that temporary file as input to the second program. That's not the real thing (suppose the first program produces an enormous amount of output or even does not terminate), but this is not a forum for arguing about operating systems. If you are not familiar with pipes, however, you might want to study this feature.

UNIX implementations of Icon support the reading and writing of pipes. The real power of pipes in Icon lies in being able not only to access UNIX commands (which can be done with the system function) but also to pass data between the program and the commands. The technique is very similar to opening a file for reading or writing, except a command string instead of a file is opened. In the case of reading, output from the pipe becomes input to the Icon program, while in the case of writing, output from the Icon program becomes input to the pipe.

Here's an example:

```
names := open("ls", "pr")
```

The first argument is the command, and the p in the second

argument indicates the first argument is a command, not the name of a file. The file assigned to names can then be read like any other file; each read produces a line of output from ls. For example, the following loop writes out the names of the files as produced by ls:

```
while write(read(names))
```

The loop terminates when the output from ls terminates, exactly as if a file had been read.

The command that is opened as a pipe can be any UNIX command. For example,

```
path := read(open("pwd","p"))
```

assigns to path the path to the present working directory. Notice that since opening for reading is the default, the r need not be specified in the option.

Pipes can be opened for writing also, as in

```
sortout := open("sort >out.sort", "pw")
```

which causes output written to sortout to be sorted and written to the file out.sort.

Like other files, pipes should be closed when they are no longer needed.

Here's a problem for UNIX gurus: Write an Icon procedure getenv(s) that returns the value of the environment variable s if it is set but fails if it is not. If you manage to distinguish between environment variables that are not set and those that are set to the null value, we'd like to see how you did it.

Puzzles and Such

Andrew Appel at Princeton University sent us this program and asked us if we could guess what it does:

```
procedure main()
   every write((i := 2 | (|i := i + 1)) &
      (not(i = (2 to i) * (2 to i))) & i)
end
```

Ken Walker contributes the shortest self-reproducing Icon program that we know of:

```
procedure main();x:="procedure main();x:= \nx[21]:=image(x);write(x);end"
x[21]:=image(x);write(x);end
```

We had to set that one in 6-point type to make it fit within our double-column format. We figure 6 points is about what it deserves.

Upcoming in the Newsletter

The following topics are scheduled for inclusion in the next Newsletter:

- The first in a series of articles on the history of Icon.
- A discussion of what is involved in adding new functions to Icon.
- What's in the works for extensions and improvements to Icon.
- More about the Icon Project.
- More contributions from our readers (we hope).

Our Army

We have mentioned in previous newsletters that many persons have contributed to Icon. These persons are scattered over the world and provide support to the Icon Project freely. Without this 'volunteer army', Icon would not have been nearly as successful as it is. It is a practical impossibility to acknowledge every contribution here — sometimes we do not even know who the contributors are. However, here are a few of the most significant contributions in recent months:

Corrections and improvements to Icon source code: Bob Alexander, Rick Fonorow, Andy Heron, Jerry Nowlin, Rob McConeghy, and Charles Richmond; *Porting to new computers*: Bob Alexander, Rick Fonorow, Rob McConeghy, Jerry Nowlin, and Charles Richmond; *Configuration files for new UNIX implementations:* Bob Alexander, Rob Asen, Andy Heron, Kevin Johnson, Andy Puchrik, and Dave Slate; *Additions to Icon's function repertoire:* Bob Alexander, Andy Heron, and Cheyenne Wills; *Contributions to the Icon program library:* Bob Alexander, Jerry Nowlin, Kenneth Sykes, Steve Wampler, Kurt Welgehausen, and Cheyenne Wills; *Other technical assistance:* Bob Alexander, Tom Hicks, Dave Slate, Steve Wampler, and Cheyenne Wills.

This list does not include persons at the University of Arizona who are directly associated with the Icon Project or the many persons who have sent helpful suggestions via electronic and postal mail.

A SNOBOL's Chance

Many persons who receive this newsletter also receive the SNOBOL4 Information Bulletin published by the SNOBOL4 Project and are familiar with that language. Some of you, however, may never have heard of SNOBOL4, Icon's ancestor. If you don't know about SNOBOL4 or only think of it as an old, archaic programming language, you may be missing something.

SNOBOL4 defies description in a few words. To give some idea of what it's about, it has string pattern matching at a higher level of abstraction than Icon's string scanning, it has sophisticated data structures, it has all the run-time flexibility of Icon and then some — even the ability to create and execute new program text during execution, and it has automatic storage management. In many respects, SNOBOL4 is a higher-level language than Icon, and more powerful.

If you're a SNOBOL4 fan already, or if we've piqued your interest, you should know about another newsletter, *A SNOBOL's Chance*, which is published by Mark Emmer of Catspaw, Inc. The first issue of this newsletter appeared last fall, followed by the second this spring. The next issue is

scheduled for publication in July.

A SNOBOL's Chance contains a variety of information about SNOBOL4 and related topics. To give you an idea of its content, here are some of the subjects that were covered in the second issue: the announcement of an electronic bulletin board, a national SNOBOL4 electronic conference, coverage of the ICEBOL '86 conference on the applications of SNOBOL, a discussion of SNOBOL4's unevaluated expressions, the announcement of the availability of a fast implementation of SNOBOL4 for the Motorola 68010, a discussion of string-processing philosophy, a description of the Proximity board for approximate pattern matching, and a discussion of new ideas about right-to-left pattern matching. *A SNOBOL's Chance* also contains a listing of programs, books, and machine-readable text that are available from Catspaw.

Above all, the newsletter is very well done. If you have an interest in SNOBOL4 or text processing in general, we encourage you to investigate *A SNOBOL's Chance*. Free sample copies are available on request from

Catspaw, Inc.
P.O. Box 1123
Salida, CO 81201

(303) 539-3884; BBS: (303) 539-4830

Ordering Icon Material

Shipping Information: The prices listed on the order form at the end of this Newsletter include handling and shipping in the United States, Canada, and Mexico. Shipment to other countries is made by air mail only, for which there are additional charges as follows: $5 per diskette package, $10 per tape or cartridge package, and $10 per documentation package. UPS and express delivery are available at cost upon request.

Payment: Payment should accompany orders and be made by check or money order. Credit card orders cannot be accepted. Remittance *must* be in U.S. dollars, payable to The University of Arizona. There is a $10 service charge for arganizations that are unable to pre-pay orders may send purchase orders, but there is a $5 charge for processing such orders.

What's Available

Icon Program material falls into four categories: UNIX, VMS, personal computer, and porting.

The UNIX program package contains source code, the Icon program library, documentation in printed and machine-readable form, test programs, and related software — everything there is. It can be configured for all the UNIX systems mentioned in the summary earlier in the Newsletter, and new configurations generally are easy to develop. The documentation includes installation instructions, an overview of the language, and operating instructions. It does not include either of the Icon books. Program material is provided on magnetic tape or cartridge.

The VMS program package contains everything the UNIX implementation contains except UNIX configuration information and UNIX-specific software. However, the UNIX and VMS systems are configured differently, and neither will run on the other system. The VMS package is distributed only on magnetic tape.

Icon for personal computers is distributed on diskettes. Because of the limited space that is available on diskettes, in most cases there are separate packages for the different components: executable files, source code, and the Icon program library. Each package contains printed documentation that is needed for installation and use.

Icon for porting is distributed on MS-DOS format diskettes. There are two versions, one with a flat file system and one with a hierarchical file system. Both versions are available in either plain ASCII format or compressed 'ARC' format.

There are two documentation packages that contain more than is provided with the program packages: one for the language itself and one for the implementation. These documentation packages contain the language and implementation books, respectively, together with supplementary material.

When ordering, use the codes given in parentheses at the ends of the descriptions that follow.

Program Material

UNIX Icon: Tapes are $25; specify *cpio* or *tar* format and 1600 or 6250 bpi (UT). Cartridges are $40 (DC 300 XL/P, raw mode only); specify *cpio* or *tar* format (UC).

VMS Icon: Tapes are $25; specify 1600 or 6250 bpi (VT).

Icon for Personal Computers:

Amiga Icon executables: one 2S/DD 3.5" diskette, $15 (AME).

Atari Icon executables: one 1S/DD 3.5" diskette, $15 (ATE).

Macintosh Icon executables: one 1S/DD 3.5" diskette, $15 (ME).

Macintosh Icon source: one 2S/DD 3.5" diskette, $15 (MS).

MS-DOS SMM Icon executables: one 2S/DD 5.25" diskette, $15 (DE-S).

MS-DOS LMM Icon executables: two 2S/DD 5.25" diskettes, $20 (DE-L).

MS-DOS Icon source and test programs: two 2S/DD 5.25" diskettes, $25 (DS).

MS-DOS Icon program library: one 2S/DD 5.25" diskette, $15 (DL).

PC/IX Icon executables: one 2S/DD 5.25" diskette, $15 (PCE).

PC/IX Icon source and test programs: four 2S/DD 5.25" diskettes, $35 (PCS).

PC/IX Icon program library: one 2S/DD 5.25" diskette, $15 (PCL).

UNIX PC Icon executables and program library: one 2S/DD 5.25" diskette, $20 (UPEL).

XENIX Icon SMM executables: one 2S/DD 5.25" diskette, $15 (XE-S).

XENIX Icon LMM executables: one 2S/DD 5.25" diskette, $15 (XE-L).

XENIX Icon source and test programs: five 2S/DD 5.25" diskettes, $40 (XS).

XENIX Icon program library: one 2S/DD 3.25" diskette, $15 (XL).

Icon for Porting:

Flat file system, ASCII format: four 2S/DD 5.25" diskettes, $35 (PF-A).

Flat file system, ARC format: two 2S/DD 5.25" diskettes, $25 (PF-K).

Hierarchical file system, ASCII format: four 2S/DD 5.25" diskettes, $35 (PH-A).

Hierarchical file system, ARC format: two 2S/DD 5.25" diskettes, $25 (PH-K).

Documentation

Language documentation package: $29 (LD).

Implementation documentation package: $40 (ID).

Back issues of the Icon Newsletter: $.25 each for single issues (specify numbers), $5.00 for a complete set (#1-23) (NL).

Order Form

Icon Project • Department of Computer Science • Gould-Simpson Building • The University of Arizona • Tuscon, AZ 85721 USA

(602) 621-6613 • BBS: (602) 621-2283

name _____

address _____

_____

city _____ state _____ zipcode _____

(country) _____

telephone _____

☐ check if new address

qty.	code	description	price	total

subtotal	
extra shipping charges	
purchase-order processing	
other charges	
total	