THE UNIVERSITY OF ARIZONA
TUCSON, ARIZONA 85721

DEPARTMENT OF COMPUTER SCIENCE

Icon Newsletter #15

Ralph E. Griswold

June 7, 1984

1. Results from the Questionnaire

Over 25% of the questionnaires included in Icon Newsletter #14 have now been returned. The results are tabulated below, followed by a discussion of some of the comments that were sent in.

Newsletters mailed: 650 Questionnaires returned: 169

Interests:

programming	122
language design	120
implementation	70
theoretical issues	56
teaching	54

Other interests mentioned included a variety of applications such as artificial intelligence, text processing, natural language processing, simulation, psychology, molecular genetics, and number theory.

Implementations Used:

Version 2:

CDC Cyber/6000	5
CRAY-1	2
DEC-10	5
DG MV8000	1
Honeywell DPS-8	2
HP 3000	1
IBM 360/370	8
VAX/VMS	3

Version 5:

Onyx C8002	2
PDP-11	22
VAX/UNIX	61
VAX/VMS	35

Level of Use:

high:	8
medium:	22
low:	72
none:	43

A number of persons remarked that they did not have access to Icon or that they were currently in the process of installing it.

Content of the Icon Newsletter:

Several persons commented that they liked the content and structure of the Newsletter as it stands. Suggestions for material to include in future issues fell into two general categories: (a) specific topics and (b) subjects that might be covered on a regular basis. Some of the specific topics follow. Items flagged with an asterisk are covered in this Newsletter.

Applications of Icon in various areas, such as artificial intelligence and office automation.

Comparison of the relative efficiency of alternative programming techniques; space/time tradeoffs.

Pattern-matching techniques, including comparisons with SNOBOL4, Awk, and Sed.

Portability of Version 5.*

Design issues; tradeoffs; evaluation of the design of Icon.

Icon's virtual instruction set and intermediate code.

The use of Icon in computer science courses.*

Information about Icon users.*

The impact of Icon on other programming languages.

Among items that might appear regularly, there was considerable interest in the presentation of more programming examples, including ones for novices and larger, annotated programs. Some material of this kind can be found on the Version 5 Icon distribution tape. Both the UNIX* and VMS distribution tapes include demonstration programs and all the program material from the Icon book. The UNIX distribution tape also includes a moderately large library of Icon programs ranging from text processing utilities to games. In addition, some recent technical reports include examples of large programs (see the **New Documents** section in this Newsletter). In any event, more programs will be included and discussed in future Newsletters.

Many readers mentioned items that fall under the general classification of status reports:

new implementations*
bug reports*
new publications*
status of the Icon program library
availability of updates
work in progress*

Material of this type generally has been covered in the Newsletter and will continue to be covered in the future. See this Newsletter for material on subjects that are flagged above with an asterisk.

One of the biggest areas of interest concerned what other persons are doing with Icon. User-contributed articles were suggested. We will be happy to publish material submitted by users — if we can get it. For work in progress, a paragraph or two is appropriate (see material elsewhere in this Newsletter). We also are willing to publish longer contributions, space permitting. If you have something of potential interest, please send it in — there are many interested readers.

The programming corner is popular with readers, although some readers would like to see more introductory material to balance the arcane subjects and trivia (of course, some readers want more trivia also).

2. Version 5 Implementation News

Mark Langley of Science Applications Inc. has implemented Version 5 under Eunice, a UNIX emulator for VMS. We expect to have a distribution tape available soon.

John Pollack of Chemical Abstracts Service has adapted Version 5 to run under IS/3, a UNIX emulator for VMS derived from UNIX System III by INTERACTIVE Systems Corporation. Steve Wampler at Northern Arizona also has successfully installed Version 5 under System III. We have incorporated several modifications suggested by them to make other System III installations easier. More information on installing

*UNIX is a trademark of Bell Telephone Laboratories.

Version 5 of Icon under System III is available from:

Professor Stephen B. Wampler
College of Engineering
Box 15600
Northern Arizona University
Flagstaff, AZ 86011

Incidentally, Version 5.7 of Icon, which is included on the Berkeley UNIX 4.2bsd tape as a user-contributed program, is out of date. In fact, some persons have had problems installing it, since modifications were made to 4.2bsd following the release of Icon Version 5.7. There also have been a number of corrections and additions made to Icon since Version 5.7. Anyone using Version 5.7 should get Version 5.8. See Icon Newsletter #12 or *Implementations of Icon* for ordering information.

Several readers keep hoping to hear that Version 5 of Icon has been implemented on other computers. Work is in progress on several such implementations, and persons have received source code for projected implementations on the following computers:

AT&T 3B20	IBM 370 (UNIX)
DG MV8000	IBM PC (DOS and UNIX)
Gould Concept	Perkin-Elmer 3200
HP 3000	Prime 450
HP 9000	Pyramid 90x
Honeywell DPS-8	

Notes on some of the returned questionnaires indicated progress on some of these implementations, although there is no definite news yet. To get some idea of the problems involved, see the section **Porting Version 5 of Icon** that follows.

3. Bugs in Version 5 VAX/VMS Icon

There is a storage allocation problem in the VAX/VMS implementation of Version 5.8 that can cause programs to run out of string or heap space after a file is opened. This is caused by the fact that VMS may allocate memory in the program's data space. VAX-11 C is smart enough to realize that this has happened, and it then prevents Icon's data space from growing. This problem is compounded by an incorrect space computation in the Icon storage management system that makes it think that it needs more space than it does.

This problem can be circumvented by increasing the initial allocations of space for Icon's data regions. See the Icon HELP file.

The bug can be fixed by adding

```
IOSEGMENTS=32,NOP0BUFS
```

to the linker options file and relinking Icon.

We hope to have a new release of VMS Icon soon with these problems resolved.

4. Work in Progress

Many readers are interested in what is going on and what to expect next. For one thing, do *not* expect major changes in Icon. Icon is a byproduct of a research program, not an end in itself. Version 5 is intended to be a relatively stable language. Most of our current work involves the use of Icon and the exploration, on an experimental basis, of language design issues related to it. In other words, Version 6 is not on the horizon.

Version 5.9

We have accumulated a number of minor improvements to be made in Version 5 of Icon and plan to produce Version 5.9 to supercede the current Version 5.8 sometime later this year. One area in which substantial improvement in performance will be made is in the implementation of tables.

We also plan to abandon the compiler version of Icon and only distribute the interpreter in the future. Compared to the interpreter, the compiler presently offers a minor advantage in execution speed at the expense of considerably increased time to get into execution. The main motivation for using the compiler at present is to be able to add external functions, written in C, to augment the built-in repertoire of Icon. Bill Mitchell recently has developed a "personalized interpreter" system for Icon that allows functions to be added easily and quickly to the interpreter version. With a personalized interpreter, there is very little motivation to retain the compiler, but there are many reasons for discontinuing it. For one thing, it adds a large maintenance, testing, and distribution burden — time and effort we otherwise could expend in more useful ways. We plan to continue to distribute the Version 5.8 compiler to persons who specifically request it, but those of us who have worked with the personalized interpreter (including students in a course at the University of Arizona on "Icon Internals") are prepared to tell you how great it is.

Sets in Icon

The Icon Internals class mentioned above undertook, toward the end of the course, the addition of a set data type and set operations to Icon. This offered the students an opportunity to participate in language design and to test their understanding of the Icon implementation by making a significant modification to it. The project turned out well and as a result, we have the nucleus of a significant extension to Icon. The details of sets as they are realized in Icon will be described in the next Newsletter. Sets probably will be included as an experimental extension in the Version 5.9 release.

Production Icon

Steve Wampler at Northern Arizona University provides the following description of a project he is working on:

> The Icon programming language was developed as a research tool with which to examine new approaches to processing data structures. It contains a large number of unusual features, often at some cost in efficiency. Some of its more novel features are not necessarily inefficient. However, it is difficult to measure their true cost because of the costs of other features within their context. The π ("Production Icon") programming language is an attempt to develop the generator-based evaluation found in Icon in an efficient, production-level language. Previous attempts to add generators into production-level code have concentrated on embedding generators into existing languages (for example, Cg). π is intended to be an efficient, uniform language retaining as much of the expressiveness of Icon as possible.

Generators in Object-Oriented Languages

Tim Budd at the University of Arizona, who implemented Cg, is currently studying object-oriented languages:

> Icon is not the only language in which the formal manipulation of generators and sequences can be expressed or investigated. One current project involves the expression of generators in the language Smalltalk-80. In Smalltalk terms, a generator is any object that responds to the messages **first** and **next** by producing a sequence of values, terminated by the special symbol nil. As is the case with Icon, there are two different ways one can view generators. The first view, what we might call the implementation view, is that of an object producing a succession of values. The alternative, what one could call the data view, is that of a sequence, perhaps infinite, and a pointer to the "current" value. One can then describe operations using generators either by their actions at the low level, in terms of first and next messages, or at a higher conceptual level, in terms of how they combine sequences.
>
> The power of generators in Smalltalk is in the ease and flexibility of defining ways of combining sequences (or generators) to form new sequences (or generators). For example, one very powerful combining operator has been defined which can be used to express such operations as catenate (appending one sequence to the end of another), shuffle (mixing two sequences), combining (adding the elements of two sequences, for example), and others. One aspect of current research involves defining pattern-matching operations in terms of operations on sequences.

5. Use of Icon in Computer Science Courses

Icon has been introduced in a number of computer science courses. Typically, it replaces SNOBOL4 in comparative programming languages courses. Institutions teaching Icon include Carnegie-Mellon University, Grinnell College, Illinois Institute of Technology, New Mexico Tech, Rutgers University, the University of Arizona, the University of Michigan, the University of Pittsburgh, and Vanderbilt University (this information comes from a number of sources and may not be completely accurate; it certainly is not complete). Ed Gehringer of CMU provides the following report on his experience with teaching Icon:

> Icon was one of three languages emphasized in my Comparative Programming Languages course at Carnegie-Mellon this spring. The course is required for an Applied Math/Computer Science degree, and is usually taken in the junior year, although some sophomores and seniors and a few graduate students from other departments are also enrolled.
>
> The course is intended to give students a background in programming languages in general, not just specific languages. We consider issues such as data abstraction, exception handling, and object-oriented programming. To give students practical experience, though, it is necessary to focus on a few languages in which students can write and run programs. Over the past two or three years, these languages had been Lisp, SNOBOL4, and APL. This semester, for the first time, Icon was substituted for SNOBOL4.
>
> We used the VAX/VMS Icon translator, with all students working on a single VAX. The class consisted of 55 students. To limit the system load to manageable levels, I utilized staggered due dates: half the class had an assignment due on Tuesday, the other half on Thursday.
>
> In the one or two days preceding a due date, the class might have been responsible for more than half the load on the Vax. Degradation of response time was noticeable, though not severe. A more serious problem was the Computation Center policy of limiting the number of simultaneous users; some students had to wait at a terminal more than an hour before the system would allow them to log on. Sometimes this policy seemed to be more restrictive than necessary, but had it not been in effect, degradation of response might have been more pronounced.
>
> We encountered only one real problem with the VMS translator; the default memory allocation for the heap was too small and usually had to be increased manually. We received a fix for this problem very quickly, but it raised another, somewhat less serious, problem.
>
> The students seemed to find Icon no more difficult to learn than Lisp or APL. Some found it the easiest of the three, usually because it is more Pascal-like than Lisp or APL, both of which tend to be more functional and less imperative. Most seemed to find it just about as difficult to debug as Lisp or APL; however, a significant minority felt that the lack of static typing, coupled with a very large number of operators, made it easy to write programs which translate successfully but then encounter run-time errors far away from the incorrect code. No one at all complained about the Icon textbook. I found most of the problems in the book to be too easy to assign as homework, though they are good exercises for self-study.
>
> A few students also knew SNOBOL4; all of them preferred Icon because it's more "modern" and has other interesting features besides pattern matching. (I'm not sure to what extent they "learned" this opinion from me.) I did find it more difficult to teach pattern-matching in Icon, however, and I produced the following table to clear up some of the confusion:

	Generator	Beginning or End?	Matching Function?
find	yes	beginning	no
match	no	end	no
upto	yes	beginning	no
any	no	end	no
many	no	end	no
bal	yes	beginning	no
move	no	end	yes
tab	no	end	yes

> "Beginning or end" refers to whether the function moves the "cursor" to the beginning or end of the string or cset it has found.
>
> In summary, my experience with Icon has been very favorable. I highly recommend it for a comparative languages course, providing sufficient computing resources are available.

6. Portability of Version 5

To understand why there are not yet more implementations of Version 5 of Icon, it is necessary to appreciate that portability was not a major design goal. The initial implementation of Version 5 was designed to run under UNIX on the PDP-11. Most of the implementation is written in C, but there are substantial components written in assembly language and the UNIX support environment is relied upon heavily.

To get an idea of what is involved, consider the Icon interpreter (the compiler is similar). It has three major components:

- A translator that converts source-language programs into an intermediate form, which is called ucode.
- A linker that combines ucode files and produces an interpretable form, which is called icode.
- A run-time system that contains an icode interpreter, routines for built-in operators and functions, a support library, and so forth.

The translator is written entirely in C and is machine independent. The linker is written entirely in C, but has parameters that depend on machine architecture. Most of the run-time system is written in C, but it also contains a substantial amount of assembly-language code (about 700 lines on the VAX-11 implementation). There are also portions of the C code in the run-time system that depend on the target machine architecture.

The main problem in implementing Version 5 of Icon on a new computer lies in the assembly-language code. Some parts, like the icode interpreter, are relatively straightforward. Code related to expression evaluation, which cannot be written in C in a reasonable way, is not straightforward. In particular, it requires an understanding of the C stack frames for the target computer and the design of Icon stack frames to interface them. (The C compiler for the target machine has to have certain properties to make this possible; this can be a stumbling block in itself.)

As mentioned above, the UNIX environment also is important. Adapting Version 5 to other operating systems may involve reorganization of the file hierarchy of the Icon system, provision of alternatives to the UNIX tools that are used to build Icon, modification of the Icon input/output system, and provision of substitutes for UNIX system routines that are used by Icon.

On the positive side, the PDP-11 and VAX-11 assembly-language portions of Version 5 are available as models. The VAX-11 assembly-language code is extensively commented. There is also a detailed guide to transporting Icon, as well as a suite of test programs.

Source material and documentation for Version 5 of Icon is in the public domain. Anyone interested in attempting to transport Icon to a new computer can get this material, free of charge. See Icon Newsletter #12 for ordering information.

7. Programming Corner

Old Business

Assignment to Subscripted Strings: In the last Newsletter, the semantics of expressions such as

```
x[i:j] := (x := expr)
```

were posed, where x is string-valued when the subscripting expression is evaluated, but in which *expr* changes the value of x before the (left) assignment is made to replace the subscripted string. (Expressions such as x[i] and x[i+:j] are just special cases of x[i:j].)

While expressions like this are uncommon (and generally are considered to be in poor style), they are legal and therefore must be well defined and handled properly in the implementation. (It should be no surprise that all the possibilities were not considered in the initial design and that there were several bugs related to these matters in the early versions of the implementation.)

This is a case where efficiency and implementation considerations influenced language design. The problem is that the translator cannot, *in general*, determine whether an expression such as x[i:j] will have a value assigned to it. Even if x[i:j] is the target of an assignment operation, the assignment never may be made because of failure in evaluation elsewhere in the expression. In the case of

```
return x[i:j]
```

the translator has even less information, since the use of the returned expression depends on the context in which the function containing this return is called. For these reasons, the translator treats all expressions such as x[i:j] in the same way*. When an expression like x[i:j] is evaluated, if the value of x[i:j] is a string, a *trapped variable* is produced. A trapped variable is a special kind of variable that points to a small block of data which contains enough information to assign a new value to x if an assignment is made to x[i:j]. This information consists of the variable x and the location of the substring in x. Every string subscripting expression produces a trapped variable. Although the block of data that is created usually is used only transiently, it causes a certain amount of storage throughput.

Now consider what happens if the value of x is changed before an assignment is made to x[i:j]. Since the value of x can be changed to anything, the assignment cannot be made blindly — the position of the replaced string might be out of range, even if the new value of x is a string. Consequently, the type of x is checked when assignment is about to be made to x[i:j]. If the value of x is a string, its length is checked to be sure the substring specified by i:j is still in range. If it is, the assignment is made, even if the value of x is different from what it was when x[i:j] was evaluated. Thus, in

```
x := "hello world"
x[3] := (x := "abc")
```

the value of x becomes "ababc". On the other hand, if the value of x is a string, but it is too short, run-time error 205 (value out of range) occurs, as in

```
x := "hello world"
x[3] := (x := "ab")
```

One might well argue that assignment to x[i:j] should be an error if the value of x has changed, even if the substring is still in range. After all, such a situation seems more likely to be an error than an intentional computation. Here, however, there is an efficiency consideration. In order to be able to detect that the value of x has changed, it would be necessary to save the value of x as well as the variable x in the trapped variable. Furthermore, this would have to be done for every evaluation of a string subscripting expression. The result would be substantially higher storage throughput just to treat a pathological case more elegantly.

From a language design viewpoint, a somewhat more radical alternative would be to bind the value of x to x at the time x[i:j] is evaluated, so that

```
x := "hello world"
x[3] := (x := "abc")
```

would change the value of x to "heabclo world". This solution also would require saving the value of x in the trapped variable — additional overhead that again does not seem justified for such a pathological situation.

Returning to the situation as it actually is handled, given that any string value for x that is long enough is acceptable, the next question is what to do if the value of x is not a string when the assignment is made to x[i:j]? In consonance with Icon's general philosophy of converting types automatically whenever possible, if the value of x can be converted to a string, it is. Thus,

```
x := "hello world"
x[3] := (x := 397)
```

changes the value of x to "39397". Weird, maybe, but consistent with the result of concatenating two integers — which is, after all, what this expression amounts to.

If the value of x cannot be converted to a string, a run-time error (103) occurs, as in

*There is the potential here for an implementation optimization, since there are many situations in which the translator could determine that a subscripting expression is not the target of an assignment. This would require a substantial modification to the implementation, however.

```
x := "hello world"
x[3] := (x := [1, 2, 3, 4])
```

Note that these problems are essentially problems of dereferencing — when and how the value of x is determined when assignment is made to x[i:j]. There are a number of other situations in Icon in which dereferencing is a problem. One is string scanning, which will be discussed in the next Newsletter, along with more material on matching expressions.

Trivia Corner: In the last Newsletter, the following problem was posed:

> What is the longest string of distinct prefix operators which, when applied to a value, might compute a meaningful result? (You may assume any value that you wish.) What if the prefix operators need not be distinct?

For distinct prefix operators, one possibility is

```
|+=-?*~\@^!x
```

It might go like this: Let x be a list of co-expressions. Generate one, refresh and activate it, being sure the result is nonnull. Assuming the result is a cset, use the size of its complement to provide a range for a randomly selected integer. Negate this integer. Match the equivalent string in &subject and convert the result back to an integer. Repeat the whole process (whatever that means). *Enough!*

Strictly speaking, repeated alternation is a control structure, not an operator, but it is denoted with operator syntax. Note that the prefix operators . and / are not included in the expression above. They can be added, but not in a "meaningful" way.

If the prefix operators do not have to be distinct, there is no specific limit on the number that can occur. Consider, for example,

```
=== ... ==x
```

The expression =x matches x in &subject, ==x matches two consecutive occurrences of x, and so on.

What about expressions such as

```
??? ... ??x
```

New Business

Pitfalls: Steve Wampler contributes the following program, in which the procedure tally echos its argument and tallies it in the table count. In the main procedure, empty input lines are converted into the more prominent marker <empty line>. Or are they? What does this program actually do? What does it take to fix the problem?

```
global count

procedure main()
   count := table(0)
   while line := read() do
      tally(("" ~== line) | "<empty line>")
                :
                :
end

procedure tally(s)
   count[s] +:= 1
   write(s)
end
```

8. Electronic Mail

As mentioned in Icon Newsletter #12, there is an electronic mail group for the discussion of topics related to Icon. This group is available to persons who have access to CSNET. To enroll, mail to

icon-group-request.arizona@csnet-relay

Persons who have questions about Icon also can send electronic mail via CSNET to

icon-project.arizona@csnet-relay

or via Usenet or Uucpnet to

arizona!icon-project

We currently have connections established with noao, mcnc, ihnp4, and utah-cs. Note that noao was previously kpno.

9. The Icon Mailing List

Several persons have inquired about the make-up of the Icon community. This, of course, is very difficult to determine, since we have no way of knowing who is interested in Icon, other than through the Icon mailing list. A superficial analysis of this list, for what it is worth, shows the following approximate breakdown by organization:

52%	academic
31%	business/industry
2%	government
15%	unknown

The business/industry category is divided approximately evenly into organizations that are solely involved in computing and other types of organizations.

10. New Documents

There are several new documents related to Icon. All are available, on request, free of charge. Use the document request form at the end of this Newsletter.

The paper "Implementing SNOBOL4 Pattern Matching in Icon" contains a description of how Icon string scanning can be used to implement higher-level pattern-matching procedures. Much of the material in this paper appeared in earlier technical reports; the paper brings this material together in a condensed and refined form.

The technical report *Expression Evaluation and Result Sequences* explains the motivation for generators and goal-directed evaluation in Icon and describes how expression evaluation can be viewed in terms of sequences of results. There is nothing new in this report except its perspective. It may be of interest to persons who are not familiar with expression evaluation in Icon or to persons who are interested in this aspect of programming language design.

Technical Report TR 83-19, *The Construction of Variant Translators for Icon*, describes a system for building source-to-source translators for languages that are syntactically close to Icon. It is particularly useful for producing preprocessors and has been used as a tool for earlier work in list scanning, the cinematic display of pattern matching, and the implementation of Seque (see Icon Newsletter #14). This report may be of interest to persons who want to know more about the Icon translator or to persons with a general interest in software development tools.

TR 83-20, *The Implementation of an Experimental Language for Manipulating Sequences*, describes how Seque is implemented. It contains the details of a variant translator (see above) and a run-time system that is implemented in Icon and makes extensive use of co-expressions. Program listings are included. Persons who are interested in the use of co-expressions may find this material useful, although the programming techniques are somewhat arcane.

TR 84-5, *Diagramming Icon Data Structures*, describes an Icon program that produces diagrams of Icon's internal data structures. The program uses external functions from the Icon program library to access memory directly from the running program. Diagrams of a number of Icon internal data structures are included. This report may be of interest to persons who want to know more about the internal representation of data in Icon. It also contains listings of some moderately large Icon programs.

TR 84-8, *An Icon Subsystem for UNIX Emacs*, describes the design and implementation of Icon as an embedded language for the Emacs editor. An appendix describes the modifications that were made to Icon for this application. This report may be of interest to persons who are interested in programmable editors, language interface issues, or Icon internals.

Finally, the *Icon Address List*, the mailing list for this Newsletter, is available.

Request for Icon Documents

Please send the documents checked below to:

_____

_____

_____

_____

_____

☐ "Implementing SNOBOL4 Pattern Matching in Icon", reprinted from *Computer Languages*.

☐ *Expression Evaluation and Result Sequences*.

☐ *The Construction of Variant Translators for Icon*, TR 83-19.

☐ *The Implementation of an Experimental Language for Manipulating Sequences*, TR 83-20.

☐ *Diagramming Icon Data Structures*, TR 84-5.

☐ *An Icon Subsystem for UNIX Emacs*, TR 84-8.

☐ *Icon Address List*.

☐ Please add my name to the Icon mailing list.

Return this form to:

Icon Project
Department of Computer Science
The University of Arizona
Tucson, Arizona 85721
U.S.A.