THE UNIVERSITY OF ARIZONA
TUCSON, ARIZONA 85721

DEPARTMENT OF COMPUTER SCIENCE

Icon Newsletter #16

Ralph E. Griswold

November 12, 1984

1. Implementations of Icon

Version 5.9 of UNIX Icon

Version 5.9 of UNIX* Icon for the PDP-11 and VAX-11 is now available. This version contains corrections for a number of minor bugs and includes a number of improvements to the implementation, most notably more efficient table references. Several new features are included as optional extensions. The most significant of these are a set data type and set operations.

With this version, the Icon compiler has been eliminated in favor of a "personalized interpreter" system that allows an individual to easily build a version of Icon with modifications to the run-time system (such as the addition of new built-in functions, which formerly was feasible only with the compiler).

The Version 5.9 distribution includes source code for the Icon system, a revised version of the Icon program library, a battery of test programs, and support material for porting Icon to other computers. Documentation is included as part of the distribution package. Use the form at the end of this Newsletter to request Version 5.9.

Other Implementation News

Version 5.8 of Icon for Eunice, implemented by Mark Langley of Science Applications Inc., is now available. There is a form at the end of this Newsletter for requesting this system.

Bob Goldberg has corrected the few known bugs in Version 5.8 of Icon for VMS. We have a copy of this revision and will start distributing it as soon as we finish integrating it with our previous VMS distribution package.

Owen Fonorow at AT&T Bell Laboratories has his implementation of Version 5.9 for the AT&T 3B20 running. He now plans to port Icon to the smaller AT&T 3B5 and 3B2. Persons interested in 3B implementations may contact him as follows:

Mr. O. Rick Fonorow
AT&T Bell Laboratories
1100 East Warrensville Road
IW 2X-361
Naperville, IL 60655

(312) 979-7173

... ihnp4!ihu1e!orf

*UNIX is a trademark of AT&T Bell Laboratories.

Here at the University of Arizona, Rob McConeghy currently is working on an implementation of Version 5.9 for the IBM PC under PC/IX. The initial stages of the coding are done and there are no apparent obstacles to the completion of this project. There is currently no scheduled completion date, but early in 1985 is a reasonable guess. Note that this implementation is for PC/IX, not PC-DOS.

We also expect to undertake an implementation of Version 5.9 for the SUN workstation at the University of Arizona. This project is waiting on delivery of SUNs that presently are on order.

2. Bug in Version 5 of Icon

A bug in the function any(c, s, i, j) recently has been discovered. The function fails to check the case in which i is equal to j (the empty substring), and succeeds if the character past s[i] is in c. This occurs even if i is past the end of s and applies to string scanning as well. For example, even

```
"" ? any(c)
```

may succeed if the character in memory following the empty subject string is in c. In particular,

```
"" ? any(&cset)
```

always succeeds.

At first sight, it may seem surprising that this bug was not discovered earlier. In practice, however, it is unusual to apply any to an empty string — and even if this is done, the chances of a character in c being next in memory are small.

This bug exists in all implementations of Version 5 of Icon. It will be corrected in Version 5.9 as distributed after the date of this Newsletter. Persons who have prior implementations should make a note of this bug and program around it as appropriate.

3. Record Field References

Randal Schwartz of Sequent Computer Systems, Inc. notes that the Icon book does not mention that field names need not be unique. In fact, different records can have the same field names.

This is useful in cases where several different record types have the same attributes. The following declarations suggest such a situation:

```
record noun(word, count)
record verb(word, count)
record adjective(word, count)
```

It is not even necessary for common fields to be in the same positions, as is illustrated by

```
record noun(word, count)
record wlist(count, nextw)
```

Note that field names are bound during program execution. The meaning of

```
x.y
```

depends on the value that x has when this expression is evaluated.

4. Comments on Teaching Icon and the Icon Book

Robert Goldberg, at the Illinois Institute of Technology, sends along the following comments on his use of Icon in a programming languages course:

> Icon has been warmly received by my "Programming Languages and Translators I" class. The reasons are:
>
> (1) A good book. The Icon book does double duty as a text and a reference manual. Also, it can be easily *read*! These attributes make it stand apart from many of the other books on programming languages. (The Modula-2 book by Wirth is another of our texts and is universally hated by the students.)
>
> I have a few quibbles with the book — the exercises are not difficult enough and the description of parameter

passing is inadequate for structured data types.

However, these quibbles are minor compared to its overall readability and wonderful sample programs.

(2) An interesting language. Most students are familiar with strongly typed languages like Pascal and Modula. Icon, however, provides them a perspective on "typeless" languages. The power of Icon amazes them and makes them curious about its implementation. Thus, Icon makes me, the lecturer, happy, because (first) we can assign suitably interesting machine problems without becoming a data structures course and (second) it provides a natural lead-in to topics like dynamic storage allocation, garbage collection, run-time representations of data objects, etc.

There is also a third reason: many of our students work and have access to computers off campus. The availability of Icon at little or no cost for most of the major VAX operating systems means that students can easily obtain it for their work computer with no "red tape".

5. Rebus

Much of the usefulness of SNOBOL4 lies in its pattern matching facilities. String scanning in Icon is designed to provide similar power, and unlike SNOBOL4, it allows the use of all language operations during high-level string analysis. Icon's string scanning facility, however, focuses on the computation processes involved in string analysis and does not provide the high-level characterization of string structure that SNOBOL4 patterns do.

On the other hand, SNOBOL4 is an old language. Many of its features seem awkward when compared with those of Icon, and its lack of control structures makes even simple programming tasks painful and often results in poorly structured programs.

A new programming language, called Rebus, has been designed as an experiment in imbedding the pattern matching of SNOBOL4 in a syntax that is similar to Icon's. Rebus is implemented by a preprocessor that produces SNOBOL4 code.

A technical report describing Rebus is available; use the document request form at the end of this Newsletter.

6. Programming Corner

Old Business

In the last Newsletter, Steve Wampler contributed the following problem:

> 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
```

If line is not empty, tally(line) is called and line is written. However, since tally has no explicit return, it fails by flowing off the end of the procedure body. Since the call fails, the argument

```
("" ~=== line) | "<empty line>")
```

is resumed, resulting in the call tally("<empty line>"). Consequently, <empty line> is written after every nonempty line, as well as in place of empty lines.

The cure is simple — insert a return at the end of the procedure body for tally. (What is another way of fixing the problem?) The lesson is a more general one — be careful to provide a return at the end of a procedure body unless calls of the procedure are supposed to fail.

Chosing Programming Techniques in Icon

There often are several ways of doing the same thing in Icon. While this is true of most programming languages, it is exaggerated in Icon, since its expression evaluation mechanism is more general than the expression evaluation mechanisms of "Algol-like" languages, such as Pascal and C. Thus, in Icon, there is often an Algol-like solution and also a solution that makes use of generators. The fact that Icon has both low-level string operations and higher-level string scanning complicates the situation.

The novice Icon programmer (and even the more advanced one) is faced with choices that may be confusing or even bewildering. Most programmers develop a fixed set of techniques and often fail to use the full potential of the language.

In the discussion that follows, a simple text processing problem is approached from a variety of ways to illustrate and compare different programming techniques in Icon. The problem is to count the number of times the string s occurs in the file f, which is formulated in terms of a procedure scount(s, f).

The first attempt at such a procedure might be

```
# scount1

procedure scount(s, f)
   count := 0
   while line := read(f) do
      while i := find(s, line) do {
         count +:= 1
         line := line[i + 1:0]
         }
   return count
end
```

This solution is very Algol-like in nature and explicitly examines successive portions of each input line. It does not take advantage of the third argument of find, which allows the starting position for the examination to be specified. Using this feature, the procedure becomes

```
# scount2

procedure scount(s, f)
   count := 0
   while line := read(f) do {
      i := 1
      while i := find(s, line, i) + 1 do
         count +:= 1
      }
   return count
end
```

While this solution is shorter than the previous one, it is still Algol-like and uses only low-level string processing. Using string scanning, an alternative solution is

```
# scount3

procedure scount(s, f)
   count := 0
   while line := read(f) do
      line ? while tab(find(s) + 1) do
         count +:= 1
   return count
end
```

This approach eliminates the auxiliary identifier i and uses scanning to move through the string. None of the solutions above uses the capacity of find to *generate* the positions of successive instances of s, however. If this capacity is used, it is not necessary to tab through the string, and the following solution will do:

```
# scount4

procedure scount(s, f)
   count := 0
   while line := read(f) do
      line ? every find(s) do
         count +:= 1
   return count
end
```

At this point, it becomes clear that string scanning provides no advantage and it can be eliminated in favor of the following solution:

```
# scount5

procedure scount(s, f)
   count := 0
   while line := read(f) do
      every find(s, line) do
         count +:= 1
   return count
end
```

Finally, the hard-core Icon programmer may want to get rid of the while loop, making use of the fact that the expression !f generates the input lines from f. The solution then becomes

```
# scount6

procedure scount(s, f)
   count := 0
   every find(s, !f) do
      count +:= 1
   return count
end
```

The relative merits of these different solutions are arguable on stylistic grounds. Certainly the last (scount6) is the most concise, but scount5 is probably easier to understand.

But what about efficiency. Is scount6 more efficient than scount5? In fact, how much difference in performance is there among all the solutions?

The relative efficiency varies considerably, depending on the data — how many lines there are in f, how long the lines are, how many times s occurs in f, and so forth. The following figures are typical, however. The figures are normalized so that the fastest solution has the value 1.

scount1	2.96
scount2	2.14
scount3	2.03
scount4	1.14
scount5	1.04
scount6	1.00

The fact that the last solution is the fastest may not be surprising — it is the shortest and uses the features of Icon that are most effective in internalizing computation. Nor should it be surprising that scount2 is significantly faster than scount1, since scount2 avoids the formation of substrings. (It is worth noting, however, that substring formation is relatively efficient in Icon — no new strings are constructed, only pointers to portions of old ones.)

It might be surprising, however, to discover that the use of string scanning in scount3 provides a significant advantage over scount2. Evidently, the internalization of the string and position that scanning provides more than overcomes the fact that scount3 produces substrings (by tab).

The real gain in efficiency comes with the use of generators in scount4, where the state of the computation is maintained in find for all the positions in any one line.

Getting rid of string scanning, which serves no useful purpose in scount5, produces an expected improvement, although perhaps not as much as might be expected. The last step of reducing the nested loops to a single loop in scount6 also produces a slight improvement in performance.

What might be learned from these examples is that there may be a very substantial difference in performance in Icon, depending on the technique used — a factor of nearly 3 between the naive solution of scount1 and the sophisticated one of scount6. The value of using the capabilities of generators is also evident, both in performance and in the conciseness of the solutions. It is notable that string scanning is not as expensive as one might imagine. It can be used without the fear that it will degrade performance substantially.

Different Ways of Looking at Things

Steve Wampler contributes the following interesting note on programming in Icon:

> How do you test to see if the value of i is not between 1 and the length of the string s?
>
> I would write:
>
> ```
> if not (1 <= i <= *s) then ...
> ```
>
> but my students wrote:
>
> ```
> if *s < i < 1 then ...
> ```

7. Electronic Mail

As mentioned in previous Icon Newsletters, 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.

8. New Documents

There are several new documents related to Icon. All are available, on request, free of charge. Use the document request form that follows.

Request for Icon Documents

Please send the documents checked below to:

_____

_____

_____

_____

_____

☐ "Expression Evaluation in the Icon Programming Language", reprinted from the proceedings of the *1984 ACM Conference on LISP and Functional Programming.*

☐ *Rebus — a SNOBOL4/Icon Hybrid*, TR 84-9.

☐ *Extensions to Version 5 of the Icon Programming Language*, TR 84-10.

☐ *Personalized Interpreters for Icon*, TR 84-14.

☐ *Tables in Icon*, TR 84-16.

☐ 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.

Version 5.9 UNIX Icon Distribution Request

Note: This system can be configured for PDP-11s with separate I and D spaces or VAX-11s.

Contact Information:

name _____

address _____

_____

_____

_____

telephone _____

electronic mail address _____

computer _____

operating system _____

All tapes are written in 9-track *tar* format. Specify the preferred tape recording density:

☐ 1600 bpi ☐ 800 bpi

Return this form to:

Icon Project
Department of Computer Science
The University of Arizona
Tucson, AZ 85721

Enclose a magnetic tape (at least 600') *or* a check for $15 payable to The University of Arizona.

Version 5.8 Eunice Icon Distribution Request

Contact Information:

name _____

address _____

_____

_____

_____

telephone _____

electronic mail address _____

computer _____

operating system _____

All tapes are written in 9-track *tar* format. Specify the preferred tape recording density:

☐ 1600 bpi ☐ 800 bpi

Return this form to:

Icon Project
Department of Computer Science
The University of Arizona
Tucson, AZ 85721

Enclose a magnetic tape (at least 600') *or* a check for $15 payable to The University of Arizona.