THE UNIVERSITY OF ARIZONA

TUCSON, ARIZONA 85721

DEPARTMENT OF COMPUTER SCIENCE

Icon Newsletter #18

Madge T. Griswold and Ralph E. Griswold

April 23, 1985

1. Implementation News

Icon for the IBM PC under PC/IX

An implementation of Version 5.9 of Icon for the IBM PC operating under PC/IX has been completed at The University of Arizona by Robert K. McConeghy[1].

At the present time we are distributing executable binaries with the Icon program library as an option, but we are not distributing the source code. Distribution is on DSDD 5¼″ diskettes in PC/IX *dump/restore* format only. Use the request form at the end of the Newsletter. Note that this implementation of Icon will *not* run on other PC operating systems, such as DOS.

Icon for AT&T 3B2/3B5 and for UNIX/PC

O. Rick Fonorow, of AT&T Information Systems Laboratories, reports that his preliminary Version 5.9 implementations of Icon for the 3B2/3B5 are running. Porting to the UNIX/PC[2] is under way.

For more information, contact:

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

(312) 979-7173

...ihnp4!iwsam!orf

Corrected Request Form for Version 5.9 VAX/VMS Icon

In the preceding Newsletter, the form for requesting Version 5.9 of Icon for the VAX/VMS incorrectly specified that distribution was in *tar* format. The actual format is *COPY*. A corrected request form is included in this Newsletter.

[1] This work was supported in part by equipment and software provided by IBM Corporation.

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

Another Implementation Project

A project to implement Icon is under way at Illinois Institute of Technology. The project is currently being conducted solely for research and education — it is uncertain whether a distribution version of the system will ever be released.

The compiler was designed and implemented by Thomas Christopher. It was inspired by a desire to try out a new implementation of backtracking. The compiler is written in Icon and is generating VAX/UNIX assembly code. Phil Hatcher is supervising seven undergraduate students who are implementing the run-time system.

Future work includes writing an optimizing compiler for Icon (again based on designs by Christopher), translation of the compilers into C, and some experimentation with different implementations of the run-time system.

2. An Icon Machine?

We recently received a proposal for an Icon "machine" that would consist of software between the UNIX kernel and user applications. Perhaps we will have something for publication on this idea in a subsequent Newsletter, but for now, we offer the following response to the proposal by Dave Gudeman at the University of Arizona:

> The Icon machine is obviously an idea whose time has come. Of course the proposal is only for software, but it's only a matter of time .... I have a few of my own ideas on the Icon machine, specifically involving terminology, which is seldom fully thought out. First of all, what is this Icon shell supposed to be called? The first thing to come to mind is, of course, the I-shell. This is very unentertaining and dry. I prefer, the Iconosphere. This word has implications which lend wings to the imagination.
>
> Of course, once you have an Iconosphere you need the positive and negative icons -- applications which are or are not considered suitable for doing in Icon. And the field of applications brings up the subject of standards. I propose a rigorous set of standards against which all Icon applications should be judged. Those which follow the standards would be referred to as "Iconoclasts".
>
> People who write the Icon software should be called software Generators. Those who are especially skilled would work on many different projects in alternation and should have the special title "Alternator". This leaves the title "Transformer" for programmers who specialize in taking C software and applying it to the Iconosphere (not to mention as a new Icon operator). Someone has to do the testing. If each programmer checked each other programmer's work, this would be referred as mutual evaluation. Another option would be to have a special team of "Scanners" to look at each program as it is finished: Each program going first to one scanner team, if they couldn't find any problem, the "second string" (as it were) would be brought in, if they couldn't find anything, the third string of scanners would try, etc. This process would be referred to as serial evaluation. (Got you, didn't I?)
>
> When a real Icon machine comes along, a CPU that supports Icon should be called an edifice. This would have various new hardware devices to speed up Icon primitives (icon-accelerators) as well as supporting Icon stack operations and data structures directly in hardware. It would probably not be classified as a RISC machine. There would be various competing but similar architectures which would be referred to as Icomers of each other.

3. Programming Corner

In the last Newsletter the following problem was proposed:

> Write a procedure `anagram(s)` that produces a string consisting of the characters in `s` in alphabetical order, but without the deletion of duplicates.

We received a number of interesting solutions from several persons. The most efficient solution, at least when working on a large amount of data, follows:

```
procedure anagram(s)
   local c, s1
   s1 := ""                              # start with the empty string
   every c := !cset(s) do                # for every character in s
      every find(c, s) do                # and every time it occurs in s
         s1 ||:= c                       # append one
   return s1
end
```

The heart of this solution is to use cset(s) to obtain an ordered set of the characters in s. It is interesting that it is noticeably more efficient to use find instead of upto in the solution. The difference does not lie primarily in the functions themselves. What else is at work here?

It is also faster to append the characters one at a time than to count the number of each and append them in groups, as in

```
procedure anagram(s)
   local c, i, s1
   s1 := ""                              # start with the empty string
   every c := !cset(s) do {              # for every character in s
      i := 0
      every(find(c, s)) do               # count the number of times it occurs
         i +:= 1
      s1 ||:= repl(c, i)                 # and append that many copies to the result
      }
   return s1
end
```

Why should this be?

Randal Schwartz submitted a number of interesting solutions in addition to one similar to the first solution above. One of his solutions uses a table of characters and their counts:

```
procedure anagram(s)
   local c, t
   c := table(0)
   s ?:= {
      while c[move(1)] +:= 1
      ""
      }
   every t := !sort(c, 1) do
      s ||:= repl(t[1], t[2])
   return s
end
```

This solution is about twice as slow, when working on a large amount of data, as the first one above, probably because of time spent in table lookup and garbage collection (due to the larger amount of storage needed for tables).

An even more interesting, albeit admittedly inefficient, solution from Randal Schwartz is the following one that puts each character of s in a separate table element:

```
procedure anagram(s)
   local t
   t := table()
   s ?:= {
      while t[[]] := move(1);
      ""
      }
   every s ||:= (!sort(t, 2))[2]
   return s
end
```

Do you see why there is a separate table element for each character?

4. New Documents

Several technical reports related to Icon have been published recently:

Technical report TR 84-21 describes a system for observing, interactively, the operation of the Icon interpreter. This system uses a two-process model in which one Icon program (the observer) controls the execution of another Icon program. Facilities include stack diagrams, a cinematic display of the execution of the code for the Icon virtual machine, and access to the memory of the observed program.

Technical reports TR 85-2 and TR 85-4 describe Seque, an experimental programming language for manipulating streams. Streams have both a storage and a computational component and are "first-class" data objects. The computational component of streams utilizes Icon expressions and particularly generators to characterize sequences of values. Seque also provides notational mechanisms for the concise representation of complicated streams. Streams can be nonhomogeneous, infinite, and self-referential.

The last technical report, TR 85-8, describes in some detail the representation of data objects in Version 5 of Icon. Descriptions of the methods for manipulating lists, sets, and tables are included.

All of these reports are available, free of charge. Use the document request form that follows.

Request for Version 5.9 of Icon for VAX/VMS

Contact Information:

name	______________________
address	______________________
	______________________
	______________________
	______________________
telephone	______________________
electronic mail address	______________________
computer	______________________
operating system	______________________

All tapes are written in 9-track COPY format. Specify 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 600′ magnetic tape *or* a check for $20 payable to The University of Arizona.

Request for Version 5.9 of Icon for PC/IX

Note: This implementation of Icon runs on IBM PCs under PC/IX, *not* under PC-DOS.

This system is distributed on 5¼″ DSDD diskettes in PC/IX *dump/restore* format. Charges include media, documentation, handling, and shipping. Make checks payable to The University of Arizona.

☐	Executable binaries and test programs (one diskette):	**$25**
☐	Icon program library (two diskettes):	**$40**

Total enclosed: ____________

Ship to:

name	______________________________
address	______________________________
	______________________________
	______________________________
	______________________________
telephone	______________________________
electronic mail address	______________________________

Return this form with payment to:

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

Request for Icon Documents

Please send the documents checked below to:

____

____

____

____

____

☐ *A Tool for Interactive Observation of the Icon Interpreter*, TR 84-21.

☐ *Seque: A Language for Programming with Streams*, TR 85-2.

☐ *Reference Manual for the Seque Programming Language*, TR 85-4.

☐ *The Implementation of Data Structures in Version 5 of Icon*, TR 85-8.

☐ Please add my name to the Icon mailing list.

Return this form to:

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