THE UNIVERSITY OF ARIZONA
TUCSON, ARIZONA 85721

DEPARTMENT OF COMPUTER SCIENCE

ICON NEWSLETTER #4

Ralph E. Griswold and David R. Hanson

June 3, 1980

Icon for UNIX

An implementation of Version 3 of Icon, written in C to run under UNIX* on the PDP-11, is now available. This implementation is significantly faster on equivalent machines than the portable (Ratfor) implementation and contains a number of new features (see the following section for features of Version 3).

The C implementation requires Version 7 of UNIX and separate I and D spaces to provide adequate memory. Thus it is limited to the PDP 11/44, 11/45, and 11/70. (Persons interested in adapting this implementation to other configurations should write to us.)

Version 3 of Icon is distributed in *tar* format on 9-track magnetic tape. A form for ordering this system is attached to the end of this Newsletter. There is no charge for this system if a magnetic tape is sent with the request. Alternatively, we will supply a tape for a charge of $15.00. See the order form for details.

Features of Version 3

Version 3 of Icon (which is available only in the C implementation, not the Ratfor implementation) contains a number of changes and new features. A number of the changes are "cosmetic", although some of these, such as an improved notation for specifying substrings, improve program readability considerably.

Among the more significant changes are:

- **case** selectors may be arbitrary expressions, not just literals.
- The **scan** expression returns the result of its **using** clause, rather than the value of &subject.
- A **transform-using** expression has been added. Its semantics are the same as those of **scan-using**, except that in **transform** e1 **using** e2, e1 must be a variable and the value of &subject upon completion of the using **using** clause is assigned to e1.
- insert(s,i) inserts the string s in &subject at position i.

There are also a number of new features, such as the ability to call a shell, that allow Version 3 programs to function comfortably in the UNIX environment.

*UNIX is a trademark of Bell Laboratories.

Future Directions

Our future effort on the portable (Ratfor) implementation of Icon will be limited to the distribution and maintenance of Version 2. We have no plans to update the portable implementation to Version 3.

The C implementation is active and is the focus of our current work. We expect to incorporate new results in this system.

Our present research efforts are concentrated in two areas:

- control structures, especially new ideas and applications related to generators.
- string scanning, including unification of apparently disparate features and problems related to the scope and lifetime of scanning variables.

Programming Corner

One aspect of Icon that distinguishes it from other languages with goal-directed evaluation (notably AI languages) is the limited scope of backtracking in Icon. In Icon, backtracking is strictly limited by syntactic constructions, as opposed to being determined by the computational history of the program. An advantage of this limitation is that the extent of backtracking can be determined from the program structure and undesired backtracking, one of the banes of goal-directed evaluation, can be avoided easily. There are also significant efficiencies that can be obtained by limiting backtracking.

Unfortunately, the early Icon documentation is not explicit about which syntactic constructions limit backtracking (this omission is corrected in the Version 3 documentation). Briefly stated, backtracking is limited by semicolons separating expressions, braces surrounding expressions, and by all control-structure reserved words except **every-do**. Thus in

```
e1; e2
```

once evaluation of e1 is complete, no failure in e2 can cause backtracking into e1. Similarly, in

```
if e1 then e2 else e3
```

once evaluation of e1 is complete, failure in e2 or e3 cannot cause backtracking into e1.

Within expressions, however, backtracking is unlimited. Thus in

```
e1 || e2
```

if e1 succeeds, but e2 fails, backtracking into e1 occurs. If e1 produces an alternative, e2 is evaluated again.

This mode of evaluation is very general. For example, in

```
f(x | y)
```

f(x) is called first. Should this call fail, backtracking to the argument expression occurs, and f(y) is called.

Note that there is nothing special about the conjunction operator: e1 & e2 is simply an operation that returns its right operand (as opposed to performing some computation).

This evaluation mechanism introduces all kinds of programming possibilities — and also may lead to unexpected results and errors if it is not understood and used properly.

Consider the problem of writing a procedure to compress repeated, successive occurrences of a character to a single instance of that character. There are, or course, many ways of doing this. One method, using string scanning, is illustrated by the following procedure.

```
procedure compress(s,c)
   local x
   scan s using {
      while tab(upto(c)) do {
         x := move(1)
         tab(many(x)) := ""
         }
      return &subject
      }
end
```

Note that c may be a set of characters. (You might speculate on the wisdom and possible consequences of returning from within a **using** clause — this style is used here to be compatible with both Versions 2 and 3.)

One question that you might ask about this solution is why the local identifier x cannot be removed as follows:

```
   scan s using {
      while tab(upto(c)) do {
         tab(many(move(1))) := ""
         }
      return &subject
      }
```

Consider the case in which there is just a single instance of a character in c. In the first solution

```
tab(many(x)) := ""
```

fails and no replacement is performed. This failure is of no consequence, since

```
x := move(1)
```

has already moved &pos past this character. (There is an implicit semicolon after the move expression.) Note that the compound expression in the **do** clause fails, but this again is of no consequence. (What would happen if backtracking was not limited by **while** and backtracking occurred into the control expression tab(upto(c))?)

However, in

```
tab(many(move(1))) := ""
```

the failure of many results in backtracking into move. Although move has no alternatives, it does restore &pos to its previous value. (What are the consequences of this?)

It might seem that the auxiliary identifier and the division of the computation into two expressions are necessary to avoid unwanted backtracking. Not so — consider

```
   scan s using {
      while tab(upto(c)) do {
         tab(many({move(1)})) := ""
         }
      return &subject
      }
```

Here the braces provide an explicit barrier to backtracking, preventing &pos from being restored in case many fails.

Tricky? Perhaps, but the issues raised by goal-directed programming and the limitation of backtracking are worth study.

Whimsy

Everyone seems to expect that the names of programming languages (unless they are also the names of famous mathematicians) are acronyms of some sort.

We assure you that Icon is not an acronym. One can, however, imagine relevant connotations or possible motivations for the choice of the name. Two definitions from *Webster's Third New International Dictionary* are suggestive:

> an object of uncritical devotion
>
> a sign . . . that signifies by virtue of sharing a property with what it represents

We have also been treated to a number of 'aphorisms' such as

> Anything you can do, Icon do better
> Icon do anything better than you

We welcome contributions of this kind for our collection, but offer no promise to publish them.

We have also accumulated a list of English words that contain *icon* as a substring. Examples are *aniconic*, *semiconductor*, and *vicontiel*. In all, we know of 377 such words (and 29 words that contain the substring *noci*). If you are a logophile and would like a copy of the list, it is available (see the section on Publications, which follows). Contributions of words we may have overlooked are also welcome (but no logodaedaly, please).

Publications

A number of new publications related to Icon are now available:

- Corrigenda to the Version 2 Reference Manual — a list of corrections, both typographical and technical, to TR 79-1a.

- Features of Version 3 — a short document summarizing the changes and additions that distinguish Version 3 from Version 2.

- Version 3 Reference Manual — a complete reference manual for Version 3, including examples of Icon programs.

- A Tour through the C Implementation — a description of the implementation of Version 3.

- "An Alternative to the Use of Patterns in String Processing" — a reprint of a paper in the April 1980 issue of *TOPLAS* that contrasts patterns in SNOBOL4 with string scanning in Icon.

- "The Words of Icon" — a list of English words that contain the substrings *icon* and *noci*.

Single copies of these publications are available without charge — use the order form that follows.

Document Request

Please send me the publications checked below:

☐ Corrigenda to the Version 2 Reference Manual

☐ Features of Version 3*

☐ Version 3 Reference Manual*

☐ A Tour through the C Implementation*

☐ "An Alternative to the Use of Patterns in String Processing"

☐ The Words of Icon

*These documents are supplied with copies of the Version 3 system; please do not request them here if you are also requesting the Version 3 system.

Request for Version 3 of Icon

Note: This system is designed to run under Version 7 of UNIX on PDP-11 computers that have separate I and D spaces. The system is distributed in *tar* format on 9-track magnetic tapes only. Tapes supplied by the requester should be at least 400' long. Longer tapes add to shipping costs, but will be accepted. If you prefer to have us supply a tape, enclose payment of $15.00 in the form of a check or prepaid purchase order payable to "The University of Arizona".

Please send me Version 3 of Icon

☐ 800 bpi

☐ 1600 bpi

☐ A magnetic tape is enclosed

☐ Payment for a tape is enclosed

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