THE UNIVERSITY OF ARIZONA

TUCSON, ARIZONA 85721

DEPARTMENT OF COMPUTER SCIENCE

ICON NEWSLETTER #3

Ralph E. Griswold and David R. Hanson

February 22, 1980

Perspective on Icon

The origins of Icon lie in research on programming language facilities for processing strings and structures. This work began at Bell Laboratories with the SNOBOL languages and has continued at the University of Arizona with support from the National Science Foundation. When the capacity of SNOBOL4 as a research tool was exhausted, SNOBOL4 was succeeded by SL5 and then by Icon. While the underlying motivation has always been research, the SNOBOL languages, SL5, and Icon were produced as completely implemented programming languages to provide contexts for testing new ideas, to serve as research tools, and to make new facilities available to the programming community. Each release in the sequence represents a reasonably complete and distinct programming language. To a certain extent, the development of these programming languages has been evolutionary, with inheritance of philosophy, style, and features from earlier versions. There have been "environmental" pressures also: changing concepts of acceptable style, increasing emphasis on structured programming, and so on. As those who have followed the development of these languages know, resistence to fads has been a major philosophical guideline.

Version 2 of Icon

We have recently completed the design and implementation of Version 2 of Icon. The change in version number indicates significant changes in language features (not just implementation changes indicated by decimal suffixes such as 1.3).

Version 2 for the DEC-10, CDC 6000/Cyber, and in portable form is available from us. Order forms are included at the end of this Newsletter. As before, there is no charge for these systems, but magnetic tapes must accompany requests.

Some of the new features of Version 2 have been motivated by our own work on language design, while other features have been added in response to the suggestions and requests of others. While the changes are not really major, some existing programs may have to be changed for Version 2.

A complete description of Version 2 features is included with distribution of the system. Briefly summarized, the new features are:

Some changes have been made to the rules for the scoping of identifiers.

The syntax for structure creation has been changed to a functional notation.

Record field names no longer need be unique.

Assignment may be made to substrings produced during string scanning. The subject is modified as a result, allowing scanning to perform incremental string transformations.

A revised language manual is distributed with the new system. This document can also be requested separately. Ask for TR 79-1a or "the Version 2 Icon Manual".

Feedback

As indicated above, we have received several useful comments on Icon. The most extensive criticism to date is contained in a draft report on PSI ("Possible Simplifications for Icon") by Messrs. P. Isbendjian of the Université Libre de Bruxelles and R. Haentjens of the Koninklijke Militaire School. Their proposals are extensive and thought-provoking. They are presently preparing a report for publication. When it becomes available, we will announce it in a subsequent Newsletter.

Critical comments, especially from persons who have actually used Icon, are very important to our work. We would like to get your comments — and we will respond to all correspondence.

Portable Icon

To date 34 portable Icon systems have been distributed to installers on a wide range of computers. In addition to the DEC-10, CDC Cyber/6000, IBM 370, and FACOM 230 implementations mentioned in the last Newsletter, there is now a running system for the IBM 3031 (done independently of the IBM 370 system), and implementations are running, or nearly so, for the Data General Eclipse and UNIVAC 1108.

As noted above, we distribute the DEC-10 and CDC Cyber/6000 systems. The IBM 370 system (Version 1.2) for OS 21.8 is available for a $5 handling fee from

Mr. William H. Mitchell
University Systems Analysis and Control Center (USACC)
Box 50298
338 Daniels Hall
North Carolina State University
Raleigh, North Carolina 27650

Send a 2400′ tape in a reusable package along with the handling fee. Tapes will be written at 800 bpi with IBM standard labels.

The C Implementation of Icon

Cary Coutant and Steve Wampler of the University of Arizona have a C implementation of Icon running on a PDP-11/70 under UNIX. This implementation is quite efficient — preliminary timing tests indicate it runs 3 to 5 times faster than the portable implementation.

We expect to release this system later this spring. Its availability will be announced in a subsequent Newsletter.

Programming Corner

Experience with Icon has shown that the full use of its facilities requires different ways of thinking about programming than are used in other languages. In the last Newsletter we mentioned some of the problems encountered in attempting to use SNOBOL4-style programming techniques in Icon. With this Newsletter we are starting a "Programming Corner" to illustrate, by examples, some programming techniques that we have found useful in Icon. The first installment follows.

The built-in scanning functions tab(i) and move(i) can be easily coded as Icon procedures. This exercise is helpful both in understanding tab(i) and move(i) better and in illustrating some Icon idioms. Since tab(i) and move(i) differ only in that the former changes &pos absolutely while the latter changes &pos relatively, only tab(i) will be considered here.

A naive procedure for tab(i) is

```
procedure tab(i)
   return section(&subject,&pos,&pos := i)
end
```

Before discussing the defects in this solution, aspects of the returned expression should be noted. In the first place, the value of &pos := i is the value of i. Thus the section of &subject returned is between the old and new values of &pos. Furthermore, if &pos := i fails (because i is out of range of &subject), &pos is not changed and the failure of &pos := i is inherited by section and in turn by **return**. Hence tab(i) fails as it should.

One defect in this procedure is that it does not restore &pos to its former value in case tab(i) succeeds, but the enclosing expression in which it occurs fails. Consider, for example

```
tab(i) & find(s)
```

Suppose tab(i) succeeds (thus changing &pos) but find(s) fails. In this case, the built-in version of tab(i) restores &pos to its previous value. This is one of the few cases in Icon where there is data backtracking (the others are move(i), =s, x <- y, and x <-> y).

From the point of view of expression evaluation, the failure of find(s) in

```
tab(i) & find(s)
```

results in a request for an alternative value for tab(i). While tab(i) itself is not a generator (it can only set &pos to the value of i), the request for an alternative value allows tab(i) to gain control in order to restore &pos to its former value. This can be accomplished in the procedure above by a few minor modifications:

```
procedure tab(i)
   local j
   j := &pos
   suspend section(&subject,&pos,&pos := i)
   &pos := j
   fail
end
```

The local identifier j provides storage for the original value of &pos. **suspend** is used in place of **return** so that the procedure can regain control if an alternative value is requested. When control is regained, &pos is restored to its former value and tab(i) returns with a failure signal.

As a detail, it is worth noting that if section fails because &pos := i fails, the **suspend** does not return to the calling procedure and control falls through to the (unnecessary but correct) expression that follows. That is, while **return** inherits its signal from the success or failure of its argument, **suspend** (in the style of **every**) returns the alternatives of its argument — if its argument fails at once, there are no alternatives to return.

The argument of **suspend** in the procedure above can be made somewhat more compact at the expense of clarity as follows:

```
suspend section(&subject,j := &pos,&pos := i)
```

in which case the immediately preceding expression can be omitted. In addition, the restoration of &pos to its former value can be accomplished by using reversible assignment, thus eliminating the need for j and the explicit save and restore of &pos. Thus the procedure becomes:

```
procedure tab(i)
   suspend section(&subject,&pos,&pos <- i)
   fail
end
```

The **fail** expression is still needed, since **suspend**, as noted above, cannot fail.

In order to have confidence that the value of &pos is restored in the procedure above, it is necessary to appreciate that when the procedure is reactivated for an alternative value, **suspend** requests an alternative value from its argument, which in turn results in a request of an alternative value for &pos <- i. Like tab(i), reversible assignment has no alternative value, but does restore the former value of its left operand when it gains control. It then fails and control falls through to **fail**.

There is one subtle difference between the behavior of the procedure as given above and that of the built-in function. Since section returns a variable to which assignment can be made, it is possible to use the procedure above as follows:

```
tab(i) := s
```

thus replacing the substring of &subject returned by tab(i) and also setting &pos to 1 (which always happens when the value of &subject is changed). On the other hand, the built-in function tab(i) does not allow such an assignment (although a feature of this kind is included in Version 2 of Icon). This extra feature of the procedure above can be removed by explicitly dereferencing the argument of **suspend** so that only a value is returned. One way to do this is by means of explicit type conversion (which actually does nothing but dereference the argument, since its type is already **string**):

```
suspend string(section(&subject,&pos,&pos <- i))
```

As a final note, it should be observed that expressions such as

```
tab(i1 | i2)
```

work properly. This is a direct consequence of the goal-directed evaluation mechanism of Icon, which causes a procedure to be called with alternative arguments if the context in which the procedure call occurs results in requests for alternative values. Thus

```
tab(i1 | i2)
```

is equivalent to

```
tab(i1) | tab(i2)
```

It is important to realize that this aspect of generation is completely independent of the code in the procedure body for tab(i) and is solely a property of goal-directed evaluation.

DEC-10 Icon Distribution Request; Version 2.0

Contact Information

name:	______________________________
address:	______________________________
	______________________________
	______________________________
	______________________________
telephone:	______________________________
cable/telex:	______________________________

Computer Information

model:	______________________________
memory capacity:	______________________________
operating system:	______________________________
comments:	______________________________
	______________________________
	______________________________
	______________________________

Magnetic Tape Information

Icon for the DEC-10 is distibuted as a BACKUP tape in interchange mode. Please specify your preferred tape recording format:

☐ 9-track	☐ 7-track
☐ 1600 bpi	☐ 800 bpi

comments:

Please return this form with a magnetic tape (at least 1200′) to:

Ralph E. Griswold
Department of Computer Science
University Computer Center
The University of Arizona
Tucson, Arizona 85721
USA

CDC 6000/Cyber Icon Distribution Request; Version 2.0

Contact Information

name: ______________________________

address: ______________________________

____

____

____

telephone: ______________________________

cable/telex: ______________________________

Computer Information

model: ______________________________

memory capacity: ______________________________

operating system: ______________________________

character set: ________ ☐ 63 ☐ 64

comments: ______________________________

____

____

____

Magnetic Tape Information

Icon for CDC/Cyber systems is distributed as UPDATE PLs on an unlabeled SCOPE-format tape. Please specify your preferred tape recording charactersitics:

☐ 9-track	☐ 7-track	
☐ 1600 bpi	☐ 800 bpi	☐ 556 bpi

Please return this form with a magnetic tape (at least 1200′) to:

Ralph E. Griswold
Department of Computer Science
University Computer Center
The University of Arizona
Tucson, Arizona 85721
USA

Portable Icon Distribution Request; Version 2.0

Contact Information

name: ________________________________

address: ________________________________

____

____

____

telephone: ________________________________

cable/telex: ________________________________

Computer Information

manufacturer: ________________________________

model: ________________________________

memory capacity: ________________________________

operating system: ________________________________

Fortran pecularities: ________________________________

comments: ________________________________

____

____

____

Magnetic Tape Information

Our standard format for magnetic tape distribution of Icon source material is 9-track, 1600 bpi (phase encoded), EBCDIC industry standard, unlabeled, fixed-block 80-character records with a blocking factor of 10 (last block filled out to 800 characters). Please indicate if you can accept this format:

☐ yes ☐ no

If you cannot accept this format, please indicate acceptable alternatives by checking the boxes below. Circle the checks that correspond to your preferred format. (You may fill out this section even if you can accept our standard distribution format — we will try to accomodate your preferences, although doing so may cause delays.)

☐ 9-track	☐ 7-track	
☐ 1600 bpi	☐ 800 bpi	☐ 556 bpi
☐ EBCDIC	☐ ASCII	

blocking factor:

☐ 1	☐ 10	☐ other (specify)

comments:

☐ check here if you also want the UA Ratfor system

Please return this form with a magnetic tape (at least 1200′ or 2400′ if Ratfor is requested) to:

Ralph E. Griswold
Department of Computer Science
University Computer Center
The University of Arizona
Tucson, Arizona 85721
USA