THE UNIVERSITY OF ARIZONA
TUCSON, ARIZONA 85721

DEPARTMENT OF COMPUTER SCIENCE

Icon Newsletter #20

Madge T. Griswold and Ralph E. Griswold

January 24, 1986

1. Implementation News

Version 5.10 of Icon for AT&T 3B Computers

Version 5.10 of Icon now supports the AT&T 3B2, 3B5, and 3B15 computers as well as the 3B20. In addition, several corrections recently have been made to the 3B portion of Version 5.10 of Icon. Persons who received Version 5.10 for use on a 3B computer prior to December 15, 1985 may wish to return their tapes for an updated version.

Version 6 of Icon

Don't panic — "Version 6" does not mean that a radically new version of Icon is on the horizon. Instead, it indicates a new implementation that deserves a distinct label, hence Version 6.

Although most of Version 5 of Icon is implemented in C, there is some assembly language code that is present for a variety of reasons. This assembly language code must be written anew for each processor to which Version 5 is ported. Even for a particular processor type, differences in assemblers and C compilers often require additional variants of the assembly code. These problems present major barriers to porting Icon to different computers.

Bill Mitchell is working on an implementation that contains a minimal amount of assembly code. Currently, the system contains no assembly-language code and is completely operational except for co-expressions. Co-expressions will require a small amount of assembly code, but it is expected to amount to no more than fifty instructions. Testing of Version 6 on various UNIX*-based processors indicates that the new code is very portable. The bottom line is that we expect Version 6 to compile and run without modification on most UNIX systems. Those who wish to have co-expressions should be able to implement the required routines quickly.

Version 6 will contain the new language features that have accumulated through version 5.10 and perhaps a few minor additions. The Icon book remains current; supplementary documentation will be provided with Version 6, as it has for the previous minor revisions to Version 5.

We hope Version 6 will be ready for distribution by summer. In the meantime, we will continue to distribution Version 5.10. However, persons who plan to port Icon to new computers should wait for Version 6 for two reasons: First, it will be much easier to port and second, we will shift our support from Version 5.10 to Version 6 as soon as Version 6 is ready for distribution.

2. The Icon Program Library for DOS Systems

The Icon program library, formerly available only for UNIX systems, is now available for computers running Version 5.9 of Icon under MS-DOS and PC-DOS. This library contains both complete programs and

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

collections of procedures that may be useful in building other programs. The programs in the library range from demonstrations and games to text-processing utilities. The procedures range from straightforward extensions of Icon's function repertoire to such esoteric areas as programmer-defined control operations.

In addition to the usefulness of the library in its own right, it provides examples of Icon programming techniques and may be helpful to persons who are just learning to program in Icon.

The Icon program library for DOS systems, which consists of a diskette and hardcopy documentation, is available for the cost of media, handling, and shipping. See the form at the end of this Newsletter. Persons who do not already have DOS Icon also can use this form for ordering a copy.

3. Icon Workshop Anyone?

Viktors Berstis has proposed an Icon workshop, to be held sometime in 1988 and has offered to coordinate arrangements. Viktors has suggested Hawaii as a possible location.

In order to determine whether or not there is sufficient interest in such a workshop, we have attached a questionnaire to this issue of the Newsletter. Please let us know your thoughts on the subject.

4. Availability of the Icon Book

Version 5 of the Icon programming language is described in the book:

The Icon Programming Language, Ralph E. Griswold and Madge T. Griswold, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1983. ISBN 0-13-449777-5.

The book is available in many college bookstores. Any general bookstore that handles special orders can obtain it from Prentice-Hall. Individuals can order directly from Prentice-Hall by calling (201) 767-9520. The book may also be ordered from the Icon Project at The University of Arizona for its list price of $22.95, which includes parcel post delivery in the United States. The form for ordering DOS Icon, attached, may be used to order the book.

The Icon Programming Language is current. No revision is scheduled, despite design changes incorporated in Version 6.

5. New Payment Policy for Icon Materials

Because of the high charges levied by banks for collection of funds outside the United States, our bank no longer accepts checks or money orders written in currency other than U. S. dollars. In addition, checks in U. S. funds must specify a bank in the United States through which payment is to be made. When placing orders from abroad, add $10 to checks written on banks that do not have branches in the United States. Checks for U. S. dollars written on banks that have branches in the United States are acceptable without additional charge.

We expect payment to accompany orders. We will, however, accept purchase orders from organizations that cannot provide payment in advance. There is a $5 charge for administrative processing of such orders.

6. Programming Corner

String Scanning

The string scanning expression in Icon,

$expr_1$? $expr_2$

is often regarded by programmers as somewhat of a mystery. This expression is technically a control structure, since it cannot be cast as a procedure call. The reason for this is that actions must be taken after $expr_1$ is evaluated but before $expr_2$ is evaluated, while in a procedure call, all arguments are evaluated before the procedure gains control.

The actions taken between the evaluation of $expr_1$ and $expr_2$ relate to the global variables **&subject** and **&pos**. These "state variables" for scanning are set to the string value of $expr_1$ and 1, respectively. If these variables were not set before $expr_2$ was evaluated, $expr_2$ could not "operate on" the value of $expr_1$.

As mentioned above, **&subject** and **&pos** are global variables and their values are accessible throughout the program. They are not affected by procedure calls. If they were, it would not be possible to write "matching procedures" to extend the built-in repertoire of matching functions (**tab** and **move**).

A string scanning expression, however, cannot just set the values of these variables. In order for nested scanning and scanning in mutual evaluation to work properly, the current values of the state variables must be saved before new values are set and then restored when the scanning expression is complete. Thus, the scope of the scanning variables can be thought of as being dynamic with respect to scanning expressions.

Although the string scanning expression cannot be cast as a procedure call, it can be cast as nested procedure calls:

$expr_1$? $expr_2$ → Escan(Bscan($expr_1$), $expr_2$)

In the nested call, $expr_1$ is evaluated first. **Bscan** is then called before $expr_2$ is evaluated. Thus, **Bscan** can manipulate the state variables before $expr_2$ is evaluated, first saving the current "outer" values, then setting the new "inner" values for $expr_2$ to operate on. If $expr_2$ produces a result, **Escan** is called. It restores the outer values before producing the result provided by $expr_2$. On the other hand, if $expr_2$ fails, **Bscan** is resumed and can restore the outer values before it, too, fails.

Note that **Escan** must have access to the outer values saved by **Bscan**. This is accomplished by passing these values as the result produced by **Bscan**. Since there are two values, a record can be used. Procedures to model the string scanning expression in this fashion are:

```
record ScanEnvir(subject, pos)

procedure Bscan(e1)
   local OuterEnvir
   OuterEnvir := ScanEnvir(&subject, &pos)
   &subject := e1
   &pos := 1
   suspend OuterEnvir
   &subject := OuterEnvir.subject
   &pos := OuterEnvir.pos
   fail
end

procedure Escan(OuterEnvir, e2)
   local InnerEnvir
   InnerEnvir := ScanEnvir(&subject, &pos)
   &subject := OuterEnvir.subject
   &pos := OuterEnvir.pos
   suspend e2
   &subject := InnerEnvir.subject
   &pos := InnerEnvir.pos
   fail
end
```

Note that if $expr_2$ produces a result, **Escan** suspends. If it is resumed, **Escan** restores the inner values and fails, forcing $expr_2$ to be resumed. This allows $expr_2$ to produce a sequence of results.

This is all there is to the string scanning expression — the maintenance of state variables. All of string analysis and "pattern matching" comes from matching functions in $expr_2$ that examine **&subject** and change **&pos**. These functions are simple. For example, **tab(i)** can be modeled as a procedure as follows:

```
procedure tab(i)
   suspend .&subject[.&pos:&pos <- i]
end
```

This leaves the question of where all of the power of string scanning comes from. It derives from the expression evaluation mechanism of Icon: generators and goal-directed evaluation.

Exercises:

1. There is a slight flaw in the procedures given above as a model for string scanning. What is it? *Hint:* It has nothing to do with the maintenance of state variables.

2. Write a model of string scanning that uses co-expressions and a programmer-defined control operation Scan{$expr_1$, $expr_2$} in place of $expr_1$? $expr_2$.

3. Why are the explicit dereferencing operations necessary in the procedure for tab?

A Programming Idiom

Version 5.10 of Icon has a new sort option for tables that produces a single list of alternating entry and assigned values. For example,

```
a := sort(t, 3)
```

assigns such a list to a. This option is more convenient and much more efficient in many cases than the standard sort option

```
a := sort(t, 1)
```

that assigns to a a list of two-element lists.

Consider the problem of writing the entry and assigned values of a table, side-by-side in two columns, using the new option. The

obvious approach is

```
a := sort(t, 3)
every i := 1 to *a - 1 by 2 do
   write(a[i],"\t",a[i + 1])
```

Students in our class on string and list processing techniques came up with a different approach:

```
a := sort(t, 3)
while(get(a), "\t", get(a))
```

To be sure, this approach "destroys" the list a, but that normally makes no difference. And not only is this approach simpler than the "obvious" one, it is faster too!

Exercise: The elements produced by sort(t, 3) are in increasing order of the entry values. How could the approach above be modified to write the output in descending order of the entry values?

Teasers

Steve Wampler provides two more teasers. Suppose that

```
t := table([])
```

Also suppose that getword is a procedure that produces a word from a line of text taken from a file in which lineno is the current line number.

1. Why does the following program segment never increase the size of the table t?

```
while word := getword() do
   put(t[word]),lineno)
```

2. Why does the following program segment increase the size of the table when the previous one does not?

```
while word := getword() do
   t[word] |||:= [lineno]
```

Other Exercises

1. What is the upper bound on the number of results that find(s1, s2) can produce?

2. The two code segments

```
        return x
```

and

```
        suspend x
        fail
```

are often thought of as being operationally equivalent. Is this really true? If not, explain the difference.

3. It is frequently claimed that the outcome of looping expressions such as

        while *expr*$_1$ do *expr*$_2$

is failure — that is, that the loop itself produces no results. Is this always true? If not, give a counter example.

Trivia Corner

Write the shortest possible Icon program whose translation produces at least one instance of every different ucode instruction in Icon's intermediate language. (See TR 85-19 for a description of ucode.) Interpret "shortest possible" to mean the fewest number of characters.

7. New Documents

Four new technical report related to Icon are available:

TR 85-25, *Programming with Generators*, is the first in a projected series of reports describing techniques for programming in Icon. This first report covers generators, goal-directed evaluation, and programming techniques that exploit these features of expression evaluation.

TR 86-1, *An Animated Display of String Pattern Matching*, presents the model of string scanning described in this Newsletter's Programming Corner and describes a tool for displaying the details of pattern matching. The tool runs on a Sun Workstation and uses its windowing system to control the display.

TR 86-2 contains accumulated material from the Programming Corner of previous Icon Newsletters. There is nothing in this report that has not appeared previously; it is being printed in this form to make it more accessible to persons who did not receive previous Newsletters.

TR 86-3 is the manual that accompanies the Icon program library for DOS systems that is described above.

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

Acknowledgement

The procedural model of string scanning described in the Programming Corner is due to Ken Walker.

Icon Workshop

A small Icon conference has been proposed for sometime in 1988. Would you attend? ______
With what probability? ______

Would you submit a paper for this conference? ______ If so, on what subject?

Specific subject areas you would like to see covered:

Preferred conference location: Hawaii ______ Other: ____________

Would you be willing to help with arrangements? ______

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

Return this form to:

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

Request for Icon Documents

Please send the documents checked below to:

- ☐ *Programming in Icon; Part I — Programming with Generators* (TR 85-25)
- ☐ *A Pattern-Matching Laboratory; Part I — An Animated Display of String Pattern Matching* (TR 86-1)
- ☐ *Programming in Icon; Problems and Solutions from the Icon Newsletter* (TR 86-2)
- ☐ *The Icon Program Library for DOS; Version 5.9* (TR 86-3)

- ☐ 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.

Request for Version 5.9 of Icon for DOS

Version 5.9 of Icon for DOS is distributed on a single 5¼" diskette. The Icon program library, which contains a variety of application programs and collections of procedures, is available on a separate diskette. The formats available are listed below.

Ship to:

name	______________________
address	______________________
	______________________
	______________________
	______________________
telephone	______________________
electronic mail address	______________________

Version 5.9 of Icon:	2S/DD diskette (most PCs)	$15.00 each	________
	2S/HD diskette (IBM AT)	$18.00 each	________
	1S/QD diskette (DEC Rainbow)	$18.00 each	________
Icon Program Library:	2S/DD diskette (most PCs)	$20.00 each	________
	2S/HD diskette (IBM AT)	$23.00 each	________
Icon Book:		$22.95 each	________
Additional charges (see below):			________
		total	________

Enclose a check payable to the University of Arizona for the amount due. There is a $5 service charge for processing purchase orders. Additionally, there is a $10 charge for a check drawn on a bank that does not have an office in the United States. Payment *must* be in U. S. dollars; other currencies are not acceptable.

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