

---

# The Icon Newsletter

---

No. 51 – December 1, 1996

---



## Contents

Third Edition of the Icon Book .....	1
Graphics Programming Book .....	1
Version 9.3 of Icon .....	2
Version 9.3 of the Program Library ...	2
New MS-DOS Implementation .....	2
Icon in Java .....	2
Teaching Icon .....	3
Web Links .....	7
Chicon .....	7

## Third Edition of *The Icon Programming Language*

The third edition of *The Icon Programming Language* is now available.

In preparing the third edition, we revised much of the material in the second edition and added chapters on debugging, the Icon program library, and an overview of graphics. There also are new appendices and more reference material.

Here's the publication information:

*The Icon Programming Language*, Ralph E. Griswold and Madge T. Griswold, Peer-to-Peer Communications, Inc., ISBN 1-57398-001-3, \$34.95.

The book can be ordered from the Icon Project, which pays handling and shipping costs to ad-

resses in the United States, Canada, and Mexico. There is an additional charge for shipping to addresses in other countries. See the order form enclosed with this *Newsletter*.

The book also can be ordered from the publisher:

Peer-to-Peer Communications, Inc.  
P.O. Box 640218  
San Jose, California 95164-0218  
U.S.A.

voice: 408-420-2677  
fax: 408-435-0895  
info@peer-to-peer.com  
<http://www.peer-to-peer.com>

Peer-to-Peer Communications' books are distributed to bookstores worldwide through International Thomson Publishing. In the United States, contact ITP at:

7625 Empire Drive  
Florence, KY 41042

For addresses of ITP offices in other countries, contact Peer-to-Peer.

---

## Graphics Programming Book

We have contracted with Peer-to-Peer Communications, Inc., the publisher of the third edition of *The Icon Programming Language*, to publish *Graphics Programming in Icon* as well.

The date of publication is uncertain at this point, but we are aiming for late in the summer of 1997.

Details are subject to change, but we expect a book about 512 pages with eight color plates and a CD-ROM. The price should be around \$45.

We're contemplating the following content for the CD-ROM, which will be "universal" and usable on different platforms:

1. Material from the book
  - program segments
  - images
  - reference material from appendices
2. Icon material
  - source code
  - executables for various platforms
  - program library
  - documentation
  - data
3. Tools
  - compression and archiving
  - file conversion
  - image viewing
4. Images
  - pictures
  - color palettes
  - patterns and backgrounds
  - icons and ornaments

Let us know if you have any suggestions for other content for the CD-ROM.

---

## Version 9.3 of Icon

Version 9.3 of Icon is in the final stages of testing. This version contains a few new features and improvements to the graphics facilities. It also fixes several bugs.

Watch our Web site for announcements of releases of the source and executable files for various platforms as they become available.

## Version 9.3 of the Program Library

Version 9.3 of the Icon program library now is available. This version of the program library not only contains new material, but more of the procedures have been packaged in modules to reduce the number of files in the library and to make it easier to find commonly needed procedures.

Version 9.3 files can be downloaded from our FTP site:

<ftp.cs.arizona.edu>

From there, cd /icon/library. That area contains library packages in several different forms.

Or you can use our Web site:

<http://www.cs.arizona.edu/icon/>

The section **Program Library** has links to indexes for the library that lead to the individual files. The complete collection can be downloaded by following the [FTP Area](#) link.

## New MS-DOS Implementation

A new MS-DOS/386 implementation of Version 9 of Icon with graphics is nearing completion. This implementation also will run under various versions of Windows in a DOS box in full screen mode, or in OS/2 2.0 similarly.

Completion of this implementation is outside of our control, but it seems likely it will be ready before the end of 1996.

---

## Icon in Java

Todd Proebsting is heading a project to construct an entirely new Icon translation system. The compiler will translate Icon into Java for execution in any Java environment. The system will also include a completely new Icon run-time system. While we don't plan to change the Icon language, modest extensions to data types are anticipated (for example, using Unicode characters in strings and csets). The project is a large undertaking and has just begun—it will be some time before it bears fruit.

This is an experiment, and we will continue to support the present C implementation of Icon regardless of the outcome.

---

## Teaching Icon

*Editor's Note: The following article was contributed by Bill Mitchell.*

I've taught Icon only once, in the context of a comparative programming languages class, C.Sc. 372 at The University of Arizona. Icon was the second of four languages studied in turn. Icon followed ML and in turn was followed by C++ and Prolog. Nine hours of lecture were budgeted for Icon.

During the course I came across a few things that perhaps are worth mentioning, as follows.

### *Overall Strategy*

A fundamental decision made when planning the order of presentation was to present first the

aspects of Icon that are close to conventional languages and later bring in generators and string scanning operations. This is based on the observation that one can do quite a number of interesting things without using either generators or string scanning.

A fair amount of time was spent describing Icon's place in the programming language community. My experience is that Icon's greatest strength is that it can be used to build tools easily, the construction of which would not be cost-effective in conventional languages.

### **ie** — *The Icon Evaluator*

Being a great fan of Icon, I wanted to introduce the language in the best light possible, but following ML presented an immediate problem: ML has an interactive mode and Icon does not.

I felt that ML's interactive mode, the commonly seen read-evaluate-print loop, proved to be a great aid in learning ML, and I wanted something similar for Icon. I produced a hasty solution, the Icon Evaluator (**ie**) based on the idea of `interpe.icn` in the Icon program library. `interpe.icn` takes an arbitrary Icon expression, wraps it in a main procedure, and then translates and runs the program. **ie** extended that idea a bit to provide context so that expressions can use the results of previous expressions.

With **ie** in hand it was possible to present overhead slides that showed expressions and the result of evaluating them. For example:

```
%
ie Icon Evaluator, Version 0.0, ? for help
][ 3+4;
  r1 := 7 (integer)
][ 3.4*5.6;
  r2 := 19.04 (real)
][ "x" || "y" || "z";
  r3 := "xyz" (string)
][ reverse(r3);
  r4 := "zyx" (string)
][ center("hello",20,".");
  r5 := ".....hello....." (string)
```

**ie** provided benefits to both the instructor and the student. When preparing slides, **ie** made it possible to see easily if an expression produced what was expected. For students, **ie** made it easy to try out code in a controlled environment that presented results in an unambiguous fashion.

### *Expression Failure*

Failure is a fundamental concept in Icon and I wanted to be sure that students really understood it. The example of an out-of-bounds string subscript was used to introduce the concept:

```
][ s := "testing";
  r := "testing" (string)
][ s[50];
  Failure
```

String subscripting was chosen because the same construct presents conventional languages with a problem: What should happen in this situation? With the notion of failure, Icon avoids the two common solutions of throwing an exception or producing a contrived value of some sort (perhaps a null string).

It was said that "s[50] fails — it produces no value" and then this rule was cited:

"An operation is performed only if a value is present for all operands. If a value is not present for all operands, the operation fails."

Examples were built on that rule:

```
][ "x" || s[50];
  Failure
][ reverse("x" || s[50]);
  Failure
][ s := reverse("x" || s[50]);
  Failure
][ s;
  r := "testing" (string)
```

My personal experience is that a very common source of bugs in Icon programs is due to unanticipated failures. Based on that I cited this rule:

"Unexpected failure is the root of madness."

### *Text Processing with split()*

An observation of mine is that a number of text processing problems can be addressed with a simple paradigm: split a line into pieces based on delimiters and then process those pieces. Without showing the implementation I introduced a procedure called `split()`. From the slides:

There is a procedure `split(s, delims)` that returns a list consisting of the portions of the string `s` delimited by characters in `delims`:

```
][ split("just a test here ", " ");
```

```

r := L1:["just", "a", "test", "here"] (list)
][ split("...1..3..45,78,,9 10 ", ", ", "");
r := L1:["1", "3", "45", "78", "9", "10"] (list)

```

A simple example then was posed:

Consider a file whose lines consist of zero or more integers separated by white space:

```

5 10 0 100 50
200 1 2 3 4 5 6 7 8 9 10

```

A program to sum the numbers in such a file:

```

link split
procedure main()
  sum := 0
  while line := read() do {

```

```

  nums := split(line, " \t")
  every num := !nums do sum += num
  }
  write("The sum is ", sum)
end

```

`split()` provides an easy way to turn input text into lists of strings. By using `split()` it was possible to delay string scanning but at the same time immediately work with nontrivial text processing problems.

### Generators

A basic question in presenting generators is whether to start with a built-in generator or with an Icon procedure. The latter approach was taken and this procedure was used:

```

procedure Gen()
  write("Gen: Starting up...")
  suspend 3
  write("Gen: More computing...")
  suspend 7
  write("Gen: Still computing...")
  suspend 13
  write("Gen: Out of results...")
  fail
end

procedure main()
  every i := Gen() do
    write("Result = ", i)
  end

```

The code still has the mystery of how `suspend` works, but I think it clearly shows that generation is a single computation punctuated with the production of values.

### Debugging Aids

Three debugging aids were discussed in some detail. The first was procedure call tracing. Icon's procedure call tracing facility is a wonderful debugging aid, and use of tracing was strongly encouraged. This program was used to show tracing in action:

### Downloading Icon Material

Most implementations of Icon are available for downloading via anonymous FTP:

`ftp.cs.arizona.edu (cd /icon)`

## *The Icon Newsletter*

Ralph E. Griswold, Madge T. Griswold,  
and Gregg M. Townsend  
Editors

*The Icon Newsletter* is published three times a year and is available on the World Wide Web. To receive printed copies, contact:

Icon Project  
Department of Computer Science  
The University of Arizona  
P.O. Box 210077  
Tucson, Arizona 85721-0077  
U.S.A.

voice: (520) 621-6613

fax: (520) 621-4246

e-mail: [icon-project@cs.arizona.edu](mailto:icon-project@cs.arizona.edu)



and



**Bright Forest Publishers**  
Tucson Arizona

© 1996 by Ralph E. Griswold, Madge T. Griswold,  
and Gregg M. Townsend

All rights reserved.

```

procedure main()
  write(sum(3))
end
procedure sum(n)
  return if n = 0 then 0 else n + sum(n - 1)
end

```

For UNIX *cs*h users, two aliases to facilitate tracing were suggested:

```

alias tn setenv TRACE -1
alias tf unsetenv TRACE

```

The second debugging aid discussed was the `image()` function and it was prefaced with a discussion of the built-in `image()`. `image()` was discussed after lists but before tables, and it was used to help in the presentation of tables.

The third debugging aid discussed was a procedure `snap()`, based on `snapshot()` from the Icon program library. `snap()` differed from `snapshot()` by displaying strings with blanks interspersed and by being less verbose to conserve space on slides. An example from a slide:

```

][ "testing" ? while move(1) do {
...   snap()
...   write(move(1))
... };
&subject = t e s t i n g
&pos = 2 |
e
&subject = t e s t i n g
&pos = 4 |
t

```

### Type Checking

Languages like Icon are typically described as being weakly typed, but Ralph Griswold claims Icon to be strongly-typed. I couldn't reconcile this for a long time, but now I find myself agreeing with his claim. The point stressed to the class was the one aspect of type-checking is when the checking is done. In Icon that checking is done at run time and the ramifications of that choice were discussed.

### Minor Points

- The design philosophy of Icon was said to be:
  - Provide a "critical-mass" of types and operations.
  - Free the programmer from worrying about details.

Put the burden of efficiency on the language implementation.

- Lists as stacks and queues

After presenting each of `push()`, `pop()`, `get()`, `pull()`, and `put()`, I used this visual summary, which I first saw used by Rich Saunders:

```

push ==>      ==> pull
pop <== List  <== put
get <==

```

- The / and \ operators

"Think of /x as succeeding when x is null because the null value allows the slash to fall flat."

- The & operator

It was emphasized that & is simply a trivial operator that returns its right-hand operand — the failure mechanism is what really does the work.

- `tab()` and `move()`

The precise operation of `tab()` was illustrated by showing this version of `tab()` in Icon:

```

procedure Tab(n)
  oldpos := &pos
  &pos := n
  suspend &subject[oldpos:n]
  &pos := oldpos
end

```

- Considerable use was made of reading from a pipe (via `open(..., "rp")`) to get some real-world textual data to work with. One of the programming problems did some simple analysis of the files in a directory tree by constructing a UNIX "find" command and processing output from it.

- A portion of an essay question on the final examination suggested that the student cite his or her favorite language among the four covered. Of those students who addressed the question, half chose Icon as their favorite.

### Icon on the Web

Icon is on the World Wide Web at  
<http://www.cs.arizona.edu/icon/>

## Headaches

By far the greatest difficulty in teaching Icon was determining an order of presentation that didn't have "forward references" — uses of concepts not covered yet. The features of Icon interlock very tightly and that produces a language that's easy to remember but difficult to unravel.

There was only one construct that created a problem that was inordinately difficult to address:

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

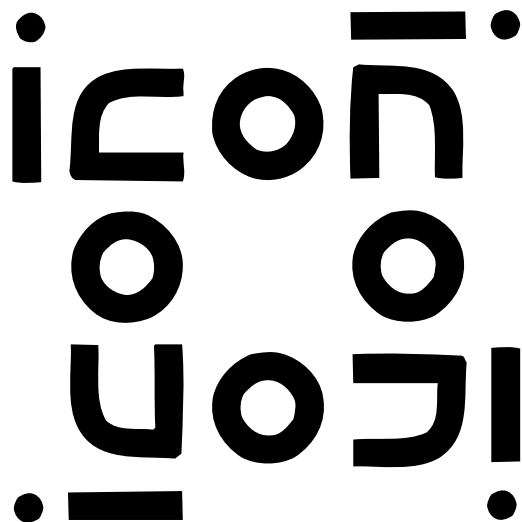
Interestingly, a number of students employed this construct without trouble by using list concatenation to add elements to the "lists":

```
T[x] ||:= val
```

## Course Outline

This is the complete order of presentation that was followed:

- introduction with brief history
- notion of type associated with values not variables
- arithmetic operations
- strings
- simple i/o
- expression failure
- if-then-else
- compound expressions
- semicolon insertion
- while
- break and next
- not
- &
- comparison operators



The logo consists of the word "icon" in a very bold, black, sans-serif font. The letters are thick and blocky. Above the 'i' and 'n' are small black dots. Below the 'i' and 'n' are larger black dots. The 'o's are also thick and blocky. The overall appearance is that of a heavy, industrial-style typeface.

- explicit conversions
- procedures
  - basics
  - call by value
  - omitted arguments
  - call tracing
  - scoping
- generators
  - basics
  - every
  - to-by
  - !
  - alternation
- lists
  - basics
  - reference semantics
  - stack and queue functions
  - sort()
- text processing with split()
- Image()
- tables
- string scanning
  - snapshots with snap()
  - move()
  - tab()
  - csets
  - upto()
  - many()
  - find()
  - match()
  - any()
  - pos()
  - default arguments for scanning
  - backtracking
- multiple generators
- the random operator

Due to time constraints, records and co-expressions were not covered.

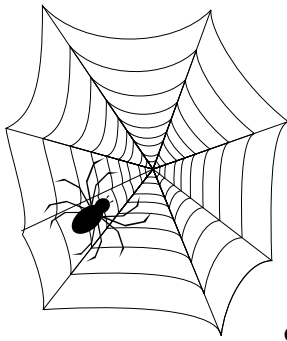
*Editor's Note: An improved version of `ie` under the name of `qei` is included in Version 9.3 of the Icon program library.*

---

## Web Links

### The Icon Analyst

We frequently get inquiries about *The Icon Analyst* from persons who are interested in subscribing but who are not sure it fits their needs or is worth the subscription price to them.



We're happy to send free sample copies of the *Analyst* to interested persons, but you also can learn a lot by browsing our Web site. From

our home page, follow the link [The Icon Analyst](#) in the **Documentation** section. That takes you to a page that provides a brief description of the *Analyst* and has links to a sample copy, a contents listing for all issues of the *Analyst*, and a permuted index of topics covered in the *Analyst* to date. Snapshots of portions of the last two are shown below and at the right.

#### Issue 35, April 1996

Building a Visual Interface ... 1-5  
 Corrections ... 5  
 Versum Numbers ... 5-11  
 Icon Glossary ... 11-12  
 Subscription Renewal ... 12  
 What's Coming Up ... 12

#### Issue 36, June 1996

Building a Visual Interface ... 1-4  
 Subscription Renewal ... 4  
 Quiz ... 5  
 Loading C Functions ... 5-9  
 Icon Glossary ... 9-12  
 Answers to the Quiz ... 12  
 What's Coming Up ... 12

#### Issue 37, August 1996

Expanded Analyst Format ... 1  
 Building a Visual Interface ... 1-3  
 Dynamic Analysis ... 3-9  
 Programming Tips (values in rotation) ... 10  
 Versum Predecessors ... 11-15  
 Icon Glossary ... 15-16  
 What's Coming up ... 16

#### Issue 38, October 1996

Random Numbers Revisited ... 1-5  
 Visualizing Concatenation ... 6-8  
 The Kaleidoscope ... 9-13  
 From the Library (filtering) ... 13-16  
 What's Coming up ... 16

#### Portion of the Contents for the *Analyst*

#### Other Links

Here are a few other Icon-related Web pages.

Clint Jeffery's work on Icon work at the University of Texas at San Antonio (with links to several Icon-related pages):

<http://www.cs.utsa.edu/research/icon/>

Designing a Visual Interface	Large Integers	4
Building a Visual Interface	Interface	33
Building a Visual Interface	Interface	34
Building a Visual Interface	Interface	35
Building a Visual Interface	Interface	36
Building a Visual Interface	Interface	37
Visual Interfaces	Visual Interfaces	31
An Introduction to X-Icon	Introduction to X-Icon	13
String Invocation	String Invocation	28
Applications of String Invocation	Applications of String Invocation	29
Programming Tips (buffered I/O)	Programming Tips (buffered I/O)	12
The Kaleidoscope	The Kaleidoscope	38
Lost Languages -- Rebus	Lost Languages -- Rebus	18
Lost Languages -- Seque	Lost Languages -- Seque	19
Lost Languages -- SL5	Lost Languages -- SL5	17
Large Integers	Large Integers	4
(last pattern)	(last pattern)	2
Launching the Analyst	Launching the Analyst	1
Libraries	Libraries	7
From the Library	From the Library	34
From the Library (filtering)	From the Library (filtering)	38
From the Library (options)	From the Library (options)	32
From the Library (organization)	From the Library (organization)	24
From the Library (structure images)	From the Library (structure images)	25
From the Library (symmetric drawing)	From the Library (symmetric drawing)	29
Lift for the Analyst	Lift for the Analyst	13
Lindenmayer Systems	Lindenmayer Systems	26
Lindenmayer Systems	Lindenmayer Systems	27
Lindenmayer Systems	Lindenmayer Systems	28
lines)	lines)	19
lists)	lists)	9
Loading C Functions	Loading C Functions	36
(local identifiers)	(local identifiers)	25
Looking Ahead	Looking Ahead	12
loops)	loops)	23
Lost Languages -- Rebus	Lost Languages -- Rebus	18
Lost Languages -- Seque	Lost Languages -- Seque	19
Lost Languages -- SL5	Lost Languages -- SL5	17

#### Portion of the Permuted Index for the *Analyst*

John Shipman's pages on writing CGI handlers in Icon and his "Cleanroom" protocol (with links to several other Icon-related pages):

<http://www.nmt.edu/tcc/help/lang/icon/cgi.html>

<http://www.nmt.edu/~john/soft/clean/>

#### Chicon

Several persons have asked us about the meaning of the Chinese characters for Chicon, Chinese Icon. Here's the response we got from the project:



[It is] pronounced "kin kwun" in Cantonese. Originally it was a phonetic translation of "Chicon" (Chinese Icon). Nevertheless, "kin kwun" means something "special" and "wonderful".

— Jason

