The Icon Newsletter

No. 28 — October 15, 1988

For New Readers

We are repeating here, for the benefit of new readers, some general information about Icon.

Program material distributed by the Icon Project is in the public domain. You may use it and provide copies to others without permission or restriction. Documents distributed by the Icon Project also may be copied freely, provided they do not bear a copyright notice. Permission is required to make copies of copyrighted documents.

The *Icon Newsletter* is free and available to anyone who wants to receive it. If you know persons who might like to receive the *Newsletter*, send us their names and addresses.

We do not sell the mailing list that we use for the *Newsletter*. Occasionally, we provide mailing labels to other organizations for some purpose related to Icon.

We also prepare a printed list of our subscribers that's available free to persons who want to locate others with common interests. If you do not want us to include your name on this printed list, let us know, and we'll take care of it.

Odds and Ends

Correction

Chris Fraser noted and Dave Hanson confirmed that the Department of computer Science got a PDP-11 late in 1977, not in 1979 as reported in the last *Newsletter* ("A Brief History of Icon"). While this may not strike you as important, it caused us to wonder why the first UNIX implementation of Icon was so long in getting started.

We're Flattered

In his keynote address at ICEBOL3 (*See* "ICEBOL3", *Icon Newsletter* No. 27, p. 2), Paul Abrahams discussed the history and characteristics of several programming languages. The following quotation, printed here by permission of Paul and Dakota State College, is from his address:

> The culture of Icon is the culture of shared connoisseurship. I think of Icon as being like a fine small French restaurant that not many people know about, where the clientele is loyal and appreciative, the chef is devoted to producing creative cuisine of the highest quality, and the assistant chefs are themselves talented and devoted to their craft. Icon is a language where everything fits together, and its design is enhanced by careful attention to small details. Many of Icon's syntactic niceties such as the use of newlines to terminate most statements, would be useful even in other languages quite different from Icon.
>
> Unlike all the other languages I have talked about with the possible exception of Algol 60, Icon has been entirely noncommercial. Tucson has been the Rome of Icon and the University of Arizona its Vatican. The University of Arizona is the seat of Icon learning and expertise and the source of all implementations of Icon, which are available for nothing more than the cost of distribution and can be freely copied.

Copies of the proceedings from ICEBOL3 are still available for $20 from:

Division of Liberal Arts
114 Beadle Hall
Dakota State College
Madison, SD 57042

Icon Workshop

Northern Arizona University hosted a workshop on the Icon programming language on July 27-29 in Flagstaff, Arizona. Fifteen persons closely involved with Icon attended and exchanged information and ideas.

Many topics were discussed, including object-oriented features, global program organization, currency, pattern matching, user interfaces, and implementation.

Copies of the workshop report are available, free of charge, from the Icon Project. Ask for IPD61.

The Icon Newsletter

Madge T. Griswold and Ralph E. Griswold
Editors

The Icon Newsletter is published aperiodically, at no cost to subscribers. For inquiries and subscription information, contact:

Icon Project
Department of Computer Science
Gould-Simpson Building
The University of Arizona
Tucson, Arizona 85721
U. S. A.

(602) 621-2018

FAX: (602) 621-4246

Electronic mail may be sent to:

icon-project@arizona.edu

or

... {uunet, allegra, noao}!arizona!icon-project

© 1988 by Madge T. Griswold and Ralph E. Griswold

All rights reserved.

Icon "Clip Art"

As you'll see in this *Newsletter*, we continue to get contributions of art work related to Icon. It's fun for us, and we hope readers enjoy seeing some visual variety in the *Newsletter*. We'll continue to offer $15 in credit at the "Icon Store" (books excluded) for each piece of art published in the *Newsletter*.

Implementation News

MS-DOS Icon for 386 PCs

We now have an implementation of Icon that runs on 386 PCs such as the PS/2 Model 80. It requires 2MB of RAM and will use more. This implementation is fast — nearly twice as fast as the standard MS-DOS Microsoft C version of Icon for many programs and up to eight times as fast as the MS-DOS Lattice C version of Icon. (*See* "Benchmarks" later in this *Newsletter*.) MS-DOS/386 Icon is available on both 3.5" and 5.25" diskettes. See the order form at the end of this *Newsletter*.

Bob Goldberg provided this implementation; our thanks to him.

MS-DOS Icon under Turbo C

We've finally managed to get Icon to compile and run under Turbo C. (The problems were ours, not Borland's.) Our thanks to Clint Jeffery and Terasalei Hiroyasu for solving the problems. Support for Turbo C is now included in the current distribution of the MS-DOS Icon source.

Downloading Icon Material

Several of the implementations of Icon are available for downloading electronically:

BBS: (602) 621-2283

FTP: arizona.edu (/usr/ftp/icon)
(128.196.6.1 or 192.12.69.1)

XENIX V/386 Icon

We now have a 386 version of Icon for XENIX. Both 3.5" and 5.25" diskettes are available. See the order form at the end of this *Newsletter*. Thanks to Ronald Florence for this implementation.

Icon for the IBM 370 Architecture

Icon for IBM 370 computers using VM is now up and running thanks to Cheyenne Wills, Robert Knight, and the Princeton University Computer Center. Testing, debugging, and documentation remain to be done. We don't have a firm release date yet, but unless there are unexpected delays, early 1989 is probable.

Amiga Icon

We've lost touch with the person who previously provided executable files for Icon on the Amiga. If you have an Amiga with at least 1MB of RAM, the Lattice 3.04 C compiler, and are willing to compile and test Version 7 of Icon for public distribution, please get in touch with us.

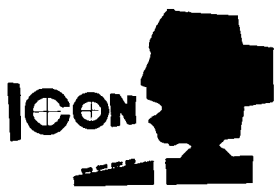
Implementation Updates

We're in the process of updating the present Version 7.0 implementations to Version 7.5. The language changes in Version 7.5 are minor; there is no need for persons who presently have Version 7.0 in executable form to update. The changes to the implementation, however, are significant and persons working with the source code should consider upgrading as Version 7.5 becomes available.

So far, only the MS-DOS executables and source have been updated. The UNIX and VMS implementations are scheduled for updating later this fall. Updated distributions are noted in the ordering information at the end of this *Newsletter*.

From Our Mail

After reading your remarks about the possibility of a commercial version of Icon, I heard a rumor that something really is in the works. Is that true?



Yes. Catspaw, Inc. plans to market a Macintosh implementation of Icon. It will be a stand–alone application with a standard Macintosh interface and some new language features. Delivery is scheduled for the first quarter of 1989. For more information, contact:

Catspaw, Inc.
P.O. Box 1123
Salida, CO 81201

719-539-3884

Does Version 7 of Icon run under VMS 5.0?
Yes.

Do you have a version of Icon for OS/2 yet?
We're testing a pre-release version.

I have Icon source for MS-DOS, but I think it's out of date. I'd like to get the most current version of the source. How do I do this?
We update the distributed version of the MS-DOS source code for Icon when there are major changes, once or twice a year. There is an update subscription service for MS-DOS that provides an update and technical notes every three or four months. The cost of the subscription service is $30 for five updates ($15 additional for airmail postage overseas). To subscribe, send payment and the serial number of your source-code diskettes.

Does Icon compile with Mark Williams Let's C under MS-DOS?
Yes, but the *Let's C* preprocessor does not do everything Icon needs, so a stand–alone preprocessor is required.

I'm using Icon for MS-DOS and am having troubles reading single characters typed from the keyboard. It seems even reads() *requires a carriage return before it will produce input.*
Function reads() was designed for reading files in "stream mode", not for reading single characters from the keyboard, where buffering prevents input until there is a "line". In MS-DOS, use the function getch() (or getche() if you want characters echoed). You may want to use these functions in combination with kbhit(), which succeeds if there is a keyboard character waiting. These functions are part of the extended function package of MS-DOS.

How can I integrate Icon with my existing C programs?
At present, the only way to do this is to write C functions for Icon that call your C programs. To do this, you need to have the source code for Icon and compile it

with the new C code. The method for doing this is described in the Icon implementation book.

I just downloaded Version 5.9 of Icon from a bulletin board. Is there a more recent version? If so, how do I get it?

Version 7 of Icon is current. There have been many additions to the language since Version 5.9 and a lot of implementation improvements as well. It seems inevitable that old versions of Icon will be around forever, but the most recent version is always available from the Icon Project.

A Contribution from Users

Run-Time Record Definition

by Andy Heron and Carole Thornton,
Government Communications Headquarters, England

Editors' Note: This is the first part of a long contribution that we'll continue in subsequent *Newsletters* as space permits.

Here we describe some work that is part of a database project written in C and Icon. Our experience with Icon (50,000 lines and growing) shows that it is an excellent prototyping language. The plan is to rewrite the most heavily used parts in C, for performance reasons, but a large body of Icon code will remain, as maintenance is also easier with Icon.

Uppercase names have been used for general-purpose functions and procedures to avoid clashing with future Icon functions.

The Icon procedure COMPLETE_IMAGE is a debugging aid that prints a description of an arbitrary data structure — a generalised version of the Icon function image. Consider the following fragment of Icon code.

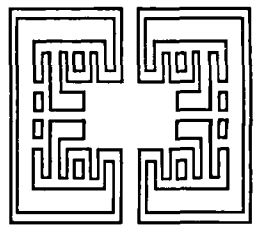
```
record address(no,street,city)

procedure debug()
  COMPLETE_IMAGE( [1,"abc",address(1,"high street",
    "Toytown")])
end
```

The procedure debug will write the following to standard error.

```
var_1: list(3)
var_1 [1]:              1
var_1 [2]:              "abc"
var_1 [3]:              record address(3)
var_1 [3].no:           1
var_1 [3].street:       "high street"
var_1 [3].city:         "Toytown"
```

The leading identifier is created by COMPLETE_IMAGE (*i.e.*, var_1) and the numeric part is incremented by one each time COMPLETE_IMAGE is called. However the user can specify an identifier (*e.g.*, the name of the variable being imaged) and request output on another file. In addition, COMPLETE_IMAGE returns its first argument so that it can be used on partial expressions.

COMPLETE_IMAGE can describe structures of arbitrary depth and it checks for cycles that would cause it to go into an infinite loop. We and other users of Icon at GCHQ have found this procedure to be extremely useful, and far superior to Icon's image for debugging.



COMPLETE_IMAGE requires some simple extensions to the Version 6.0 implementation of Icon to preserve field names until run-time. It was a simple task to include a list of field names in each Icon record constructor and this extension supports the function FIELD that generates the field names of a specified record constructor. The function RECORD_CONSTRUCTOR returns the record constructor of any record, and this pair of functions is all that is needed for COMPLETE_IMAGE.

First we examine how Version 6.0 of Icon accesses fields of a record. Each field name in an Icon program is assigned a unique number and references to a field name in the source code are converted to its numeric identifier in the compiled program (*i.e.*, in the icode read by the interpreter). Each record type in an Icon program is also assigned a unique number (stored in the record constructor) and the linker builds a table with a row for each record type and a column for every field name, using their numeric identifiers as indices. Entries in this table specify where in the record a field is to be found (–1 means it is not present in the record).

Thus, to form a new record constructor one must add another row to the record/field table. The fields of the new record type need not be defined in the existing program, but any field references must be defined in a record declaration or the linker will complain (*e.g.*, x.y requires y to be in some record declaration).

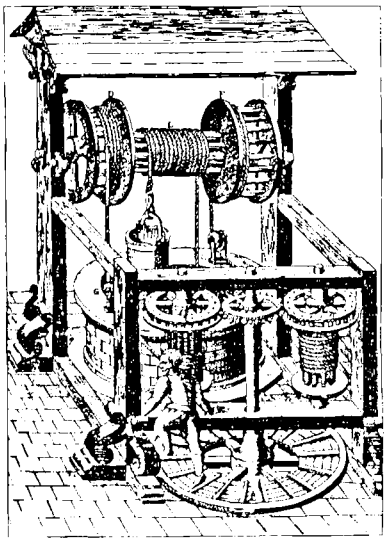
A complete list of the field names used in the program is formed and retained as part of the program by a simple extension to the Version 6.0 implementation of Icon. The function FIELD with a null argument

will generate all field names in the order needed. Thus one can form a new row of the record/field table in Icon code provided a simple function is used to update the table itself, and in this way one can create records at run-time and access them in the normal way. However we keep the name of the function a closely guarded secret, and provide more user friendly ways of creating new record types.

Sometimes the record is completely unknown until run-time, including the fields needed. The function RECORD_FIELD_POS returns the position of a named field in the record and then Icon will allow us to access the field by position.

Editors' Note: The listing of COMPLETE_IMAGE is too *long to include here.* Copies are available, free on request, from the Icon Project. Ask for IPD60.

Inside Icon



Occasionally we're asked how we keep track of the source code for all the different implementations of Icon. Fortunately, almost all of Icon is written in C. There is just one master copy of the C source code in Icon. All the variants for the different implementations, ranging from UNIX to MS-DOS, are obtained mechanically from the master copy. This enables us to make changes at only one place and to keep track of differences among implementations without having to synchronize many versions of the same files.

Maintaining a single master source may seem simple enough on the surface, but it's really rather complicated. The source for Icon must account for differences in operating systems, differences in C compilers, and even differences in the syntax of path names for include files. We use several techniques (none novel) to handle all of this.

As far as possible, we use conditional compilation, via the C preprocessor, to configure Icon for specific operating systems and C compilers. If you look at the source code, you'll see a number of segments of the form:

```
/*
 * The following code is operating-system dependent
 *  [@fsys.12]. Perform system call. Should be
 * RunErr(-121, NULL) if not supported.
 */

#if PORT
  RunErr(-121, NULL);
#endif                          /* PORT */

#if ATARI_ST || VMS
  MakeInt(system(systemstring), &Arg0);
  Return;
#endif                          /* ATARI_ST || VMS */

#if HIGHC_386
  RunErr(-121, NULL);
#endif                          /* HIGHC_386 */

#if MACINTOSH
  RunErr(-121, NULL);
#endif                          /* MACINTOSH */

#if AMIGA || MSDOS || UNIX || MVS || VM
  MakeInt((long)((system(systemstring) > 8) & 0377), &Arg0);
  Return;
#endif                          /* AMIGA || MSDOS || UNIX ... */

/*
 * End of operating-system specific code.
 */
```

The names like AMIGA and VMS refer to operating systems. PORT is a hook for new systems. Exactly one of these names is defined to be 1; the rest are defined to be 0. This allows conditional compilation to be phrased in terms of logical expressions that the preprocessor understands. Within the conditional compilation for a particular operating system, there are similar names for different C compilers to handle the code that is compiler-dependent. The root of the definitions for the names is an include file define.h, which is different for each operating system and compiler.

It's here that a single master file is not sufficient. In addition to define.h, there are a few other files that vary from implementation to implementation. Examples are assembly-language files needed for the co-expression context switching and arithmetic overflow checking. These files are contained in a hierarchy, organized first by operating system and then by C compiler, when that's necessary.

To put together the source code for a particular implementation, a copy of the master source is made first. Next, any files specific to the particular implementation are added. Unfortunately, that's not all there is to

it. As mentioned above, path syntax for include files varies. To handle this, edit scripts are applied to the source-code files. Some C preprocessors do not support "splicing" that Icon needs to concatenate strings. For this, a macro processor is used. Finally, some operating systems do not support hierarchical file systems. For such cases, all this has to be adapted to a flattened version of the source.

This process is all controlled by a *make* utility and a system of Makefiles that start at the top of the Icon source hierarchy.

What we have now works automatically. To configure a particular version of the source, we just "push a button". That button pushes other buttons, and so on. Eventually the desired version of the source code comes out the other end.

Perhaps this makes it sound simpler than it is. Engineering the overall system, crafting the buttons, writing the scripts for the various processors, and getting all the details right has taken a lot of work. But, now that it works, it's very satisfying to see it in action.

Our problem is that things are always changing. In order to get Icon running on a new C compiler or, worse, operating system, a lot of changes have to be made, and files specific to the new implementation have to be added. It's also easy to forget how things work. If some "button" needs changing, it make take a while to figure out how it relates to other "buttons" — or even why it's there, documentation notwithstanding.

Bugs

Several persons have commented that the function Int86 for MS-DOS is very slow and causes many garbage collections.

While this is not really a "bug" in the sense of producing the wrong results, it's enough of a practical problem to MS-DOS users of Icon to discuss here.

The problem really is one of design. The argument to Int86 is a nine-element list that specifies an interrupt number and the values of eight registers. The value returned is a similar list containing flags and new values of registers.

These two nine-element lists are typically allocated for every call of Int86. Icon lists carry a certain amount of storage overhead to support stack access and queue access — they're relatively large. Thus, calling Int86 frequently produces a lot of "storage throughput".

Storage allocation in Icon is fast, but garbage collection to reclaim unused storage can be time consuming. (The amount of time depends more on how much storage is in use than on how much is "garbage".)

If you need to use Int86 frequently, you can cut the expense of storage throughput in half by reusing a list rather than creating a new one. Thus,

```
while ... do
  L := Int86( [...] )
```

creates a new list with each call of Int86 in the loop, but

```
L1 := [...]
  ...
while...do {
  L1 [1] := ...
  L1 [2] := ...
  ...
  L2 := Int86( [L1] )
  }
```

creates only one list, L1, for the argument to Int86 and reuses it in the loop. This still doesn't avoid the list produced by Int86 each time it returns.

While lists are natural enough for the Icon programmer using Int86, a nine-argument function would have avoided the allocation for the call altogether. The best solution for returning the nine results of Int86 is not as obvious.

For the time being, users of Int86 will have to live with it as it is.

Icon Benchmarks

Every so often someone asks how fast Icon runs on a particular computer or how its performance compares when compiled with two different C compilers on the same computer. Complaints about poor performance of the expandable-region version for MS-DOS also are common.

Benchmarks are, of course, easily misinterpreted. It's easy to give a lot of figures without regard for the many factors that may affect them or their real meaning. Nonetheless, taken with appropriate reserve, benchmark results can be helpful in providing general ideas about performance.

Benchmarking is time consuming and, for us, relatively unrewarding. We don't have the stomach for doing hundreds of benchmarks, but we've done a few. We picked four programs that are relatively representative of Icon applications:

ipxref. This program, similar to the one in Version 6 of the Icon Program Library, produces a cross reference listing of an Icon program. It does lots of text processing and some list manipulation.

queens. This program produces the solutions to the non-attacking *n*-queens problem, including producing board representations for all solutions. It does a lot of generation and backtracking, as well as text synthesis. For testing purposes, *n* was 9.

rsg. This program, similar to the one in Version 6 of the Icon Program Library, generates randomly selected sentences. The program uses tables and lists extensively and synthesizes text. For testing, it was given a grammar for short poems and asked to produce 100 samples.

sieve. This program implements the sieve of Erastothenes, using set manipulation. The test produces the primes in the integers to 2000.

	ipxref	queens	rsg	sieve
AT/Microsoft	45.0	216.0	25.0	10.0
AT/Turbo	46.0	227.0	27.0	11.0
AT/Lattice	175.0	924.0	106.0	35.0
PS2/Microsoft	23.0	110.0	14.0	7.0
PS2/Turbo	23.0	143.0	20.0	7.0
PS2/Lattice	100.0	526.0	61.0	20.0
PS2/HighC	11.0	56.0	7.0	3.0
VAX 8650	5.4	27.4	3.0	1.2
MacII/MPW	17.4	87.0	10.9	4.7
Sun-2/120	37.8	179.0	21.7	9.4
Sun-3/110C-4	13.4	64.7	7.8	3.2

Timings in seconds (some clocks have only one-second resolution).

The table that follows gives timings for several different computers, operating systems, and with Icon produced under different C compilers.

Output was suppressed in all tests to avoid differences due to factors like disk access speed. This also suppresses differences in performances of different input/output libraries. Tests without suppressing output show minor differences in some cases, but nothing major. All tests were done with 65K string and block regions; none required region expansion.

The first two groups of tests were run under MS-DOS. "AT" refers to a turbo AT clone with a Norton computing index of 9.7. "PS2" refers to a PS/2 Model 80 with a Norton computing index of 17.6. The C compilers used to produce Icon for the MS-DOS tests were Microsoft C 5.0, Turbo C 1.5, Lattice C 3.22, and Metaware HighC (32-bit protected mode). The VAX 8650 tests were run under UNIX 4.3bsd with Icon built using the standard "cc" compiler.

The only figures that might be surprising are those for MS-DOS Icon compiled under Lattice C, which averages about four times slower than Icon compiled under Microsoft C. One of the problems here is that Lattice C is doing 32-bit pointer arithmetic, rather than just offset arithmetic. What you get in return is the ability to address a large amount of memory and also support for expandable memory regions in Icon — but at a very substantial penalty.

Documents Related to Icon

Several recently published papers and technical reports related to Icon are available from the Icon Project. Reprints of papers are free but they are in limited supply. There is a charge for the technical reports, as noted, to recover the cost of printing and shipping. (We used to be able to offer technical reports without charge and absorb the costs. That was when we had 200 subscribers to the *Newsletter*. We now have over 2,700 subscribers.) However, we'll provide one technical report free for every $15 of purchases of programs or books, provided the request for reports accompanies the order.

Reprints (free, limited supply). Order by title.

"Seque: A Programming Language for Manipulating Sequences", Ralph E. Griswold and Janalee O'Bagy, *Computer Languages*, Vol. 13, No. 1 (1988), pp. 13-22.

"Programming with Generators", Ralph E. Griswold, *The Computer Journal*, Vol. 31, No. 3 (1988), pp. 220-228.

"Garbage Collection of Strings and Linked Data Structures in Real Time", Kelvin Nilsen, *Software — Practice & Experience*, Vol. 18, No. 7 (1988), pp. 613-640.

Technical Reports. Order by TR number.

A Type Inference System for Icon, Kenneth Walker, TR 88-25. 31 pages. $2.50.

The Design and Implementation of a High-Level Programming Language for Pattern Matching in Real Time (doctoral dissertation), Kelvin Nilsen, TR 88-30. 126 pages. $6.50.

The Implementation of Generators and Goal-Directed Evaluation in Icon (doctoral dissertation), Janalee O'Bagy, TR 88-31. 93 pages. $5.50.

Quick Reference Sheets for Icon

We've been asked several times for a brief summary of Icon operations — something that would fit on a couple of pages and that could be used for quick reference. We never saw quite how all that information could be arranged in a small space.

Well, it's been done for us. The reference sheets on pages 9 and 10 of this *Newsletter* were provided by Bob Alexander (using MacDraw II). Our thanks to him for this "labor of love".

Clip Art Credits

Graphics that first appeared in *Newsletter* No. 26 and *Newsletter* No. 27 are credited in those issues.

Page 1. EPS clip art from Kwikee's *Potpourri.*

Page 3. Jacques Nel, Superpaint bitmap graphic, autotraced with Illustrator '88.

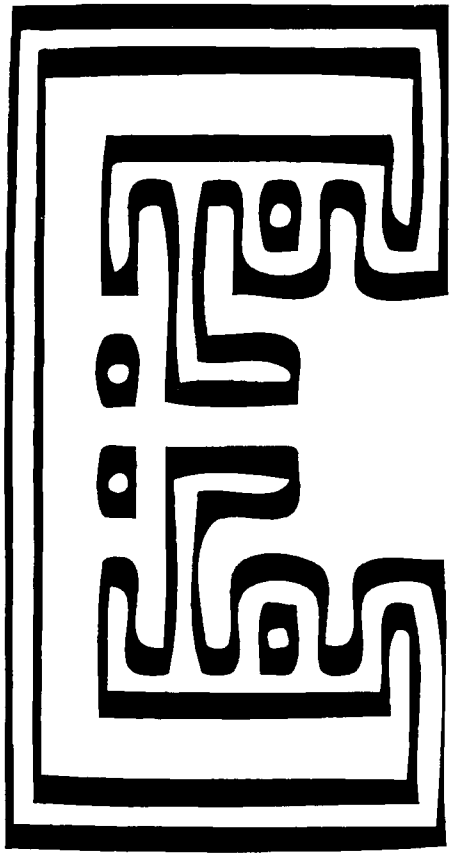
Page 4. Charles Richmond, Atari ST printed output, scanned and drawn over with Illustrator '88.

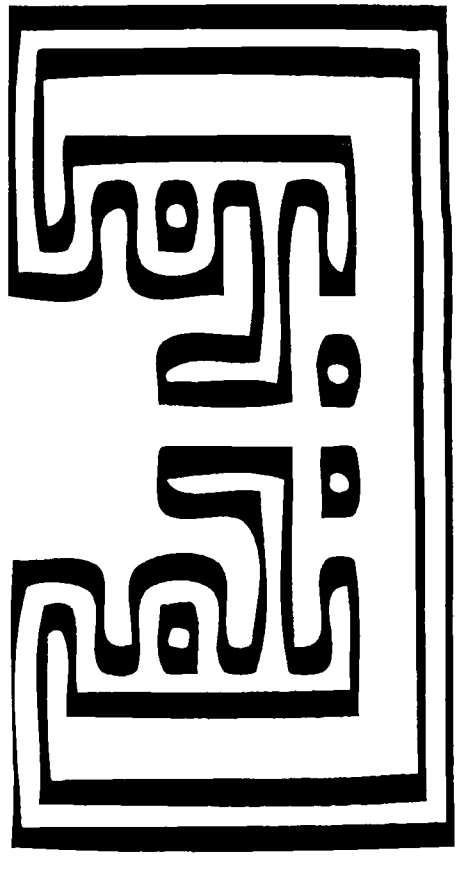
Page 5. Water drawing machine. Scanned image from *The Various and Ingenious Machines of Agostino Ramelli*, Dover Publications and Scolar Press, 1987.

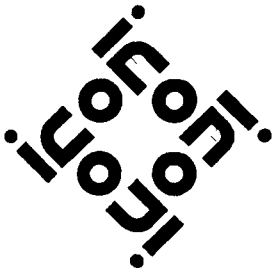
Page 6. *Australostoma opacum*, Stenopelmatidae, scanned image.

Page 6. Alan Davis, trukese love stick, pencil drawing, scanned and autotraced with Illustrator '88.

Page 8. Charles Richmond, Atari ST printed output, scanned and autotraced with Illustrator '88.







Icon Programming Language Reference Sheet

Functions

abs(n)	ior(i,j)	rename(s1,s2)
any(c,s,i,j)	ixor(i,j)	repl(s,i)
bal(c1,c2,c3,s,i,j)	ishift(i,j)	reverse(s)
center(s1,i,s2)	left(s1,i,s2)	right(s1,i,s2)
char(i)	list(i,x)	runerr(i,x)
close(f)	many(c,s,i,j)	save(s)
collect()	map(s1,s2,s3)	seek(f,i)
copy(x)	match(s1,s2,i,j)	seq(i,j)
cset(x)	member(x1,x2)	set(a)
delete(x1,x2)	move(i)	sort(a)
detab(s,i1,i2,...,in)	numeric(x)	sort(t,i)
display(i,f)	open(s1,s2)	stop(x1,x2,...,xn)
entab(s,i1,i2,...,in)	ord(s)	string(x)
errorclear()	pop(a)	system(s)
exit(i)	pos(i)	tab(i)
find(s1,s2,i,j)	proc(x)	table(x)
get(a)	pull(a)	trim(s,c)
getenv(s)	push(a,x)	type(x)
iand(i,j)	put(a,x)	upto(c,s,i,j)
icom(i)	read(f)	where(f)
image(x)	reads(f,i)	write(x1,x2,...,xn)
insert(x1,x2,x3)	real(x)	writes(x1,x2,...,xn)
integer(x)	remove(s)	

Keywords

&ascii	&errout	&pos
&clock	&fail	&random
&collections	&features	&regions
&cset	&file	&source
&current	&host	&storage
&date	&input	&subject
&dateline	&lcase	&time
&digits	&level	&trace
&error	&line	&ucase
&errornumber	&main	&version
&errortext	&null	
&errorvalue	&output	

Data Types

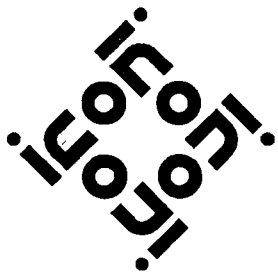
null	co-expression
integer	procedure
real	list
string	table
cset	set
file	*record types*

Escape Sequences

\b	backspace
\d	delete (rubout)
\e	escape (altmode)
\f	formfeed
\l	linefeed (newline)
\n	newline (linefeed)
\r	carriage return
\t	horizontal tab
\v	vertical tab
\'	single quote
\"	double quote
\\	backslash
ddd	*octal code*
\x*dd*	*hexadecimal code*
\^*c*	*control code*

Reserved Words

break	fail	record
by	global	repeat
case	if	return
create	initial	static
default	link	suspend
do	local	then
dynamic	next	to
else	not	until
end	of	while
every	procedure	



Icon Programming Language Reference Sheet

Expressions by Precedence

	all high precedence expressions
	all prefix expressions
\	limitation
@	transmit to co-expression
^	raise to power
*	multiply
/	divide
%	remainder
**	set intersection
+	add
–	subtract
++	set union
– –	set difference
||	string concatenation
|||	list concatenation
<	numeric comparisons
<=	...
=	...
>=	...
>	...
~=	...
<<	string comparisons
<<=	...
==	...
>>=	...
>>	...
~==	...
===	value comparisons
~===	...
|	alternation
to-by	integer sequence
:=	assignment
<–	reversible assignment
:=:	exchange
<–>	reversible exchange
augmented:=	augmented assignment
?	string scanning
&	conjunction
	all low precedence expressions

High Precedence Expressions

(*expr*)	parenthesized
{*expr1;expr2;...*}	compound
[*x1,x2,...*]	list creation
x.field	field reference
x[*i*] *x*[*i*+:*j*] *x*[*i*:*j*] *x*[*i*–:*j*]	subscripting
x(*x1,x2,...*)	invocation
(*x1,x2,...*)	mutual evaluation

Prefix Expressions

not	reversal of success/failure
|	repeated alternation
!	element generation
*	size of
+	numeric value
–	negative
.	value of (dereference)
/	succeed if null
\	succeed if non-null
=	match-tab
?	random selection
~	cset complement
@	activate co-expression
^	refresh co-expression

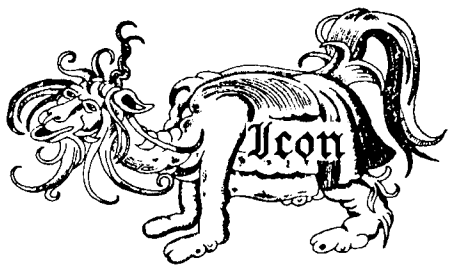
Low Precedence Expressions

create *expr*
return [*expr*]
suspend [*expr*] [**do** *expr*]
fail
break [*expr*]
next
case *expr* **of** {
expr: expr
...
[**default**: *expr*]
}
if *expr* **then** *expr* [**else** *expr*]
repeat *expr*
while *expr* [**do** *expr*]
until *expr* [**do** *expr*]
every *expr* [**do** *expr*]

Ordering Icon Material

Shipping Information: The prices listed at the end of this section include handling and shipping in the United States, Canada, and Mexico. Shipment to other countries is made by air mail only, for which there are additional charges as follows: $5 per diskette package, $10 per tape or cartridge package, and $10 per documentation package. UPS and express delivery are available at cost upon request.

Payment: Payment should accompany orders and be made by check or money order. Credit card orders cannot be accepted. Remittance *must* be in U.S. dollars, payable to The University of Arizona. There is a $10 service charge for a check written on a bank without a branch in the United States. Organizations that are unable to pre-pay orders may send purchase orders, but there is a $5 charge for processing such orders.



What's Available

Icon is available for several personal computers, UNIX, VMS, and also for porting to other computers. Source code is available in most cases.

The UNIX package contains source code, documentation in printed and machine-readable form, test programs, and related software – everything there is. It can be configured for most UNIX systems. The documentation includes installation instructions, an overview of the language, and operating instructions. It does not include either of the Icon books. Program material is available on magnetic tape, cartridge, or diskettes.

The VMS package contains everything the UNIX package contains except UNIX configuration information and UNIX-specific software. However, the UNIX and VMS systems are configured differently, and neither will run on the other system. The VMS package also contains object code and executables, so a C compiler is not required. The VMS package is distributed only on magnetic tape. *Note:* VMS Version 4.6 or higher is required to run Version 7 of Icon.

Source-code distributions for personal computers generally are provided separately from the executables. Each package contains printed documentation that is needed for installation and use. *Note:* Icon for personal computers requires at least 512KB of RAM.

Icon for porting is distributed on MS-DOS format diskettes. There are two versions, one with a flat file system and one with a hierarchical file system. Both versions are available in either plain ASCII format or compressed ARC format.

There are two documentation packages that contain more than is provided with the program packages: one for the language itself and one for the implementation.

Program Material

Note: All the distributions listed below are for Version 7 of Icon. Earlier Version 6 implementations that are not supported for Version 7 are still available. If you wish to order a Version 6 implementation, ask for a Version 6 order form, which is free.

Legend: The following symbols are used to indicate different types of media:

- 9-track magnetic tape
- DC 300 XL/P cartridge
- 360K (2S/DD) 5.25" diskette
- 400K (1S) 3.5" diskette
- 800K (2S) 3.5" diskette

All cartridges are written in raw mode. All 5.25" diskettes are written in MS-DOS format.

When ordering tapes, specify 1600 or 6250 bpi (1600 bpi is the default). When ordering diskettes that are available in more than one size, specify the size (5.25" is the default).

The symbol ☞ identifies material that is new since the last *Newsletter*. The symbol ☛ identifies material that has been updated since the last *Newsletter*.

Use the codes given at the beginning of the descriptions that follow when filling out the order form.

Icon for UNIX:

UT-T:		tar format	$25
UT-C:		cpio format	$25
UC-T:		tar format	$40
UC-C:		cpio format	$40
UD-M:	(5)	cpio format	$40

Icon for VMS:

VT:			$25

Icon for the Atari ST:

ATE:		executables	$15

Icon for MS-DOS:

DE:	(2)	executables	$20
DS:	(2)	source	$25

Icon for MS-DOS/386:

DS-386:	or	executables	$15

Icon for the Macintosh/MPW:

ME:		executables	$15
MS:	(2)	source	$25

Icon for the UNIX PC:

UPE:		executables	$15

Icon for XENIX:

XE:		executables	$15

Icon for XENIX/386:

XE-386:	or	executables	$15

Icon Source for Porting:

PF-A:	(4)	flat system, ASCII	$35
PF-K:	(2)	flat system, ARC	$25
PH-A:	(4)	hierarchical system, ASCII	$35
PH-K:	(2)	hierarchical system, ARC	$25

Documentation

LD: Language documentation package. *The Icon Programming Language* (Prentice-Hall, 1983) and two technical reports. $30.

ID: Implementation documentation package. *The Implementation of the Icon Programming Language* (Princeton University Press, 1986) and update. $40.

NL: Back issues of the *Icon Newsletter:* $.50 each for single issues (specify numbers). $6.00 for a complete set (Nos. 1-27). There is no charge for overseas shipment of single back issues, but there is a $5.00 shipping charge for the complete set.

Order Form

Icon Project • Department of Computer Science • Gould-Simpson Building • The University of Arizona • Tucson, AZ 85721 USA

Ordering information: (602) 621-2018

name _____

address _____

_____

city _____ state _____ zipcode _____

(country) _____ telephone _____

☐ check if this is a new address

qty.	code	description	price	total

subtotal	
sales tax (Arizona residents*)	
extra shipping charges	
purchase-order processing	
other charges	
total	

Make checks payable to The University of Arizona

*The sales tax for residents of the city of Tucson is 7%. It is 5% for all other residents of Arizona.