The Icon Newsletter

No. 40 – December 21, 1992

Contents

Reflections

The first *Icon Newsletter* was published late in 1978.

Looking back over fourteen years of publication, we're struck not only by how long this has been going on, but by all the things that have happened related to Icon.

Here we are at the fortieth issue. When we started we never imagined it would go on this long or that we would have so many enthusiastic users of Icon and subscribers to this *Newsletter*.

There are still a lot of things going on and we're not about to run out of material.

For the immediate future, at least, we plan to publish this *Newsletter* three times a year (although you'll notice we came perilously close to not getting this one into the 1992 schedule).

And, although the Icon Project is a bit strapped financially, we expect to be able to continue to provide this *Newsletter* without charge.

New Implementations

We've recently brought several more implementations of Icon up to the current version, which is 8.8. Version 8.8 has essentially the same functionality as 8.7, but it includes a few changes to the implementation (we use minor version numbers to enable us to track implementation changes as well as minor changes to language features).

Version 8.7/8 implementations are now available for Macintosh/MPW, MS-DOS, MS-DOS 386/486, OS/2, UNIX, and VMS.

The OS/2, UNIX, and VMS implementations include the X-Icon graphic capabilities.

The UNIX and VMS implementations include the optimizing compiler for Icon, as does the source-code distribution for MS-DOS. (Persons who subscribe to the source-code updates for Icon already have the source code for the compiler.)

Macintosh/MPW Source Code

The Macintosh/MPW source code includes both the interpreter and optimizing compiler, but no attempt has yet been made to build the compiler under MPW.

MS-DOS and OS/2 Source Code

The MS-DOS source code includes both the interpreter and the optimizing complier for Icon.

Building and running the compiler for Icon under MS-DOS requires a 32-bit (386 or 486) platform, a 32-bit C compiler, and at least 4MB of RAM. The presently supported C compilers are Intel Code Builder, Watcom C/386, and Zortech C++ .

The Icon compiler requires significant resources; a fast 486 with considerably more RAM is recommended for running the Icon compiler.

The source code includes configuration information for OS/2, but no attempt has yet been made to build the optimizing compiler under OS/2.

VMS

As noted above, VMS Icon includes the X-Icon graphic capabilities. The distributed executable files require the DECWindows library even if the graphic capabilities are not used. This library is included with the VMS operating system, although it may not be installed if X is not used at your site.

One possibility is to have the library installed even if it's not used. Alternatively, the Icon source code included with the distribution can be configured without X-Icon and built so as not to require the library.

Should You Update?

Persons who have old versions of Icon may wonder whether they should bother to update to the new ones. The answer depends on what version you have and what your interests are.

Certainly anyone using Icon who has a version of Icon prior to 8.0 should update to the latest version. Version 8.0 added many new features and substantially improved the implementation.

If you have an interest in graphics and the current version of Icon for your platform supports X-Icon facilities, you should update if you haven't already.

For other persons with earlier Version 8 implementations, the decision is more problematical. Version 8.5 added several new functions and keywords. Version 8.7/8 automatically expands tables used by the Icon translator, eliminating those annoying command-line options that formerly were necessary for large programs. And several bugs have been fixed.

The next version of the Icon program library, scheduled for release early next year incudes programs that require features of Version 8.7/8.

Also, we cannot provide help with problems that may come up in obsolete versions of Icon.

If you're like us, you may just wish to have the latest software release. At least with Icon, it's not a major expense.

What's in the Works?

Several implementations of Icon still are at Version 8.0. We'd like to get these up to the current version. This may not be easy. From an implementor's point of view, going from Version 8.0 to 8.8 involves a lot of work, including platform-specific changes.

In addition, we can't do any of these implementations locally and therefore have to rely on the volunteer efforts of others.

At the moment, work is under way for CMS and MVS, but as far as we know, nothing serious is being done about the Amiga and Atari implementations.

You'll notice from the preceding sections that the optimizing compiler presently is available only for UNIX, VMS, and for persons who want to build their own under MS-DOS. In principle, implementing the compiler is not much more difficult than implementing the interpreter, but it involves dealing with several technical problems and requires considerable computational resources. We expect that the compiler will be implemented for the Macintosh/MPW and OS/2 platforms shortly after users of those platforms get the source code.

Using the compiler is complicated by the fact that it produces C code, so a person using the Icon compiler needs a C compiler as well — in fact, the same C compiler that was used to build the Icon compiler.

We have the Icon compiler running under MS-DOS for the 32-bit C compilers mentioned earlier. We plan to distribute it, although we're not sure there are enough persons with these C compilers to make it worthwhile. We'd appreciate hearing from those of you who have an interest in this. If you have a 32-bit MS-DOS compiler that is different from those listed earlier, we're interested in knowing that also.

HOPL-II

The second History of Programming Languages conference (HOPL-II) will be held in Cambridge, Massachusetts on April 20-23, 1993.

We're proud to announce that a paper on the history of Icon is among the 14 papers to be presented at the conference.

Those of you who attended the first History of Programming Languages conference in 1978 will recall how interesting it was. It's the only conference in our experience where the attendance at successive sessions increased rather than decreased.

We hope the second conference will be equally interesting and that we'll see some of you there.

Information about the conference can be obtained from:

Dan Halbert
DEC Cambridge Research Laboratory
One Kendall Square, Building 700
Cambridge, MA 02139

617-621-6616, voice

617-621-6650, fax

hopl@crl.dec.com

Icon Graphics Now Available Electronically

At the request of some of our "fans", we've started to place some Icon-related images on our FTP and RBBS facilities so that they can be downloaded electronically.

We haven't decided yet just how to organize the images. If you're using FTP,

```
cd /icon/images
get READ.ME
```

for guidance. The RBBS organization probably will mirror the FTP one.

You'll find things like the Icon "Rubik's Cube" and the auto-stereogram from *Newsletter 39*, the Icon logo, and so forth.

We're using the popular CompuServe GIF format for most images, but some are in Encapsulated PostScript format.

If there are images from past Icon publications that you'd like to have, let us know and we'll see what we can do.

TEXT Technology

Starting with the January, 1993, issue, the journal *TEXT Technology* will move its home base to Dakota State University.

TEXT Technology publishes articles and reviews about all facets of using computers for the creation, processing, and analysis of texts. It is designed for academic and corporate writers, editors, and teachers. The bi-monthly journal contains timely reviews of software for writing and publishing, discussions of applications for the analysis of literary works and other texts, notices of significant events in computing around the world, bibliographic citations, and much more.

Submissions of articles and reviews are welcome. They should be sent as ASCII files via electronic mail to the Editor, Eric Johnson, at eric@sdnet.bitnet. They also may be submitted on MS-DOS diskettes and sent to the address below.

Subscription rates for one year (six bi-monthly issues of sixteen pages each) are $20.00 for the U.S., $27.00 for Canada, and $35.00 for other countries (all prices are in U.S. funds). Subscriptions (which may be charged to MasterCard or Visa) should be sent to:

TEXT Technology
114 Beadle Hall
Dakota State University
Madison, SD 57042-1799
U.S.A.

Downloading Icon Material

Most implementations of Icon are available for downloading electronically:

BBS: (602) 621-2283

FTP: cs.arizona.edu (cd /icon)

ICEBOL6

The Sixth International Conference on Symbolic and Logical Computing was held on October 15 - 16, 1992.

The conference was devoted to study of non-numeric computing, especially for applications in the humanities. In addition to a focus on Icon and SNOBOL, papers described applications written in Prolog, Lisp, C, and Scheme, as well as algorithms for non-numeric processing that might be coded in almost any language.

Proceedings of the conference may be ordered from Dakota State University. The volume contains seventeen papers, 258 pages, by participants from countries on four continents. The cost is $40.00 (which includes postage).

Proceedings for three previous conferences are available. The cost of the Proceedings for ICEBOL2 (131 pages) is $18.00. The only copies of the proceedings for ICEBOL4 (390 pages) that remain have flaws in the binding; they are being sold for $20.00. The cost of the proceedings for ICEBOL5 (331 pages) is $35.00.

All prices are in U.S. dollars, and they include the cost of postage.

To order, send a check for the proper amount to:

ICEBOL PROCEEDINGS Orders
114 Beadle Hall
Dakota State University
Madison, SD 57042-1799
U.S.A.

The supply of some of the proceedings is limited.

Acknowledgments

Icon has benefited greatly over the years from the contributions of interested persons throughout the world. In fact, all the implementations of Icon except for the UNIX one initially were done as volunteer efforts by persons outside the Icon Project. And some features, including large-integer arithmetic, were initially done elsewhere.

We don't mention persons working for the Icon Project, since their contributions are part of their jobs, although you'll find most of their names on various documents. Two recent contributions by persons outside the Icon Project deserve special mention. Bob Alexander updated Macintosh/MPW Icon to Version 8.8 and Darren Merrill not only updated OS/2 Icon to Version 8.8 but also added X-Icon graphics capabilities.

We'd also like to thank the many others who have contributed to Icon in other ways, ranging from suggesting new features to providing help for other Icon programmers to contributing to the Icon program library.

It sounds trite, but it's certainly true that Icon would not be what it is without the contributions of literally hundreds of persons over the last fourteen years.

The Icon Newsletter

Madge T. Griswold and Ralph E. Griswold
Editors

The Icon Newsletter is published three times a year, at no cost to subscribers. To subscribe, contact

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

voice: (602) 621-8448

fax: (602) 621-4246

Electronic mail may be sent to:

icon-project@cs.arizona.edu

or

...uunet!arizona!icon-project

THE UNIVERSITY OF
ARIZONA
TUCSON ARIZONA

and

The Bright Forest Company
Tucson Arizona

Icon Class Projects

In the last *Newsletter*, we showed examples of student projects written in X-Icon for a course on string and list processing. Several readers have asked us to show more of the projects. Here are three.

Mohammad Basel Al-Masri wrote the grade-book program shown in Figure 1. It operates like a spreadsheet, with facilities for adding and deleting rows and columns, entering text in a cell, computing averages automatically, and so forth. Other facilities include generating histograms, summary listings, and so forth. (If the names in Figure 1 look a bit Martian to you, its because we scrambled them to protect the identity of the students.)

Song Liang wrote the tool for examining Icon data that is shown in Figure 2 on the next page. Calls to visualization tools are added to the Icon program to be examined. For example, executing Show_structure(x) produces a window displaying the value of x. Values like lists and strings are shown in boxes; clicking on a box brings up a window showing more detail for the value. Scrolling is provided for long strings and structures that have many elements.

In a lighter vein, K'vin De Vries wrote an adventure game in the classical style. See Figure 3 on the next page. The player can move from room to room in the "space" of the game, ask about the rooms' contents, pick up and drop objects, and so on. The program is self-descriptive to a point, but part of the game is to discover the rules. K'vin remarked that the game is intended to be frustrating. We can attest to that.

You might wonder how big these programs are. It's hard to measure program size with differences in code density and the extent of comments. For what it's worth, the grade-book application is 1,200 lines, the data examination tool is 1,576 lines, and the adventure game is 2,057 lines. For the projects shown in the last *Newsletter*, the maze game is 492 lines, the X-Icon interface builder is 2,682 lines, and the object-oriented drawing program is 1,903 lines.

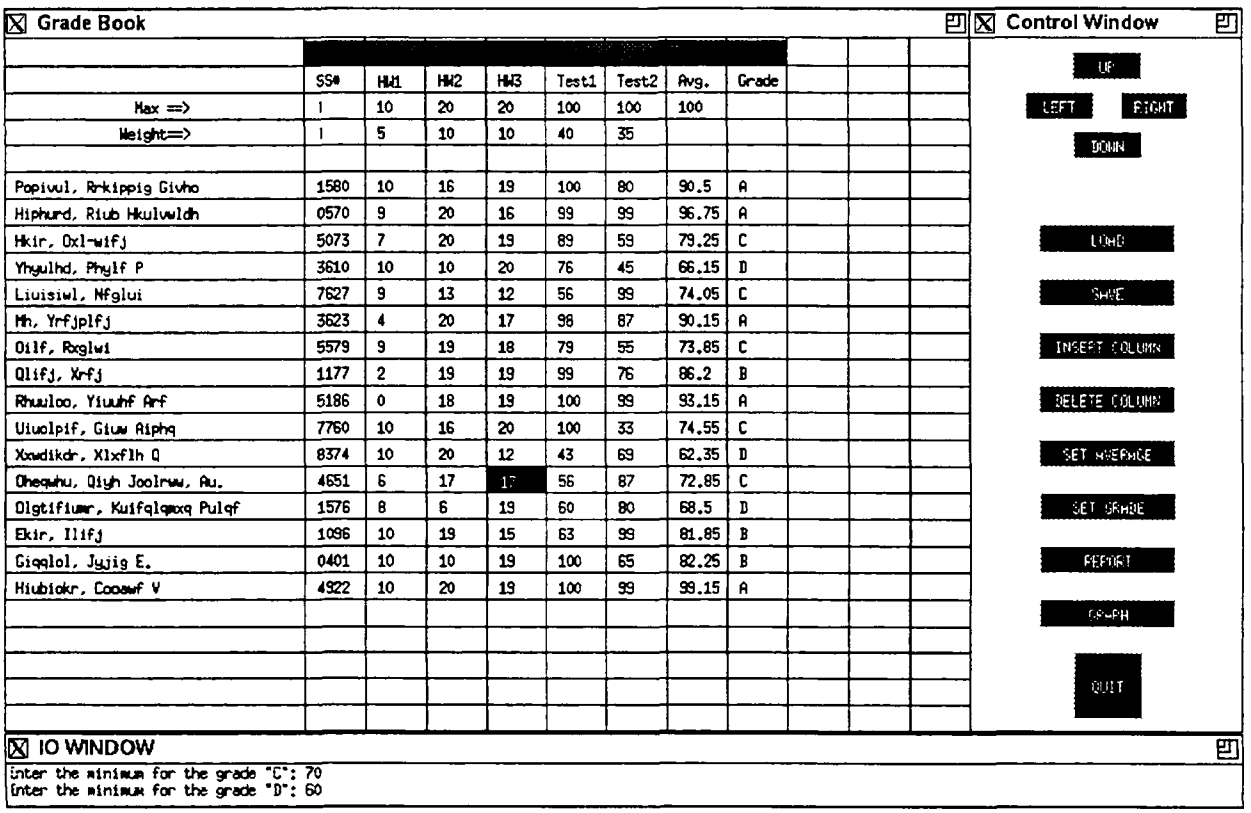


Figure 1 — Grade-Book Application

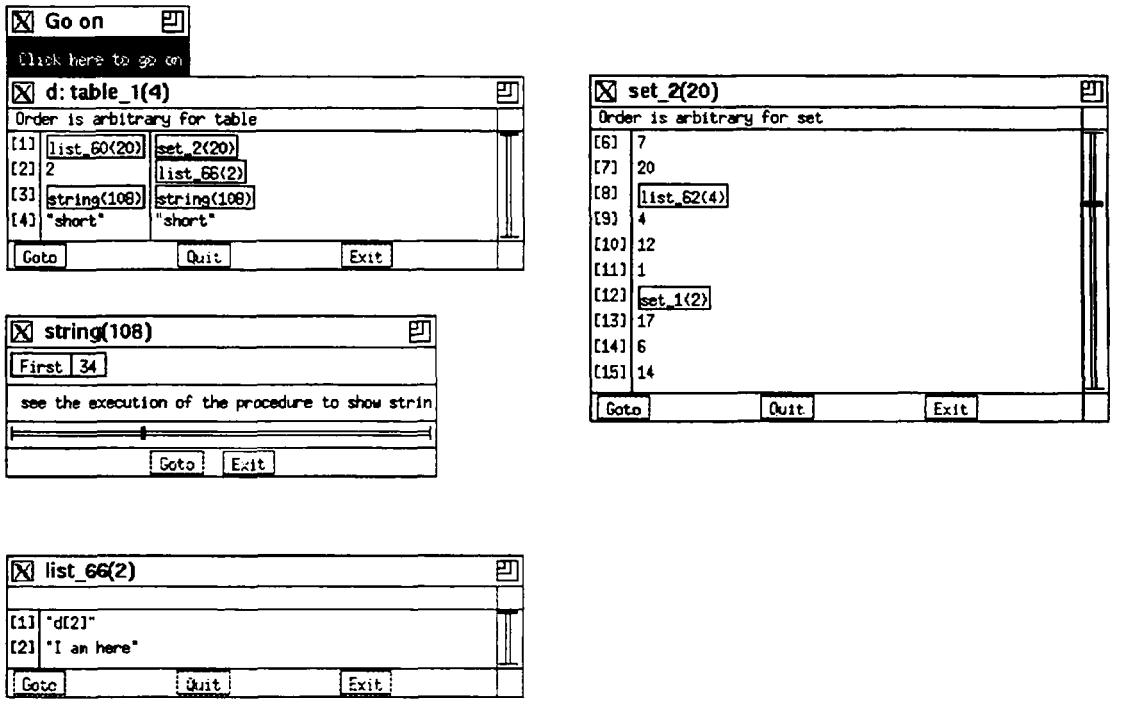


Figure 2— Data Examination Tool

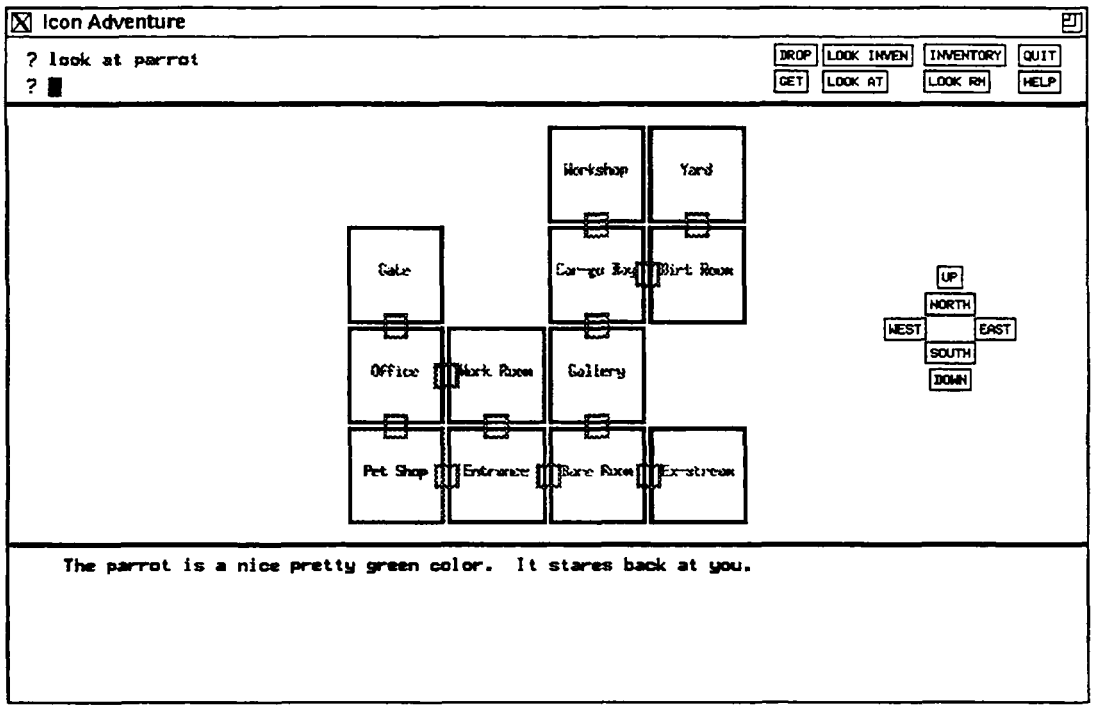
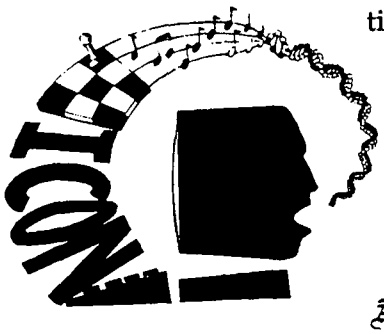


Figure 3 — Adventure Game

Programming Corner



Responses to the questionnaire in *Newsletter 38* indicated considerable interest in articles on simple programming techniques for Icon novices.

While the Icon Analyst is designed to cover matters concerning programming, we'll include some short programming examples in this and future issues of the *Newsletter.*

This article concerns the use of sets. Sets, in the generality found in Icon, are rare in programming languages, and persons who are used to programming in a language like Pascal or C tend to overlook the usefulness of sets in Icon.

The idea of a set is very simple: A set is simply a collection of distinct values. The word distinct is significant; a value can appear only once in a set, as opposed to a list, which can contain many instances of the same value. And, unlike lists, sets are unordered. Consequently, sets embody the concept of membership and are useful for holding values that share a common property, such as a set of all the (distinct) words in a file.

The nice thing about sets is that they are easy to use. A set starts out empty, with no values. Values can be inserted into a set or deleted from a set without having to worry about the space they occupy; sets grow and shrink automatically.

To see how easy it is to use sets, suppose you have a procedure genword() that generates successive words from the input file. If you're willing to accept the definition of a word as a string of consecutive letters, something as simple as the following will do:

```
procedure genword()
  while line := read() do
    line ? {
      while tab(upto(&letters)) do
        suspend(tab(many(&letters))) \ 1
      }
end
```

See pages 88-89 of the second edition of *The Icon Programming Language* for an explanation if this kind of string scanning is unfamiliar to you.

Most text files contain multiple instances of the same words. To find all the distinct words, it's necessary to keep track of the words that have occurred. The distinct nature of values in sets takes care of this automatically. The way a value x is added to a set S is by

```
insert(S, x)
```

If x is not in S, it is added to S. If x already is in S, it is not added. It's that simple.

So to create a set of the distinct words produced by genword(), all that's needed is

```
words := set()          # start empty
every insert(words, genword())
```

The every control structure keeps resuming genword(), causing it to produce all the words in the input file. Each new word is added to words, while words that have occurred before are ignored. The end result is that words contains all the distinct words in the input file.

The resulting set could be used in many ways. For example, to write a list of all the distinct words, all that's necessary is

```
every write(!words)
```

The element-generation operator ! generates the values of words and every causes all of them to be produced.

That's fine if you don't care about the order in which the words are listed. Since a set is (conceptually) unordered, the words may come out in any order. In theory, the last word generated by genword() might be the first one to be listed. Of course, there's some order to the set inside the computer; it's just that it's not one you can predict or that is useful.

It's simple enough, however, to list the words in alphabetical order. Sorting a set produces a list with the set values in order. In the case of strings, this is alphabetical order.

The element-generation operator works on lists as well as sets, but for lists it starts at the beginning and progresses to the end. Consequently,

```
every write(!sort(words))
```

lists the words in alphabetical order.

We've taken a bit of a shortcut here. The function sort() produces a list and we've applied the element-generation operation to this list without bothering to save the list. We could have written

```
wordlist := sort(words)
every write(!wordlist)
```

In fact, that would have been the thing to do if we needed the sorted list after listing its contents.

There's another order for listing the distinct words that you might want: in order of first occurrence. This is just a little more difficult.

What's necessary is to know when a new word is being added to words. The function insert(S, x) doesn't indicate this, but there's another function, member(S, x), that does. It succeeds if x is in S but fails if it is not.

To get the first-occurrence order of listing, this will do:

```
every word := genword() do {
  if not member(words, word) then {
    write(word)
    insert(words, word)
    }
  }
```

Thus, if a word is not in words, it is written and inserted into words.

If you need to keep a list of words in order of first occurrence, rather than just list them as they occur, you can build a list, using Icon's capability to put values on to the end of a list:

```
wordlist := []          # start empty
every word := genword() do {
  if not member(words, word) then {
    put(wordlist, word)
    insert(words, word)
    }
  }
```

When this is complete, wordlist contains all the distinct words in order of first occurrence. A listing then can be produced by

```
every write(!wordlist)
```

or wordlist can be used in any other way that is needed.

There are many other things you can do with sets. Some are simple, like forming the union and intersection of two sets. For example,

```
all_words := words1 ++ words2
```

creates a new set, all_words, that contains all the words in words1 and words2. Similarly,

```
common_words := words1 ** words2
```

creates a new set, common_words, that contains only the words that are in both words1 and words2.

In our examples, we've used words and hence all the values in the sets are strings. Sets are more versatile than this. Their values can be anything — strings, integers, real numbers, even structures.

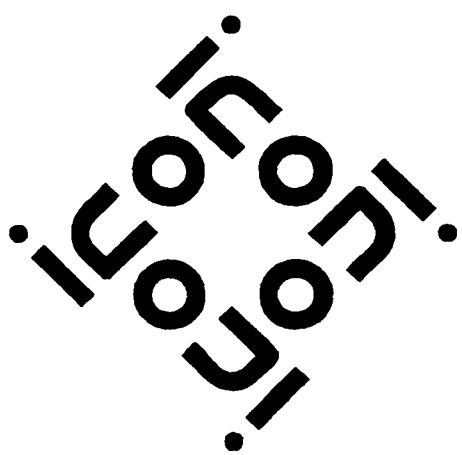
And a set can contain values of different types. This allows sets to be used for purposes far afield from keeping track of distinct words. But using them in such ways is more complicated and we're trying to restrict programming material in the *Newsletter* to things that a novice Icon programmer can easily use. If you're interested in other possible uses of sets, see pages 195-197 of *The Icon Programming Language.*

Icon on the NEC 9801 PC

Gotoo Hitosi has informed us that MS-DOS Icon as distributed by the Icon Project does not run on the NEC 9801 personal computer, even though the 9801 runs MS-DOS. The reason is that the 9801 is not a PC clone, but has a totally different architecture (including BIOS).

He suggests two solutions to the problem: (1) use it with the SIM TSR, which is distributed by major communications networks, such as PC-VAN, or (2) recompile Icon from the source code using a C compiler for the 9801.

Note: The MS-DOS executables distributed by the Icon Project *do* work on PC clones. In fact, the NEC 9801 is the only MS-DOS machine we know of on which our executables do not work.



Ordering Icon Material

What's Available

There are implementations of Icon for several personal computers, as well as for CMS, MVS, UNIX, and VMS. *Note:* Icon for personal computers requires at least 640KB of RAM; it requires more on some systems. Source code for most implementations is available.

There also is a program library that contains a large collection of Icon programs and procedures, as well as an object-oriented version of Icon that is written in Icon.

Icon Program Material

Icon programs provided by the Icon Project are in the public domain.

All program material is accompanied by documentation in printed and machine-readable form that describes how to install and use Icon. This documentation does not, however, describe the Icon programming language in detail. A book is available separately.

Personal Computers: Executable files and source codes are provided in separate packages. Source code for MS-DOS includes the Icon optimizing compiler, configurations for several C compilers, and also OS/2. *Note:* Personal computer distributions are stored in compressed format, and most diskettes are nearly full. It therefore is necessary to have a second drive to extract the material.

CMS and MVS: The CMS and MVS packages contain executable files, source code, test programs, and the Icon program library.

UNIX: The UNIX package contains source code (but not executable files), test programs, related software, and the Icon program library. UNIX Icon can be configured for most UNIX platforms.

VMS: The VMS package contains executable files, source code, test programs, and the Icon program library.

Update Subscriptions: Updates to the Icon source code and the Icon program library are available by subscription.

Source-code updates are distributed on MS-DOS diskettes in LHarc format, and are suitable for compilation under MS-DOS and OS/2 or for porting to new computers. Each update normally provides a completely new copy of the source. A source-code subscription provides five updates. Updates are issued about three times a year.

Icon program library updates are available for MS-DOS, the Macintosh, and UNIX. A library subscription provides four updates. Updates are issued about four times a year.

Documentation

In addition to the installation guides and users' manuals included with the program packages, there are three books on Icon. One contains a complete description of the language, another describes the implementation of Icon in detail, and a third is an introductory text designed primarily for programmers in the Humanities.

There are two newsletters. *The Icon Newsletter* contains news articles, reports from readers, information of topical interest, and so forth. It is free and is sent automatically to anyone who places an order for Icon material. There is a nominal charge for back issues of the *Newsletter*.

The Icon Analyst contains material of a more technical nature, including in-depth articles on programming in Icon. There is a subscription charge for the Analyst.

Payment

Payment should accompany orders and be made by check, money order, or credit card (Visa, MasterCard, or Discover). The minimum credit card order is $15. Remittance must be in U.S. dollars, payable to The University of Arizona, and drawn on a bank with a branch in the United States. Organizations that are unable to pre-pay orders may send purchase orders, subject to approval, but there is a $5 charge for processing such orders.

Prices

The prices quoted here are good until February 28, 1993. After that, prices are subject to change without further notice. Contact the Icon Project for more current pricing information.

Versions

Version information is shown in parentheses. The symbol ↤ identifies recently released material.

Ordering Instructions

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

- 9-track magnetic tape
- data cartridge
- 5.25" diskette
- 3.5" diskette

Tapes are written at 1600 bpi. Cartridges are written in QIC-24 format. 5.25" diskettes are 360K. 3.5" diskettes are 720/800K unless otherwise noted.

Diskettes are written in MS-DOS format except for the Amiga, the Atari ST, and the Macintosh. When ordering diskettes that are available in more than one size, specify the size (the default is shown first).

In some cases, there are several diskettes in a distribution.

Shipping Charges: The prices listed include handling and shipping by parcel post in the United States, Canada, and Mexico. Shipment to other countries is made by air mail only, for which there are additional charges as noted in brackets following the prices. For example, the notation $15 [$5] means the item costs $15 and there is a $5 shipping charge to countries other than the United States, Canada, and Mexico. UPS and express delivery are available at cost upon request.

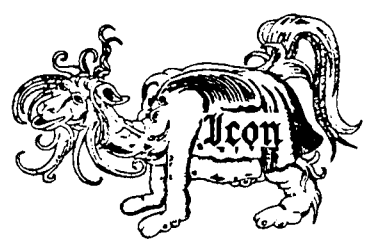
Ordering Codes: When filling out the order form, use the codes given in the second column of the list to the right (for example, DE, SU, ...).

Executables				
Acorn Archimedes (8.0)	ARE		$15	[$5]
Amiga (8.0)	AME		$15	[$5]
Atari ST (8.0)	ATE	[1]	$15	[$5]
MS-DOS (8.8)	DE	↤	$15	[$5]
MS-DOS 386/486 (8.8)	DE-386	↤	$15	[$5]
Macintosh (8.0)	MET		$15	[$5]
Macintosh/MPW (8.8)	MEM	↤	$15	[$5]
OS/2 (8.8)	OE	↤	$15	[$5]
Source				
Amiga (8.0)	AMS		$15	[$5]
Atari ST (8.0)	ATS		$15	[$5]
MS-DOS & OS/2 (8.8)	DS	↤	$30	[$5]
Macintosh (8.0)	MST		$15	[$5]
Macintosh/MPW (8.8)	MSM	↤	$25	[$5]
MS-DOS updates (5)	SU		$60	[$15]
Complete Systems				
CMS (8.0)	CT		$30	[$10]
MVS (8.0)	MT		$30	[$10]
UNIX (8.7)	UD	[2]	$25	[$5]
UNIX (8.7)	UT		$30	[$10]
UNIX (8.7)	UC		$45	[$10]
VMS (8.7)	VT	↤	$32	[$11]
Program Library				
MS-DOS	DL		$15	[$5]
Macintosh	ML		$15	[$5]
UNIX	UL		$15	[$5]
Macintosh updates (4)	LU-M		$30	[$12]
MS-DOS updates (4)	LU-D		$30	[$12]
UNIX updates (4)	LU-U	[2]	$30	[$12]
Books				
The Icon Programming Language	LB		$40	[$13]
The Implementation of Icon + update	IB		$53	[$14]
Icon Programming for Humanists + diskette	HB		$36	[$10]
Newsletters				
The Icon Newsletter (complete, 1-39)	INC		$18	[$5]
The Icon Newsletter (back issues, each)	INS		$1	[$2[3]]
The Icon Analyst (1 year, 6 issues)	IA		$25	[$10]
The Icon Analyst (back issues, each)	IAS		$5	[$2[3]]

[1] 400K.

[2] 1.44M.

[3] Per order, regardless of the number of issues purchased.



Order Form

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

Ordering information: (602) 621-8448 • Fax: (602) 621-4246

name _____

address _____

_____

city _____ state _____ zipcode _____

(country) _____ telephone _____

❒ check if this is a new address

qty.	code	description	price	shipping*	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.

❒ Visa ❒ MasterCard ❒ Discover ❒ check or money order

I hereby authorize the billing of the above order to my credit card: ($15 minimum)

card number | | | | | | | | | | | | | | | |

exp. date | | | | |

name on card (please print) _____

signature _____

*Shipping charges apply only to addresses outside the United States, Canada, and Mexico

COCO