

# The Icon Newsletter

No. 29 – February 14, 1989



## Implementation News

### Icon for OS/2

We now have an OS/2 version of Icon.

It is a large-memory-model implementation, which limits region sizes to slightly less than 65K. (We're working on a huge-memory-model implementation that will remove this restriction.) OS/2 Icon is available on either 3.5" or 5.25" diskettes. See the order form at the end of this *Newsletter*.

Cheyenne Wills developed this implementation; our thanks to him.

### Implementation Updates

We are continuing to update implementations of Icon from Version 7.0 to 7.5. The latest updates are for UNIX, VMS, and the source for porting. See the order form at the end of this *Newsletter*.

As mentioned in the last *Newsletter*, the source-language differences between 7.0 and 7.5 are minor, and there is no reason to upgrade if you're using Version 7.0. (If you're using an earlier version, you definitely should upgrade.) However, we no longer support Version 7.0 for implementations that have been upgraded to 7.5, so if you run into a problem, you may need to upgrade.

There are a lot of internal differences in the implementation between Version 7.0 and 7.5, and if you're working with the source code, you should upgrade.

Since we only charge for the costs of producing and distributing Icon material, we don't offer special prices to persons who upgrade from an earlier version. However, we've been distributing Version 7.5 for the systems mentioned above since early January, so if you've obtained a copy recently, you should already have Version 7.5.

## Letter from an "Old Icon Hand"

*Editors' Note: Shortly after Icon Newsletter 28 was distributed, Ralph Griswold received this letter from Walt Hansen, who participated in the early development of Icon.*

I just got a copy of the latest *Icon Newsletter*. Under "Odds and Ends" you wonder why the UNIX implementation took so long to get started. I think I remember. Maybe it's easier for me because my participation in the project ended just as the UNIX implementation was getting off the ground.

I recall the department got the PDP 11/70 in December 1977. I distinctly remember playing with it for the first time following the Christmas party at Dick Orgass's house. Only the DECwriter II console was functioning. It was a hell of neat new toy.

I joined the Icon project in January 1978.

The two implementations I worked on were the ones done in Ratfor on the DECSys-10. The first implementation was done by Dave Hanson, Tim Korb and me. I did the front end, Tim & Dave the runtime. You were doing documentation and QA. Steve Wampler participated in the second implementation. He was doing the port to the CDC machine(s) — the computer center was transitioning from the 6400 to the Cyber 176 (?) about this time. Cary Coutant came on board toward the end of the second implementation. I don't recall if he had any implementation responsibilities for this release.

The first implementation was strictly an interpreter. The details seem vague now, but I recall the entire runtime was always present. It was a pig: fat and slow. We had this working pretty quickly. But the

Icon!  
Icon!  
Icon!

performance penalty prompted us to try a new approach. This led to the second implementation which took up the whole summer of '78. Our target was to have a working system by the start of the Fall term for a class you were teaching.

The second implementation was a "compiled" interpreter. The front end generated FORTRAN code including function calls to the runtime. The generated mess was then compiled and linked. It was a lot faster. We got it built on time and the class liked using it. Of course they also found a lot of bugs. Fixing bugs and adding features took up the remainder of my participation through the next year. I left in August 1979.

## 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,allegro,noao)!arizona!icon-project

© 1989 by Madge T. Griswold and Ralph E. Griswold

All rights reserved.

We weren't using UNIX for a number of reasons. First, we already had the momentum of the first implementation and the goal of building a working interpreter by the start of the class. Second, it was just too new. None of us (with the exception of Dave) had used UNIX before. Third, we were all very familiar with the DECSys-10. Why did it take a couple of years to get the first UNIX implementation started? Momentum.

Now I have a couple of recollections of your opinions from that time that may have influenced events. I don't think you were sold on UNIX as god's gift to programmers and I think you had severe reservations about C. I don't think you liked the language. Of course C wasn't available outside UNIX in those days so portability was out. The FORTRAN on UNIX was F77. We were assuming a Ratfor targeted to FORTRAN 66. Maybe that had some influence; I recall Steve having conniptions converting to FORTRAN 77 on the Cyber. One last thing. I suggested using YACC in the front end to generate a Ratfor parser. Your experience with SL5 using a tempermental parser generator (whose name eludes me now) that lacked portability discouraged the use of YACC at this time. It seems funny now to think that back then UNIX and UNIX tools were considered hinderances to portability.

I hope this helps. Maybe it will only muddy the waters. Whatever. I enjoyed my time on those projects. I have fond memories of working with you, Tim, and Dave on that first implementation. The give and take in the meetings and commeraderie were unique in my experience. It really spoiled me.

Walt

P.S. Icon is now over ten years old. It's about the same age SNOBOL4 was when Icon was born (and post SL5). Icon is now fairly stable. Radical changes might screw up a lot of existing code. So, any plans to launch a new research language in the near future?

## Downloading Icon Material

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

BBS: (602) 621-2283

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

## A Contribution from Users (cont'd)

*Editors' Note: This is the conclusion of a contribution from Andy Heron and Carole Thorton that started in the last Newsletter. It describes some modifications the authors made to Icon for a database project.*

### Run-Time Record Definition

New facilities have allowed us to develop Icon procedures to store and retrieve structured data from disk.

Consider the following fragment of Icon code.

```
count := table(0)
every item := get_item() do
  count[item.type] += item.number
```

One can deduce that the procedure `get_item` generates a sequence of records that include the fields "type" and "number". The fragment of code is making a table of the totals for each type.

If `get_item` were reading data from a file that included not only the data but also a description of the record, then the format of the data is not hard-wired in the procedure. The user of `get_item` must have some idea of the data to be read (i.e. the fields "type" and "number") but complete knowledge is not necessary.

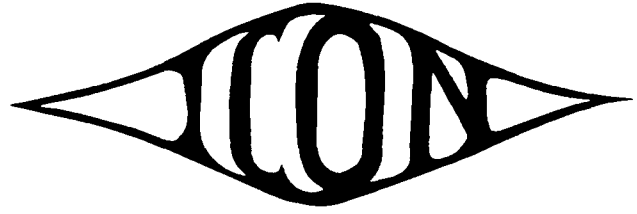
We have used a database to hold our items, which are almost arbitrary structures (no loops) made up of lists, records, strings and integers (reals and csets are not yet implemented). The database items consist of a packed form of the Icon data structure as stored in the memory of the executing Icon program. The items are accessed by key in the database in the following manner.

```
# This procedure adds one to every member of the list
# counts in the database item with key item_key
```

```
record bump_counts_references(counts)
```

```
procedure bump_counts(item_key)
  local item
  item := READ_DB_RECORD(item_key)
  every !(item.counts) += 1
  UPDATE_DB_RECORD(item,item_key)
end
```

The variable `item_key` must be the string key of the database item and then `READ_DB_RECORD` will return the Icon structure that was stored in packed form on the database. This can be used in the normal way as shown above. In our implementation, the formats of items are stored in the database and each database item contains the key of its format, thus avoiding unnecessary repeated storage of format information. These format definitions are called Layouts, and an example of the source form is given below.



```
1 name char(30) # only 30 characters for name
1 address
  2 no int(4) # only 4 digits for house number
  2 street char(30)
  2 city char(20)
1 remarks[*]
  2 comment text # no limit on string recorded
  2 date char(6) # yymmdd
```

The style of this description owes much to our earlier PL/I background, and the example should be self explanatory for the most part. Data using this Layout is a record with three fields: "name", "address" and "remarks". Similarly the "address" field is a record with three fields: "no", "street" and "city". The field "remarks" is an unlimited size list of records (specified by the "\*"). The field "comment" is a text field, which is an unlimited length string. The concept of unlimited size objects is quite natural to the Icon programmer, but we have also used the types `char` and `int`, which are Icon strings and integers respectively and these are checked for size at various points in our system. When formatted output is required, the sizes may be used; and when a string that is described as a `char` field is stored in the database, its length is checked to avoid storing unintentionally long strings on the database (i.e., spot bugs wasting disk space).

Provision is made for modifying a Layout without changing all the items on the database that use this Layout. If an item is stored using an old Layout, then it is converted to the new Layout before it is presented to the application program and, since this conversion is rare, this work is done in Icon. Converting items between Layouts allows small changes, such as adding or deleting fields in a record, but more drastic structural changes will need a different approach.

Database items may be viewed or edited by writing them to a file in External Format, which describes both the data and its format in an easily understood form. For example a record in the earlier Layout would look like:

```
[name P. Pat
[address
[2no 20
[2street Puddle Lane
```

[2city Greendale  
[remarks  
[2comment  
This entry is just an example for later reference.  
[2date 880213

Some items are just lists of simple records (e.g. each field of the record is of type char or int) and then a more compact form is possible, which uses the maximum size of the fields. For example, a list of addresses can be displayed as follows:

```
[address  
  ..no street..... city.....  
    1 High Street      Toytown  
    20 Puddle Lane    Greendale
```

---

## From Our Mail

*Just a compliment from a satisfied reader of the Newsletter.*

*It is a very informative and attractive publication.*

*What tools are you using to produce it? I didn't notice any description (at least in the October 15 issue).*

*Congratulations again, and thanks.*

*And we thank you! While producing this Newsletter is a lot of work, it certainly helps to be appreciated.*

We use a variety of tools in the preparation of the *Icon Newsletter*. Recent issues prior to this one were composed using Xerox Ventura Publisher on an IBM XT clone. Starting with this issue, we're using Aldus PageMaker on a Macintosh II. We use several programs for preparing graphics, most notably Adobe Illustrator '88. We also have an Abaton scanner for converting printed graphics to machine-readable form.

*Do I need OS/2 to run MS-DOS/386 Icon?*

No. In fact, MS-DOS/386 Icon will not run under OS/2. It uses a memory extender product and only runs under standard MS-DOS.

*I need large memory regions for some of my MS-DOS programs. I can get them with the expandable regions version of Icon, but it's much slower than the fixed regions version. Is there any way I can get larger regions with the fixed regions version?*



The fixed regions implementation of Icon for MS-DOS is a large memory model program. This limits regions to slightly less than 65K. While there currently is no way to get around this limitation, we're working on a huge memory model version of the fixed regions implementation. The huge model will allow regions to be as large as available memory will accommodate.

*Will Version 7.5 of Icon compile and run under the MS-DOS MIX Power C compiler?*

We don't know. However, since Version 7.5 of Icon works with Lattice C, Microsoft C, Turbo C, and Let's C, there's a good chance it will work with other high-quality C compilers.

*In Icon Newsletter 26 you noted that Version 6 of Icon was going to be available for Prime computers. I've seen nothing since. Whatever became of this implementation?*

This implementation was being done by a person at Prime. We've never received it and don't know what its current status is.

*I just got my first Icon Newsletter and would like to get more information about Icon. I saw references to IPD60 and 61. Would you please send me a list of all IPDs and tell me how to get them?*

Most IPDs (Icon Project Documents) are installation and user manuals that accompany the various implementations of Icon. These are available only as part of the distribution packages for these implementations. Most other IPDs are out of date and no longer available. The best way to find out more about Icon is to get back issues of the *Icon Newsletter*, where you'll find mention of various documents that are available.

*What's the status of Version 7 of Icon for the Amiga?*

A couple of folks have expressed an interest in implementing Version 7.5 of Icon for the Amiga. We don't have a working version yet, though.

*Any more word on Icon for IBM 370 mainframes?*

It's presently being tested.

*How do I do binary input/output in Icon?*

Icon doesn't provide specific functions for binary input or output. It can be done on most systems using other built-in functions (starting with Version 7), though it is a bit tedious.

The built-in function `reads(f,i)` reads exactly `i` bytes from file `f`, giving no special interpretation to newlines or nulls or anything else. `ord(s)` converts a byte to its integer value, and multibyte integers can be built using `ishift(i,j)` and `ior(i,j)`.

Output can be done similarly, breaking integers into 8-bit values using `ishift(i,j)` and `land(i,16rFF)`, then converting to character using `char(i)`.

`seek(f,i)` is also useful for reading binary files; keep in mind that `l` on numbers the bytes of a file beginning at 1 and not 0.

## Programming Corner

### Table Keys

If `T` is a table, `!T` generates the values in the elements of `T`, not the keys (entry values). This probably was a design mistake, since it's possible to get the values from the keys, but not the other way around. So ...



Problem: Write a procedure `key(T)` that generates the keys from table `T`.

Comment: The only way to do this is to convert `T` to a list.

#### Approach 1: Create a list of key/value lists:

```
procedure key(T)
  L := sort(T)
  every pair := !L do
    suspend pair[1]
  end
```

Whenever a `suspend` appears in the `do` clause of an `every` expression, look to see if both are needed, since `suspend` also forces its argument to generate all its results. Thus,

```
procedure key(T)
  L := sort(T)
  suspend (!L)[1]
end
```

Note the parentheses: `!L[1]` groups as `!(L[1])`, not what you want.

One more refinement: get rid of the identifier `L` and just put the call of `sort` in the `suspend` expression:

```
procedure key(T)
  suspend (!sort(T))[1]
end
```

#### Approach 2: Create a list of alternate key/value pairs:

```
procedure key(T)
  L := sort(T,3)
  while x := get(L) do { # get key
    suspend x
    get(L) # discard value
  }
end
```

Observation: The `while/suspend` construction suggests a more compact technique as in `every/suspend`. Use the "make a generator from a non-generator" paradigm:

```
procedure key(T)
  L := sort(T,3)
  every x := |get(L) do { # get key
    suspend x
    get(L) # discard value
  }
end
```

Now the `every/suspend` can be collapsed as usual:

```
procedure key(T)
  L := sort(T,3)
  suspend |get(L) do
    get(L)
  end
```

Note the use of the optional `do` clause for `suspend`.

This could also be written more compactly (but opaquely) as

```
procedure key(T)
  L := sort(T,3)
  suspend |1(get(L),get(L))
end
```

However, it's not possible to get rid of the identifier `L`, since the list must be an argument of two separate function calls.

Which approach is best? At the bottom line, both are obscure and make hard reading. The first approach produces the most compact code and with no auxiliary identifier. However, the list of lists produced by the default option for sorting takes up a lot of room, which may be a practical consideration.

Note also that both of these methods do something more than generate the keys – they generate them in sorted order, which is different from `!T`.

### Unique Values in a List

Here's another problem: Write a procedure `uniquelem(L)` that returns a list containing only the distinct values of `L` – that is, filtering out duplicates.

*Approach 1: If order doesn't matter, there's a very simple way:*

```
procedure uniquelem(L)
  return sort(set(L))
end
```

This takes advantage of the fact that distinct values can be members of a set only once.

This procedure, in fact, produces the elements in sorted order (since the only way to convert a set to a list is to sort it).

Although simple, this method requires both constructing a set that is discarded and also sorting it (perhaps unnecessarily).

*Approach 2: If you want to preserve the order of the elements of the list, you have to do something like this:*

```
procedure uniquelem(L)
  local L1, x
  L1 := []
  every x := !L do
    if x == !L1 then next else put(L1,x)
  return L1
end
```

Of course, this can be very slow if L is large and has mostly distinct elements.

Note that it won't do to rephrase the test and addition of an element as

```
if x ~== !L1 then put(L1,x)
```

since the test succeeds if x is not the same as (say) the first element of L1 but is the same as (say) the second. Be careful of inverting logic in expressions that contain generators.

Also, never do something like

```
while x := get(L) do ...
```

since this destroys the list L, a pointer to which is passed in as an argument and "belongs" to someone else, who may not appreciate having the list trashed.

Can you think of a better method?

## Data Backtracking

We were recently asked the following question: Why are

```
while move(1) do { ... }
and
every |move(1) { ... }
different?
```

The difference has to do with data backtracking. `move(1)` increments `&pos`, but it suspends (like `find(s)`). It does this not so that it can produce another result if it's resumed (there is only one way to increment `&pos` by 1), but instead to restore `&pos` to its previous position (data backtracking). See p. 126 in the Icon book.

On other hand, the control clause of `while-do` is bounded (that particular term is not used in the Icon book, but see p. 119 there). In a bounded expression, suspended generators are discarded, once a result is produced. Thus, in

```
while move(1) do { ... }
```

the change to `&pos` by `move(1)` is irreversible, since `move(1)` won't be resumed.

In `every-do`, on the other hand, the control clause is not bounded – in fact, the control clause must be able to generate a sequence of results for `every-do` to have the effect it does. Thus, in

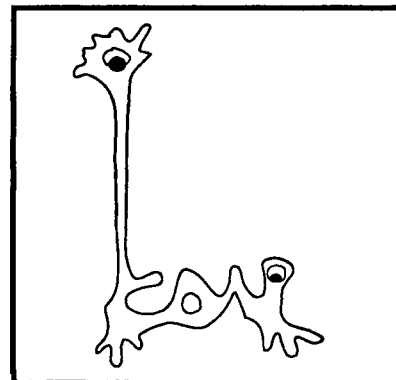
```
every |move(1) do { ... }
```

`move(1)` is resumed after the `do` clause is evaluated. It restores `&pos`, and then repeated alternation causes `move(1)` to be evaluated again, with the same value of `&pos` as before – an endless loop.

The problem in understanding this probably is not in what `while-do` does – most folks find that to be what they expect. It's in `every-do` that the problem with understanding arises. A simple prescription that will do for most cases is "don't use `every-do` in string scanning". That may keep you out of trouble, but a deeper understanding of expression evaluation in Icon will show you both why you usually don't want to use `every-do` in string scanning and also why you sometimes might want to use it.

Quiz: Assuming `s := "Hello world!"`, what does the following expression write?

```
every write(s ? move(1 to 10))
```



## Demography

Subscribers to this *Newsletter* sometimes ask if there are other persons interested in Icon who live nearby. We've wondered ourselves what the geographical distribution of the *Icon Newsletter* is, and so we fed our mailing list database into a program that plots "sites" according to ZIP Code centroids. The result for the United States, except Alaska and Hawaii, is shown below. While you can't pick the individual dots out of the picture, and the boxes with crosses in the center mark multiple entries, it does give an overall impression of the distribution.

We're working on getting a "world view" for a future *Newsletter*.



## What's in the Works

We're often asked what we're doing, what new features are being planned for Icon, and so on.

As always, we have more things we want to do than we can possibly handle. There are "little" things, like bringing all the existing implementations of Icon up to Version 7.5, pushing new implementations (like the one for VM/CMS) out the door, revising the Icon language book, completing the Version 7 Icon program library, and so on.

As to the Icon programming language itself, we don't have any plans for major new features or a new version in the near future. Of course, we could surprise ourselves, as we have in the past.

One present project of importance is the development of a true optimizing compiler for Icon. This is a research project at the moment (and a hard one), but

we hope that eventually it will provide the basis for a high-performance implementation of Icon.

A few things that we've been working on for some time should be available for public distribution within the year. These include tools for measuring the performance and behavior of Icon programs and the Icon implementation.

For example, we've instrumented storage allocation and garbage collection and can produce allocation history files that tell a lot about how Icon programs use memory. We also have programs that use allocation history files to produce graphical displays of Icon's allocation and garbage collection. On color graphic devices, the displays are truly spectacular.

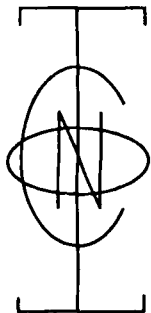
We aren't making any promises yet of what or exactly when, but there are all kinds of things in the works.

## Faculty Positions

The Department of Computer Science at the University of Arizona invites applications for faculty positions at all ranks to begin in August, 1989. Applicants must have a doctorate in Computer Science or a closely related field. Applicants for senior positions should have made substantial research contributions to the field, while applicants for junior positions should show promise of future excellence.

The Department of Computer Science at Arizona emphasizes excellence in research and teaching. There are currently 12 faculty members, with plans to expand over the next few years as the department institutes a selective undergraduate major. Research is currently conducted in a variety of areas including programming languages, software systems, parallel and distributed computing, logic programming, and theory of computation. Qualified individuals working in these areas as well as other areas such as artificial intelligence, computer architecture, scientific computation, and performance analysis are encouraged to apply.

The research program is supported by numerous grants to individual faculty as well as a department-wide NSF grant for Coordinated Experimental Research (CER). Computational facilities include a VAX 8650, dozens of Sun workstations, an Intel iPSC Hypercube, and an HP 9000 graphics workstation. Also



available are high-resolution color terminals, microcomputers, laser printers, and an L-300 Imagesetter. A soon-to-expand instructional laboratory contains two VAX 11/785s.

Send a complete resume and the names of at least three references to Richard D. Schlichting, Faculty Recruiting Committee Chairman, Department of Computer Science, The University of Arizona, Tucson,

AZ 85721. Applications will be reviewed until the positions have been filled. The University of Arizona is an equal opportunity/affirmative action employer.

---

## ICEBOL4 in October

Once again, Dakota State College is hosting the annual ICEBOL conference. This year, it's October 5-6. We've included material from their flyer below.

ICEBOL4, the International Conference on Symbolic and Logical Computing, is designed for teachers, scholars, and programmers who want to meet to exchange ideas about non-numeric computing. In addition to a focus on SNOBOL, SPITBOL, and Icon, ICEBOL4 will feature introductory and technical presentations on other dangerously powerful computer languages such as Prolog and LISP, as well as on applications of BASIC, Pascal, and FORTRAN for processing strings of characters. Topics of discussion will include artificial intelligence, expert systems, desktop publishing, and a wide range of analyses of texts in English and other natural languages. Parallel tracks of concurrent sessions are planned: some for experienced computer users and others for interested novices. Both mainframe and microcomputer applications will be discussed.

For further information, contact:

Eric Johnson  
ICEBOL Director  
114 Beadle Hall  
Dakota State College  
Madison, SD 57402 U.S.A.

(605) 256-5270

eric@sdnet (bitnet)

---

## Art Credits

Graphics that first appeared in earlier *Newsletters* are credited there.

Page 3. Benton Carter, pencil drawing, scanned and autotraced with Illustrator '88.

Page 6. Ralph Griswold, *iconopod*, pencil drawing, scanned and autotraced with Illustrator '88.

Page 7. U.S. map, GeoQuery.

Page 8. Fleming Rembish, pen and ink monogram, scanned and traced with Illustrator '88.

Page 8. Charles Richmond, Atari ST, printed output, scanned and traced with Illustrator '88, converted to a PostScript font with KeyMaster; border entered as text.

Back page. Bob Alexander, *Wallpaper 2*, MacDraw II.

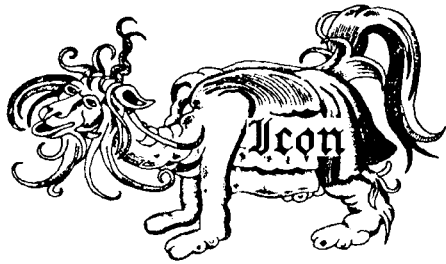




## Ordering Icon Material

**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, subject to approval, 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 yet 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:  (6) cpio format \$40

#### Icon for VMS:

-  VT:  \$25

#### Icon for the Atari ST:

- ATE:  executables \$15

**Icon for MS-DOS:**

DE: [disk icon] (2) executables \$20  
DS: [disk icon] (2) source \$25

**Icon for MS-DOS/386:**

DE-386: [disk icon] or [floppy icon] executables \$15

**Icon for OS/2:**

[hand icon] OE: [disk icon] or [floppy icon] executables \$15

**Icon for the Macintosh/MPW:**

ME: [floppy icon] executables \$15  
MS: [floppy icon] (2) source \$25

**Icon for the UNIX PC:**

UPE: [disk icon] executables \$15

**Icon for XENIX:**

XE: [disk icon] executables \$15

**Icon for XENIX/386:**

XE-386: [disk icon] or [floppy icon] executables \$15

**Icon Source for Porting:**

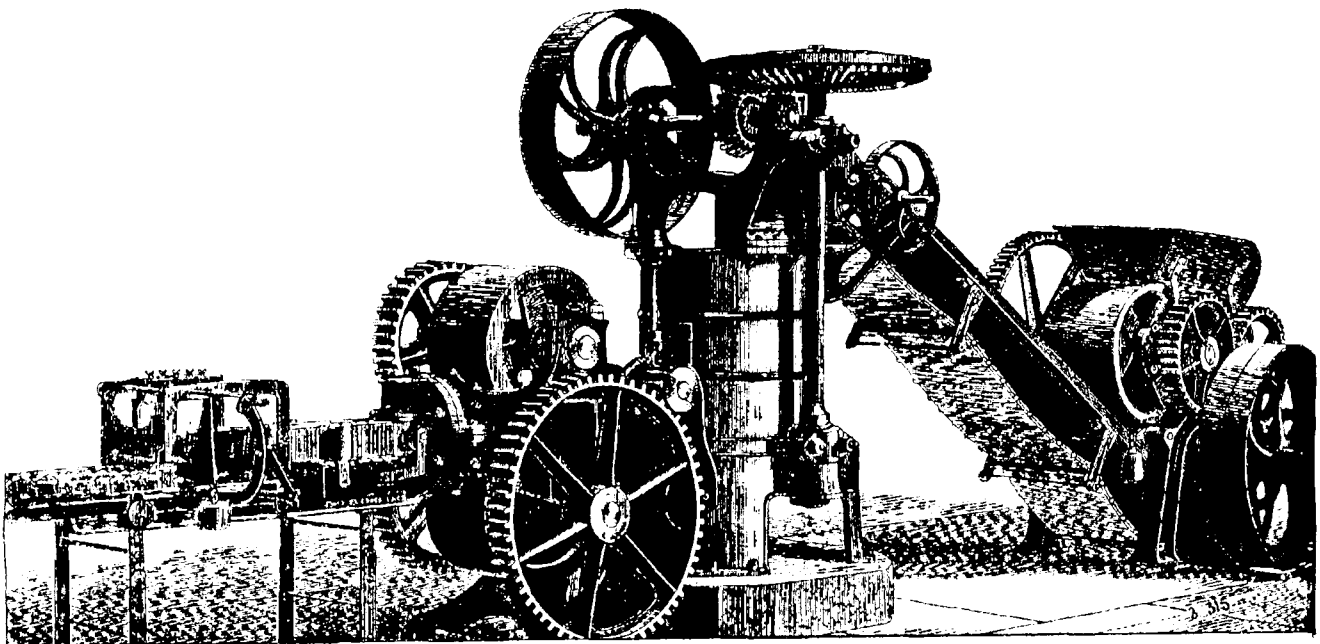
[hand icon] PF-A: [disk icon] (5) flat system, ASCII \$35  
[hand icon] PF-K: [disk icon] (2) flat system, ARC \$25  
[hand icon] PH-A: [disk icon] (5) hierarchical system, ASCII \$35  
[hand icon] PH-K: [disk icon] (2) hierarchical system, ARC \$25

**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. 1-28). 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         |  |
| Make checks payable to The University of Arizona | purchase-order processing      |  |
|  | other charges                  |  |
|  | total                          |  |

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

