

# The Icon Newsletter

No. 31 – September 15, 1989



## Price Increases

If you look over the order form at the end of this *Newsletter*, you may notice a few price increases. We try to keep the prices down, but inflation and increased costs for preparing some distribution material have led to a few adjustments. In addition, the prices of the Icon books have been increased by the publishers. A publisher's representative once told us that they increase their prices regularly in anticipation of inflation. No wonder we *have* inflation! In case you've wondered, authors have no control over the prices that publishers charge for their books.

## Implementation News

### *Icon for the IBM 370*

Icon is now available for computers with IBM 370 architecture. There are implementations for both the MVS and VM/CMS operating systems. See the order form at the end of this *Newsletter*.

In addition to the implementations distributed by the Icon Project, there is another VM/CMS implementation done in Germany that may be more accessible to European users. Contact:

Walter H. Schiller  
Lagesche Straße 32  
D-4790 Paderborn  
West Germany

There also is another MVS implementation of Icon. While it is not being distributed, interested persons may contact the implementor:

Nick Maclaren  
University of Cambridge  
Computer Laboratory

New Museums Site  
Pembroke Street  
Cambridge CB2 3QG  
England

+44 223 334761

[nmml@phx.cam.ac.uk](mailto:nmml@phx.cam.ac.uk)

## *Other Implementation News*

All implementations of Icon distributed by the Icon Project are now up to Version 7.5. In addition, source code for the Amiga implementation is now available.

Icon has been implemented for OS-9. This implementation probably will not be distributed by the Icon Project, but if you are interested, let us know and we'll put you in touch with the implementor.

## *Source Updates for MS-DOS*

The source code for Icon changes frequently as improvements are made and new features are added to the language. Although we only update our distributed versions of Icon infrequently, we do provide a subscription update service for persons who are using Icon code on MS-DOS systems.

This service provides the current version of the source code about three times a year, together with a brief description of changes that have been made and what is in the works. Updates are distributed on 5.25" diskettes only.

If you're using Icon source code for MS-DOS and want to stay on top of things, this is the way to do it. And it's a real bargain: only \$50 for 5 updates, including first-class postage in the United States (add \$15 for air mail overseas). See the order form at the end of this *Newsletter*.

## *A Word of Thanks*

Many of the implementations of Icon are done by persons outside the Icon Project. We greatly appreciate their help.

Recent assistance has been provided by Cheyenne Wills, Robert Knight, and Eric Johnson (VM/CMS); Alan Beale (MVS); Bob Goldberg (MS-DOS/386); and Clint Jeffery (Amiga).

## Geographical Distribution of Newsletter Subscriptions

In *Newsletter 29* we published a map of the United States showing the distribution of subscribers. We've been working on a more global view, but we haven't found a good way to do it yet. In the meantime, here's a list of subscribers by country:

United States	2323
United Kingdom	229
Canada	132
West Germany	110
Australia	73
The Netherlands	71
France	67
Japan	31
Sweden	26
Israel	23
India	22
Finland	20
Italy	16
Brazil	12
Belgium	11
Denmark	11
Poland	10
Spain	10
New Zealand	9
Switzerland	9
Austria	7
Mexico	7
Singapore	7
Czechoslovakia	5
Ireland	5
Norway	5
Republic of South Africa	5
Thailand	5
Korea	4
Hungary	3
Portugal	3
Venezuela	3
Algeria	2
Bulgaria	2
Cuba	2
Cyprus	2
East Germany	2
Fiji	2
Iceland	2
Peoples Republic of China	2
Philippines	2
Saudi Arabia	2
Taiwan	2
Yugoslavia	2
Argentina	1

British West Indies	1
Chile	1
Eastern Caroline Islands	1
Greece	1
Hong Kong	1
Iran	1
Kuwait	1
Luxembourg	1
Malaysia	1
Nigeria	1
Papua New Guinea	1
Peru	1
Romania	1
USSR	1

### *The Icon Newsletter*



Madge T. Griswold and Ralph E. Griswold  
Editors

*The Icon Newsletter* is published aperiodically, usually three times a year, 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) 621-4049

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.

# Improving the Performance of Sets and Tables in Icon

*Editors' Note: This article is contributed by Bill Griswold at the University of Washington.*

## Introduction

Icon's current table and set implementations depend on a hashing scheme with a fixed number of slots (i.e., buckets). For large tables or sets this has a bad impact on performance, since collision-resolution chains become quite long. Dynamic hashing is a technique that dynamically expands or contracts a hash table (i.e., the number or slots it has) in proportion to the number of elements in the table. By assuring that the number of slots and the number of elements is related by a constant factor (and assuming even element distribution), constant-time performance for insertions and lookups can be achieved. This article describes the use of this technique to improve the performance of Icon's table and set implementations.

This work was motivated by a few data-intensive applications that ran slowly due to poor table performance. The solution comes from the article "Dynamic Hash Tables" by Per-Åke Larson in the April, 1988 *Communications of the ACM*.

## Algorithm Overview

Dynamic table expansion is achieved with dynamically-sized arrays. When the average hash-chain length becomes too long, a slot is added to the hash table. When all the slots are consumed, a new list of slots (called a segment) is added. Each new segment doubles the number of slots in the table. However, elements are moved into these slots incrementally in order to avoid long pauses in program execution while handling the expansion. Only one slot is added (chosen from the end of the last segment) for every expansion, and only one slot has to be "split" in order to move all the appropriate elements to that new slot. This is because as the table grows the slot computation function changes both by increasing the modulo factor by 2 and by sometimes "looking" at two slot indices when all slots of the last segment are not yet in use. That is, when the table is growing into a new segment, some upper slots are not in use yet, so the elements that later (upon further expansion) will occupy those slots are still in the lower half of the slots. The effect of doubling the modulus value is to have one more bit (bit  $n$ ) of the hash number become significant. For example, if bit  $n$  is 0, then a lower index  $i$  is selected; if it is 1, then the value  $2^n+i$  is selected. So if slot  $2^n+i$  is being expanded, only elements in slot  $i$  could have values that have  $i$  in the lower bits *and* have 1 in the newly significant bit  $n$ .

Many of the computations involve numbers that are a power of 2. This requires (or permits, depending on how you look at it) integer logarithm (via table lookup) and power (via shifting) operations to compute the segment address and the index into the segment.

Although the basic ideas of dynamic hashing are simple, making them fit the needs and requirements of the Icon interpreter resulted in several changes to the algorithm and to the Icon interpreter. This is outlined below.

## Hashing

Hash functions were reimplemented for strings and csets, which had poor distributions under the scheme described above, although they worked fine under the old implementation. (The distributions now appear to be very good, judging from profiling.) The basic problem is that the number of slots is a power of 2, which does not scramble non-random hash numbers very well. Thus, the hash functions were made more robust by using scrambling (large prime) multipliers and modulus functions. The performance of hashing integers was improved by calling a macro that in-lines integer hashing rather than calling the hash function.

## Segment Handling

Larson's algorithm assumes that a table is a central data structure. An Icon program, however, may create many tables. Consequently, per-table memory overhead is an important consideration. This consideration requires changes in Larson's segment expansion algorithm.

Larson's algorithm adds one slot per expansion, as does the algorithm described here. However, to add a new segment of slots, Larson's algorithm adds a constant number of slots, but the Icon version doubles the current number. This avoids a large segment directory.

The Icon algorithm uses 12 segments, but Larson's has 256 of 256 slots each. Larson's allows about 300,000 elements to be stored before performance degrades. The doubling scheme with 12 segments starting with a 32-slot segment allows for approximately  $2^{17}$  elements to be stored before performance degrades.

Since there is an extra level of indirection necessary to handle a dynamic slot list, some performance is lost. To overcome this, the first segment is treated as a special case. The first segment is stored directly in the header for tables and sets. Then, during slot indexing, the hash value is tested to see whether it lands in this first segment. This allows direct access to

lands in this first segment. This allows direct access to the segment without extra segment and subscript computation. It improves small table performance considerably, but has little impact on large tables.

### *Changes to Operations*

Because sets and tables no longer have a uniform number of slots, the methods used in chaining through sets and tables have changed considerably. In operations such as element generation (!x) and copy(x), one more level of looping is added to index through the dynamically added segments. In the set operations union, intersection, and difference, basic assumptions have changed. Previously, to perform set intersection individual slot element-chains were intersected. This worked because elements not on the same chain could not have the same hash number, and hence could not be the same value. Now two elements with the same hash number but in different sets can be assigned to different slots. Thus, it is not possible to make useful assumptions about the relative positions of a value in two different sets. This requires that the basic set operations perform more general lookups. This could result in a performance penalty, especially for smaller sets.

### *Reorganization and Element Generation*

The most significant interaction between Icon and dynamic hashing is in element generation. Element generation from a table or set can be adversely affected by insert or delete operations because they regularly call expansion and contraction routines. Element generation visits the slot chains in order. Reorganization migrates elements between slots, potentially causing element generation to visit elements multiple times, or not at all. To prevent this, two fields were added to the internal set and table representations. One holds a count of the number of element-generation expressions currently working on the hash table, and the other keeps track of the maximum possible slot being addressed by any of the active element-generation expressions. The maximum value is useful because all reorganizations *above* this point cannot affect the element-generation invocations. Since Icon has coexpressions, however, element-generation expressions do not necessarily start and complete in a LIFO manner. This means that no element-generation expression can be sure where in the table another element-generation invocation is; the maximum value can only be adjusted "downward" if one or no element-generation expressions are active on a table. Thus, the maximum value is safe, but not precise.

### *Performance Measurements*

The results of the implementation are very positive. For a small overhead (approximately 5% in time on small tables, 10% in space), element access time is uniform over all tables.

Below are graphs showing the impact of dynamic hashing on the performance of Icon programs that use tables or sets. All of these measurements were made on a Sun 3 running Sun UNIX.

The first graph shows performance snapshots of a program that loads a table with words from a dictionary. It was run on both the old and new table implementations, at two different initial memory allocations. Initial allocation has as much impact as the table implementation.

The second graph shows the performance of a broad class of applications that use sets and tables intensively. The programs *concord* and *sets* create large numbers of small tables and sets, respectively. They are the only programs that do not show an improvement in performance. The performance of sets also can be attributed to the large number of set unions, intersections, and differences it performs.

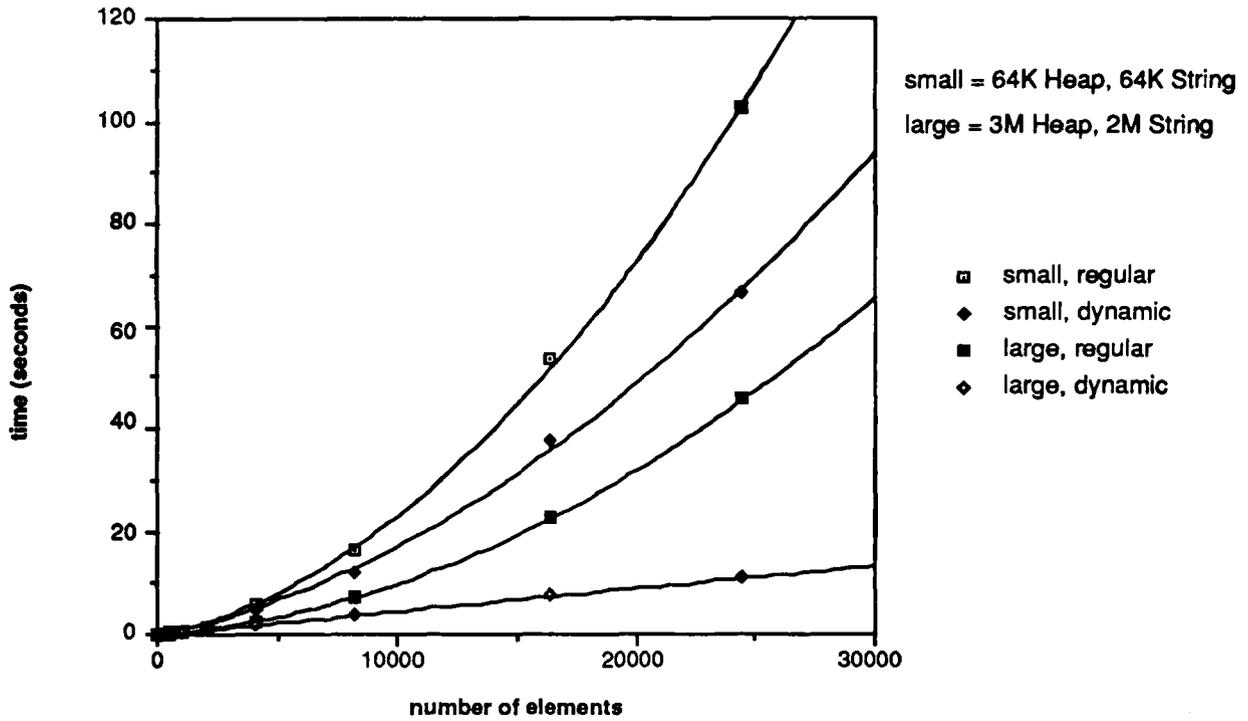
### *Conclusions*

The implementation is complete. Contraction and expansion are supported for both tables and sets. The implementation is likely to appear in a future release of Icon.

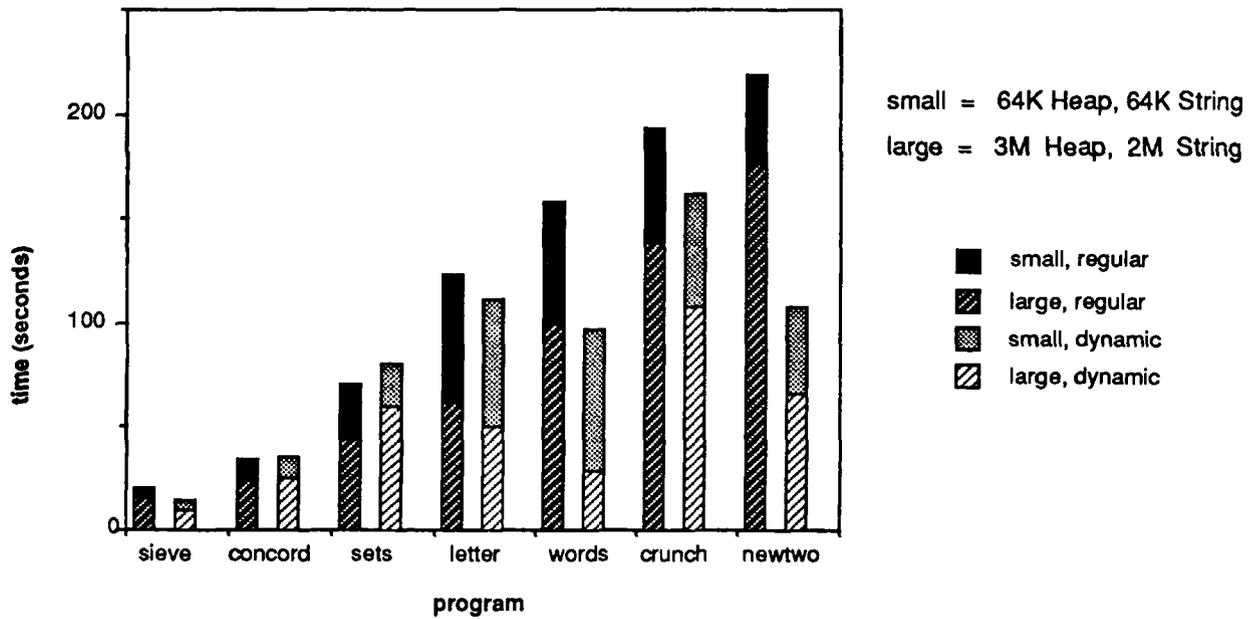
The poor performance of built-in abstractions does not encourage good use of language features. The choice of flexible algorithms like dynamic hashing for high-level features in programming languages frees the programmer from many concerns of implementation by assuring good performance of the abstractions provided. Although the work required to produce good performance for tables and sets was large, its availability to all users of Icon will assure the effort pays off.

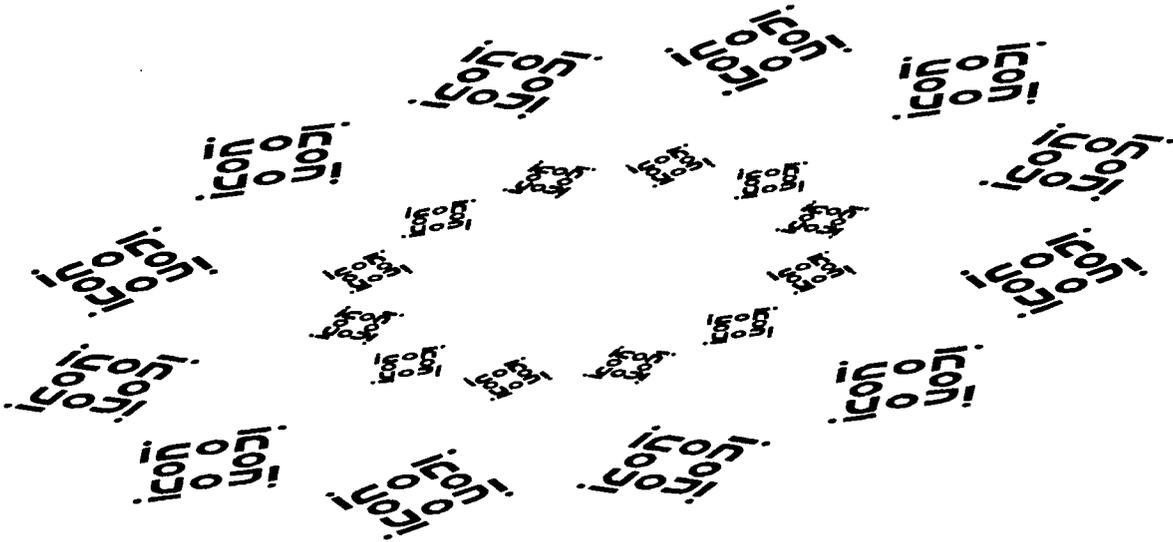
The fact that performance does not improve much for tables under 1,000 elements is unfortunate. Much of this is probably due to the dynamic typing of Icon, which introduces a significant overhead in referencing values. In an efficiently compiled implementation of Icon the cross-over point likely would appear much sooner.

### Inserting Strings in a Table



### Application Benchmarks





## Icon Benchmarks

In *Newsletter 28* we gave some benchmarks for Version 7.5 of Icon for different computers and C compilers. Since that time, we've done more benchmarking and also received some from others. The results are given in the table that follows.

Please note that we cannot verify many of the figures or vouch for their correctness.

The programs used for the benchmarks are the same as those used previously:

**ipxref:** This program, similar to the one in Version 7.5 of the Icon program library, produces a cross-reference listing of an Icon program. It does lots of text processing and some list manipulation. The test input (the program itself) consists of 7,049 characters and the output is 5,246 characters.

**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. The output is 273,171 characters.

**rsg:** This program, similar to the one in Version 7.5 of the Icon program library, generates randomly selected sentences. The program uses tables and lists extensively and synthesizes text. The input is 664 characters and the output is 10,117 characters.

**sieve:** This program implements the sieve of Eratosthenes, using set manipulation. The test produces the primes in the integers to 2,000. The output is 2,357 characters.

Program output was suppressed in timing runs 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 times shown on the next page are CPU times spent in the Icon run-time system (iconx) in seconds. The last column is a weighted sum that ranks the programs equally.

There is a considerable range in clock resolution among systems, ranging from microseconds to seconds. In some cases, the full internal clock resolution is not available at the program level, resulting in crude timings. In some cases, the figures listed are the averages of several runs. Timings on multi-user systems tend to vary with load. Where possible, tests were run on unloaded systems. The times given should be considered only as approximations.

In some cases, information about computer models, operating systems, and C compilers is incomplete. Where no C compiler is listed, the one used is the standard one for the operating system.

In most cases, Icon was compiled with the standard C optimization. Unoptimized compilations are indicated by the symbol  $\emptyset$ .

Our thanks to all the persons — over 30 — who contributed to the listings on the next two pages.

If you're running Version 7.5 of Icon on a system for which benchmarks are not listed and you'd like to run them, let us know. The benchmark programs are available from the Icon Project via RBBS and FTP.

## Benchmarks for Version 7.5 of Icon

Computer	System	C Compiler	Time (seconds)				
			ipxref	queens	rsg	sieve	wt. sum
IBM 3090-200E	VM/CMS	Waterloo 3.0	1.40	6.31	0.75	0.29	8.59
Amdahl 580	System V		1.38	7.12	0.78	0.28	8.85
Apollo DN10000	AEGIS	new compiler	2.32	10.73	1.21	0.55	14.69
Decstation3100	Ultrix		2.45	11.53	1.38	0.52	15.47
MIPS/R3000	System V		2.52	12.23	1.43	0.52	15.97
Sun-4/280	SunOS 4.0		2.94	13.33	1.83	0.63	18.92
Sun SPARCstation 1	SunOS 4.0		2.75	12.97	2.32	0.57	19.46
IBM 370/3084	MVS/XA	SAS 4.00	3.49	17.21	1.67	0.64	20.80
IBM 370/3081	VM/CMS	Waterloo 3.0	4.73	21.28	2.53	1.00	29.12
Apollo DN10000	AEGIS	old compiler	4.90	22.80	3.35	0.90	31.59
DEC VAX 8650	System V		4.90	26.08	2.78	1.23	33.29
DEC VAX 8650	4.3 BSD	Gnu	5.30	26.10	2.90	1.20	34.11
DEC VAX 8650	4.3 BSD	PCC	5.40	27.40	3.00	1.20	35.01
DEC VAX 8650	VMS		5.96	30.02	3.04	1.24	37.24
Compaq 386 <sup>1</sup>	MS-DOS	Metaware HighC <sup>2</sup>	6.20	26.31	3.46	1.53	39.48
IBM 370/4381	VM/CMS	Waterloo 3.0	6.90	31.15	3.69	1.47	42.58
Apollo DN10000	AEGIS	old compiler Ø	6.98	32.08	4.88	1.22	44.75
Sun-3/280	SunOS 4.0		7.02	33.15	4.37	1.92	48.38
386 Clone <sup>3</sup>	MS-DOS	Metaware HighC <sup>2</sup>	7.00	34.00	5.00	2.00	50.95
Compaq 386 <sup>4</sup>	MS-DOS	Metaware HighC <sup>2</sup>	6.20	41.73	5.27	2.32	55.15
IBM RT 135	AIX		9.60	47.00	5.10	2.20	61.44
Apollo DN4500	AEGIS		8.50	44.88	5.32	2.63	62.31
NeXT	Mach	(Sun-3 binaries)	9.67	47.40	5.48	2.33	63.63
IBM PS/2 Model 80 <sup>5</sup>	Xenix/386		10.30	50.00	5.73	2.40	66.76
Sun-3/60	SunOS 4.0		10.38	50.32	5.90	2.53	68.35
NeXT	Mach	Ø	11.25	55.62	6.27	2.48	72.36
Zenith Z-386 <sup>6</sup>	MS-DOS	Metaware HighC <sup>2</sup>	12.00	54.00	6.00	3.00	75.75
IBM PS/2 Model 80 <sup>5</sup>	MS-DOS	Metaware HighC <sup>2</sup>	11.00	56.00	7.00	3.00	77.58
Sun-3/140	SunOS 4.0		13.48	65.37	7.70	3.25	88.64
Apollo DN4000	AEGIS		14.27	65.88	9.03	3.32	94.38
Apollo DN3500	AEGIS		14.00	73.48	8.28	3.77	97.40
Macintosh SE/30	MPW	MPW 3.0	16.18	80.53	9.43	4.02	108.43
Macintosh IICx	MPW	MPW 3.0	16.32	80.88	9.73	4.07	109.96
Sun-3/50	SunOS 4.0		17.50	85.30	9.50	4.20	113.69
Apollo DN570T	AEGIS		17.65	87.03	9.60	4.12	114.30
ATT 3B4000	System V		18.17	89.08	9.91	3.92	115.44
Macintosh II	MPW	MPW 1.02	17.40	87.00	10.90	4.70	121.38
Zenith Z-386 <sup>6</sup>	System V	Green Hills	18.63	92.15	11.70	4.82	128.30
ATT 3B2/70	System V		20.38	98.18	10.88	4.42	128.37
DEC VAX 785	4.3 BSD	PCC	22.28	104.85	11.20	4.53	135.51
Macintosh II	MPW	MPW 3.0	21.27	99.08	12.40	4.97	138.12
Zenith Z-386 <sup>6</sup>	System V	PCC	20.92	100.62	12.33	5.25	139.72
DEC VAX 780	System V		22.07	118.83	11.68	5.05	144.78
Zenith Z-386 <sup>6</sup>	MS-DOS	Microsoft 5.10 <sup>7</sup>	22.00	109.00	14.00	6.00	154.13
IBM PS/2 Model 80 <sup>5</sup>	MS-DOS	Microsoft 5.10 <sup>7</sup>	23.00	110.00	14.00	6.00	156.16
IBM PS/2 Model 80 <sup>5</sup>	OS/2 <sup>8</sup>	Microsoft 5.10 <sup>7</sup>	23.00	111.00	14.00	6.00	156.51
Apollo DN3000	AEGIS		25.35	120.00	13.37	5.70	159.81
IBM PS/2 Model 80 <sup>5</sup>	MS-DOS	Turbo 1.5	23.00	117.00	15.00	6.00	161.44
IBM PS/2 Model 80 <sup>5</sup>	MS-DOS	Microsoft 5.10 <sup>9</sup>	28.00	131.00	16.00	8.00	190.76
ATT 6386	System V		19.62	100.10	19.20	10.93	194.38

See the footnotes on the next page.



## Programming Corner



We have two contributions from readers this time. Rich Clayton sent the following note:

My Icon programs often are written as a series of filters on objects in a

stream. The filters do one object look-ahead on the stream with a read/pushback sequence, implemented something like this:

```
global push_back_o
procedure next_o()
  local o
  if \push_back_o then {
    o := push_back_o
    push_back_o := &null
    return o
  }
  else return read_o()
end # next_o
```

It eventually occurred to me that I could use the local variable's &null initialization to rewrite the then part of the if expression as

```
return o :=: push_back_o
```

which lets the procedure body collapse to

```
return ((\push_back_o & (o :=: push_back_o)) | read_o())
```

and again to

```
return (o :=: \push_back_o) | read_o()
```

Alan D. Corre, author of the soon-to-be-published book, *Icon Programming for Humanists*, sent the following:

I wanted to write an Icon procedure to check if a string has precisely 22 characters (the size of the Hebrew alphabet) and no duplicate characters, so I wrote the following:

```
procedure checkstring(abc)
local cs, current
if *abc ~= 22 then fail # check length
cs := '' # initialize cset
abc ? every 1 to 22 do {
  current := move(1) # select a char
  if cs ** current ~== '' then fail # already a member
  cs ++:= current } # char is ok
return
end
```

The procedure worked fine, but I said: "That isn't an Icon procedure. It's a thinly disguised Pascal function. Now write an Icon procedure." So I forsook "if mouse in hole" and wrote:

```
procedure checkstring2(abc)
local t, current
if *abc ~= 22 then fail
t := table(0)
abc ? every 1 to 22 do {
  current := move(1)
  !t[current] | fail
  t[current] := 1 }
return
end
```

More Icon constructs, but no better really. Then I wrote:

```
procedure checkstring3(abc)
return *abc = *cset(abc) = 22
end
```

Now that's an Icon procedure.

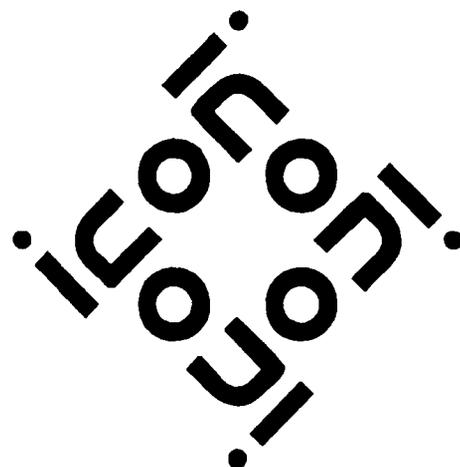
---

## ProIcon Licenses

The ProIcon Group has announced site and networking licensing for ProIcon, an implementation of Icon for the Macintosh (see *Newsletter 30*). Discounts for educational institutions also are available.

For more information, contact

The ProIcon Group  
P.O. Box 1123  
Salida, Colorado 81201-1123  
U.S.A.  
719-539-3884



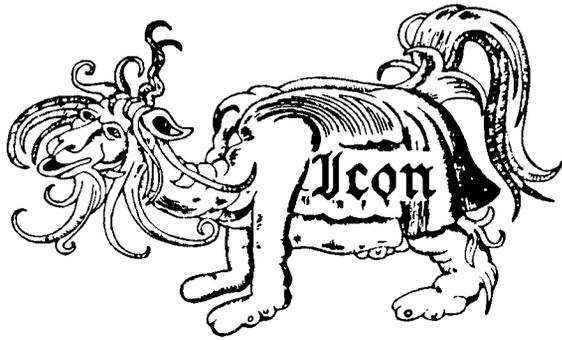
## Ordering Icon Material

### *What's Available*

There are implementations of Icon for several personal computers, as well as MVS, UNIX, VAX/VMS, and VM/CMS.

Source code for Icon is available. There also is a program library and documentation both on the Icon programming language itself and on its implementation.

The current version of Icon is 7.5. All the program material here is for Version 7.5.



### *Icon Program Material*

**Personal Computers:** Executables and source code for Icon for personal computers are provided separately. Each package contains printed documentation that is needed for installation and use. *Note:* Icon for personal computers requires at least 512KB of RAM; it may require more on some systems.

**MVS and VM/CMS:** The MVS and VM/CMS packages contain executables, source code, and documentation in printed and machine-readable form.

**UNIX:** The UNIX package contains source code (but not executables), documentation in printed and machine-readable form, test programs, and related software. 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. *Note:* executables for XENIX and the UNIX PC are available separately.

**VAX/VMS:** 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.

**Porting:** Icon source code for porting to other computers 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.

### *Source Updates for MS-DOS*

Updates to the Icon source code for MS-DOS are available by subscription. A subscription provides five complete updates. Updates are released about three times a year.

### *Icon Program Library*

The Icon program library consists of Icon programs, collections of procedures, and data. Version 7 of Icon is required to run the library. The Icon program library is being issued in parts. Part 1 presently is available. *Note:* Version 7 of the Icon program library is available only on diskettes. The UNIX tape and cartridge packages and the VMS tape package presently contain an older version of the Icon program library. The Icon program library is not yet available for MVS or VM/CMS.

### *Documentation*

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.

### *Shipping*

Except as noted, the prices listed 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, money order, or credit card (Visa or MasterCard). 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.

The symbol  identifies material that is new since the last *Newsletter*. The symbol  identifies material that has been updated to Version 7.5 since the last *Newsletter*.

### Ordering Instructions

*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. 3.5" diskettes are written in the format appropriate for the system for which they are intended.

MVS and VM/CMS tapes are available only at 1600 bpi. When ordering UNIX or VMS 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).

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

### Program Material

#### Amiga:

AME:		executables	\$15
 AMS:		source	\$15
AML-1:		library	\$15

#### Atari ST:

ATE:		executables	\$15
ATS:		source	\$20
ATL-1:		library	\$15

#### Macintosh/MPW:

ME:		executables	\$15
MS:		source	\$25
ML-1:		library	\$15

#### MS-DOS:

DE:	 (2) or 	executables	\$20
DS:	 (2) or 	source	\$25
DL-1:	 or 	library	\$15
DU:		source updates	\$50

#### MS-DOS/386:

DE-386	 or 	executables	\$15
--------	---	-------------	------

#### MVS:

MT:		entire system	\$30
-----	---	---------------	------

#### OS/2:

OE:	 or 	executables	\$15
-----	---	-------------	------

#### UNIX:

UT-T:		entire system (tar)	\$30
-------	---	---------------------	------

UT-C:		entire system (cpio)	\$30
-------	---	----------------------	------

UC-T:		entire system (tar)	\$45
-------	---	---------------------	------

UC-C:		entire system (cpio)	\$45
-------	---	----------------------	------

UD-M:	 (6) or  (4)	entire system (cpio)	\$40
-------	--	----------------------	------

UL-1:	 or 	library (cpio)	\$15
-------	---	----------------	------

#### UNIX - UNIX PC:

UPE:		executables	\$15
------	---	-------------	------

#### UNIX - XENIX:

XE:	 or 	executables	\$15
-----	---	-------------	------

#### UNIX - XENIX/386:

XE-386:	 or 	executables	\$15
---------	---	-------------	------

#### VAX/VMS:

VT:		entire system	\$30
-----	---	---------------	------

#### VM/CMS:

CT:		entire system	\$30
-----	---	---------------	------

#### Other systems (for porting):

PF-A:	 (5)	flat system (ASCII)	\$40
-------	---	---------------------	------

PF-K:	 (2)	flat system (ARC)	\$30
-------	---	-------------------	------

PH-A:	 (5)	hierarchical system (ASCII)	\$40
-------	---	-----------------------------	------

PH-K:	 (2)	hierarchical system (ARC)	\$30
-------	---	---------------------------	------

PL-1:		library (ASCII)	\$15
-------	---	-----------------	------

### Documentation

LD: Language documentation package. *The Icon Programming Language* (Prentice-Hall, 1983) and six technical reports. \$32.

ID: Implementation documentation package. *The Implementation of the Icon Programming Language* (Princeton University Press, 1986) and update. \$45.

NL: Back issues of the *Icon Newsletter*. \$.50 each for single issues (specify numbers). \$7.50 for a complete set (Nos. 1-30). There is no charge for overseas shipment of single back issues, but there is a \$5.00 shipping charge for the complete set.

