THE UNIVERSITY OF ARIZONA

TUCSON, ARIZONA 85721

DEPARTMENT OF COMPUTER SCIENCE

The Icon Newsletter

Number 23 — February 3, 1987

Questionnaires

By the end of January, 638 questionnaires from Newsletter 22 had been returned to us. This amounts to about 32% of the newsletters sent. We are presently tabulating the results and will publish them in subsequent newsletters.

Status of the Newsletter

Everyone who returned the subscription verification form from Newsletter 22, or who has been added to our mailing list since that newsletter was sent out, is scheduled to receive newsletters through number 25 at no charge.

Each address label for the newsletter has a number in brackets at the end of the top line that indicates the last issue for which the subscription is scheduled.

None of this indicates that we necessarily are planning to charge for the newsletter. We'd prefer not to, and we realize that payment would be administratively complicated for persons who would subscribe through organizations, as well as expensive and cumbersome for persons outside of the United States. However, the cost of printing and mailing the newsletter is significant, when taken as a whole.

Incidentally, we should have made it clear what we meant by a 'nominal' charge when we prepared the questionnaire; several persons had concerns about just how nominal it might be. To clarify the issue, we were thinking of a dollar or two per newsletter.

For what it's worth, here is the tabulation of the questionnaires received to date on the issue of willingness to subscribe to the newsletter:

yes: 455   no: 89   maybe: 16   no answer: 78

The persons who indicated that their answers depended on the definition of 'nominal' are tabulated as 'yes'.

Thus, approximately 80% of the persons who answered this question expressed a willingness to pay for their subscriptions.

All of this is academic at this point. There will be no subscription charge for the next few issues and there may never be.

Implementation News

Icon for the Macintosh

Bob Alexander has completed an implementation of Icon for the Macintosh and has placed it in the public domain. This implementation runs as part of the *Macintosh Programmer's Workshop* (MPW) integrated environment. MPW is required to run Macintosh Icon — it cannot run as a stand-alone application.

This is a large-memory-model implementation that requires a 512K or larger Macintosh.

Icon for the Macintosh is distributed by the Icon Project on the same terms as all other implementations it distributes. See the order form at the end of this Newsletter, but do not order Icon for the Macintosh unless you have MPW.

The Icon program library and source code for the Macintosh implementation will be available at a later date.

MPW is a window-oriented development system that comes highly recommended to us. It consists of the MPW shell (which is needed to run Icon), an assembler, a linker, and a number of utilities. MPW is available exclusively through:

Apple Developer's and Programmer's Association
290 SW 43rd Street
Renton, WA 98055

(206) 251-6548

The fee to join APDA is $20 and MPW costs $150. There also is an MPW C compiler, but it is not needed to run Icon.

Icon for the Atari ST

O. Rick Fonorow and Jerry D. Nowlin report that they have completed an implementation of version 6.3 for the Atari ST computer family. The Lattice C compiler was used for the port (giving 32-bit ints). This version takes all available memory, but uses fixed-sized regions for Icon's data. It will run on the 520ST with no other programs (i.e. desk accessories) resident. This implementation is available on the Atari Base BBS: (408) 745-4758.

MS-DOS Icon

Source Code: Source code for Icon for MS-DOS is now available; see the order form at the end of this newsletter.

So far, Icon for MS-DOS has been successfully compiled under Version 3.2 of Lattice C (which has not been officially released yet, but is due soon) and Version 4.0 of Microsoft C. The source-code distribution includes configurations for these two C compilers and provisions for adding others. An update service is provided so that persons who get this source code can stay up to date with corrections and extensions. Information about this service is included with the distribution.

Icon Produced by Microsoft C: The small-memory-model (SMM) implementation of Icon as produced by the Microsoft C compiler is somewhat smaller and faster than the one produced by Lattice C. The SMM version of Icon currently being distributed was compiled under Microsoft C.

The large-memory-model (LMM) situation is more complicated. The one produced by Microsoft C is smaller and runs about twice as fast, in our tests so far, as the one produced by Lattice C. However, Microsoft C does not allow Icon to expand its regions for allocating strings and blocks in the way that Lattice C does. Consequently, the LMM implementation from Microsoft C may not be able to use all memory that is available. The form of memory allocation in which regions are not expandable is called *fixed-regions* (FR).

The situation is further complicated by the fact that each region presently is limited to 64k. For many users, however, the FR implementation provides sufficient memory (much more than the SMM) and its speed advantage may outweigh its limitations. The Microsoft FR implementation therefore is being offered as a third MS-DOS Icon alternative; see the order form at the end of this newsletter.

LMM Upgrade Offer: Early versions of the LMM implementation of MS-DOS Icon had a number of bugs in storage management. The symptoms range from hung systems to trashed strings. These problems have now been corrected.

Persons who wish to bring their LMM implementation up to date may do so by sending $4 ($7 outside of the United States and Canada), together with the serial number of their LMM diskette, to the Icon Project. Do not send back the original diskette; we will provide a new one. This offer applies only to LMM serial numbers 1 though 173 (later ones already have the corrections) and to persons who purchased their LMM diskettes from the Icon Project.

Porting Icon to Other Computers

A form of the Icon source designed to facilitate implementations on new computers is now available. See 'Icon for porting' on the order form at the end of this newsletter.

This form of the source does not require a hierarchical file system like the usual version of the Icon source. This source is otherwise identical to the version of the source used for other implementations.

The distribution includes documentation on modifying the source for new computers and test suites.

The Implementation Book

The 'Implementation book' is now available. The book *The Implementation of the Icon Programming*

The Icon Newsletter

Madge T. Griswold and Ralph E. Griswold

Editors

The Icon Newsletter is published three or four times a year, at no cost to subscribers. For inquiries and subscription information, contact:

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

Language, by Ralph E. Griswold and Madge T. Griswold, first in the new Princeton Series in Computer Science, has just been released by Princeton University Press. This book concentrates on the run-time system and describes data structures, the Icon virtual machine, how generators work, and the techniques used for storage allocation and garbage collection. It contains information on modifying the source code for Icon and includes both exercises and suggested projects. The description of the implementation corresponds to the source code that is now available.

The book, which is published in hard cover, may be ordered from Princeton University Press for $39.50. There is no shipping charge for orders sent Fourth Class. For UPS delivery, an additional amount of $1.50 is charged. Sales tax is charged for shipments to California (6.5%) and New Jersey (6%). To order your copy call (609) 896-1344 or write to

Princeton University Press
3175 Princeton Pike
Lawrenceville, NJ 08648

Please note: This book is not available from The Icon Project.

A clarification: There has been some misunderstanding about this book. It describes the *implementation* of the language and is not a revision of *The Icon Programming Language* (Prentice-Hall, 1983), which describes the language itself. The two books relate to different versions of Icon. The language book refers to Version 5, while the implementation book refers to Version 6. A technical report describing the Version 6 language supplements the language book.

Access to the Icon Project

Since this newsletter is going to many persons who have not received previous ones, we are repeating here the material on how to get information and program material from the Icon Project.

Our mailing address, telephone number, and electronic mail addresses are:

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

(602) 621-6613

The electronic mail addresses are:

icon-project@arizona.edu
... {allegra, ihnp4, noao}!arizona!icon-project

It is best to write or use electronic mail for technical questions, bug reports, and so forth, since the telephone usually is answered by a secretarial person who is not familiar with technical matters.

When sending electronic mail to the Icon Project, it is important to address it to icon-project, rather than an individual. This will assure your message gets prompt attention in the case that someone happens to be away.

Program Material via FTP

Icon program material is available to persons who have access to Internet via anonymous FTP from arizona.edu. The Icon material presently available includes the UNIX, VMS, and MS-DOS implementations described on the order form at the end of this newsletter, as well as the source for porting and even some SNOBOL4 material. To see what is available and how large the files are, cd to icon and get README. For information on SNOBOL4 material, cd to snobol4 and get README.

It is essential to use the binary ('image') mode when getting program material via FTP.

Electronic Bulletin Board

Our electronic bulletin board is available to persons who want to dial up and download program material. The UNIX and VMS implementations are not available, since their size makes downloading impractical, but all the other material available via FTP also is available on the bulletin board.

Both Kermit and Xmodem protocols are supported. There is also a limited message facility.

The telephone number of the bulletin board is (602) 621-2283. Before dialing, set your computer's telecommunication program to full duplex, 7 bits, even parity, 1 stop bit, at a baud rate of 2400, 1200, or 300. The initial answering baud rate is 1200. If you dial in at another rate, you will need to cycle the baud rate detection with break/carriage-return sequences until your baud rate is recognized.

When you are properly connected, you should receive the following message:

```
xenix286!login:
```

Respond to this with guest followed by a carriage return and you will get a menu of the available services. From there, the operation of the bulletin board should be self-explanatory.

The bulletin board normally is in operation from 5 pm MST to 8 am MST, as well as all day on weekends. It also is in operation during daytime hours when the computer on which it runs is not otherwise occupied, but service interruptions during daytime hours may occur without warning.

This bulletin board is designed primarily for downloading program material and does not have a general message facility. It is, however, possible to leave remarks and questions for the Icon Project. Answers are posted where all callers can read them.

BIX

The BYTE bulletin board BIX has a discussion group for Icon and SNOBOL4. Information about BIX can be found in any recent issue of BYTE. See, for example, page 306 of the February 1987 issue. As a BIX subscriber, you can enroll in this discussion group by

```
join other.langs/icon.snobol
```

The MS-DOS implementations of Icon, including source code, can be downloaded from BIX.

From our Mail

The returned questionnaires have given us enough material for this feature of the Newsletter for years. The problem has been selecting the questions of most general interest. Here are a few:

Can I get a printed listing of the source code for Icon?

Icon is a large program — four programs, actually — and its source code totals about 27,000 lines. Even when printed in a reduced size in a 'two-up' format, it runs several hundred pages. Listings are bulky and expensive to produce. Consequently, we have not offered to provide listings. If you *really* want one, we'll figure out what it would cost to run one off and ship it to you. If enough persons ask, we'll try to come up with a reasonably economical way to print copies in quantity.

How can I get back issues of the Icon Newsletter?

We'll send you copies of recent issues at no charge. Most of the early issues are out of stock and their contents are very stale. However, material from the 'Programming Corner' is collected in a technical report (TR 86-2), which is available on request. If you *really* want a complete set of Newsletters, we can have one made up but would have to charge you for the cost of copying and shipping.

Will your SCO XENIX V implementation of Icon run on my IBM XENIX 3 system?

We are told so, although the extent of binary compatibility among different XENIX systems remains somewhat of a mystery to us. We generate 8086 code and use the loader compatibility option. Several persons with different flavors of XENIX have used our SCO XENIX Icon successfully.

Can I FTP program material from your computer via BITNET?

No. FTP is a connection-oriented service, while BITNET is batch-oriented.

I don't have access to FTP. Can you send me a copy of Icon by electronic mail?

Icon is too large to send by electronic mail, which was not designed for that kind of transmission. Our electronic bulletin board is an alternative.

Has there been any work on architectural support for Icon.

Not as far as we know. We have no plans to do work in this area.

Is there a symbolic debugger or programming environment for Icon?

No. These are interesting projects and we've given some thought to them, but we haven't done anything yet. If someone else is working in these areas, we don't know about it.

Is a library of mathematical functions going to be added to Icon?

We are considering this. However, it would increase the size of Icon, which already strains the memory capacity of some computers. An alternative for the present is the package of mathematical procedures in the Icon program library.

I've heard there are undocumented UNIX interfaces in Icon that make it handy as an alternative to shell scripts.

No, everything is documented. However, Icon string-processing facilities in conjunction with pipes and the `system` function make Icon attractive for many tasks for which shell scripts often are used.

I'd like to be able to use the full graphics capabilities of my PC in Icon. Even being able to read a single character would be very helpful. Is there any hope here?

Cheyenne Wills is working on a set of functions for MS-DOS Icon that will make it possible to do many such things. We expect these enhancements to be available in a month or so. Source code is available for persons who want to do things like this themselves. As a general remark, Icon is designed to run in a wide range of computational contexts where there are widely varying needs. The language itself does not attempt to include all possibilities. Instead, enhancements are added to different implementations to suit the environments. This takes a while.

Is it possible to get Cinema for MS-DOS?

[Cinema provides an animated display of string scanning, showing the progression of states, the movement of the cursor, and so forth.] It might be possible, but it would be a lot of work. Cinema, as presently written, depends heavily on the facilities provided by the Sun Workstation windowing system. It's also possible that the maximum amount of memory available under MS-DOS would not be enough.

Programming Corner

A Program to Deal and Display Bridge Hands

The choice of data representations often is one of the most important aspects in the design of programs that perform nonnumerical computations. Not only do data representations affect program speed and space requirements, but they also may play a central role in the difficulty or ease of writing the program.

Icon offers an unusually wide variety of data types. While it provides more flexibility than is found in some other programming languages, it also presents the programmer with more choices. Sometimes the best choice is not the obvious one.

The following program, which is an adaptation of one from the Icon program library, illustrates a compact data representation that often is useful in programs that manipulate a small number of objects. This data representation also gives computational efficiency, since it allows the use of built-in operations that otherwise might have to be provided as procedures.

The problem is to produce and display hands in the game of bridge. The basic operations are shuffling the deck, dealing the cards to the players, and displaying the results. Not surprisingly, displaying the results is the most difficult part of the program.

In the game of bridge, the deck consists of 52 cards with 13 denominations in four suits. The suits are clubs, diamonds, hearts, and spades, and the denominations are 2 through 10, jack, queen, king, and ace.

There are lots of possible ways of representing the cards. Since a card has two attributes — its suit and its denomination — a record type with these fields is a possibility. This representation presents the problem (among others) of how to represent the deck — that is, how to keep track of the cards. A list of the records is a possibility. A simpler, if less elegant, choice of data representation is simply a list of 52 elements in which the position encodes the attributes of the individual card. In this case, the real representation of the card is its index in the list. In other words, 52 integers are all that are necessary, and the list is used to keep them together. Other representations are possible. For example, the string "8C" might represent the eight of clubs, and so on.

There is clearly some advantage in having a simple object to represent a card. The method used in the following program is to associate a unique character with each card. This allows groups of cards to be represented by strings, and cset operations can be used to operate on groups of cards.

Since there are 52 cards, 52 different characters are needed. For the program that follows, any 52 characters will do, but a convenient choice (purely by coincidence) is

```
deckimage := &lcase || &ucase
```

This choice has the additional advantage of facilitating debugging.

The next question is which character corresponds to which card. This decision can be made in many ways. The one chosen here is to consider the deck to be a concatenation of the suits in order, with the first 13 characters corresponding to the clubs, the next 13 to the diamonds, and so on. Thus, the characters abc ... m are clubs, the characters nop ... z are diamonds, the characters ABC ... M are hearts, and the characters NOP ... Z are spades. Except for possible debugging, however, these explicit correspondences never come up. The specific order of denominations is rather arbitrary, but it turns out to be convenient for display purposes to rank the cards according to the order of characters in the following string:

```
rank := "AKQJT98765432"
```

Thus, the character a is the ace of clubs, the character z is the two of diamonds, and so on. Again, this never comes up explicitly in the program.

It might appear that encoding the card deck as a string of characters would introduce all sorts of problems, especially in figuring out which card is which and producing output that gives an understandable representation. Actually, most of the operations in the program do not require such determinations. For example, shuffling is insensitive to the suits or denominations of cards — it simply is the rearrangement of a number of objects that are as anonymous as the backs of real cards are supposed to be.

Shuffling is a good place to start:

```
procedure shuffle(deck)
   local i
   every i := *deck to 2 by -1 do
      deck[?i] :=: deck[i]
   return deck
end
```

This procedure is an implementation of a method given by Knuth in his book, *Seminumerical Algorithms*. It operates by starting at the end of the deck, exchanging that card with a randomly chosen one, and then working down toward the beginning, chosing the exchange card from the remainder of the deck. Whether or not this produces a 'good' shuffle is somewhat of an open question, but it seems to work well in practice.

Once the deck is shuffled, it is customary to distribute the cards to the four players by dealing them one-by-one to the four players in turn. This method of distribution is more of a convention than a necessity and is motivated partly by social considerations. If the deck really is shuffled properly, it is good enough to give the first 13 cards to the first player, the next 13 to the next player, and so on. It is also a lot easier to program.

The real fun comes in displaying the results of the deal. In bridge, it is customary to separate the cards in each hand into suits and to arrange the cards in each suit from higher to lower denomination. Here is where Icon's cset and mapping operations can be used to advantage. The idea is to extract from a hand of 13 cards all of the cards of a given suit by mapping the cards of the desired suit into themselves and mapping all other cards into a single character that is not in the deck, effectively throwing away the cards that are not in the desired suit. The blank character is useful for this elimination. For example, the mapping string to discard all cards that are not clubs is constructed as follows:

```
denom := deckimage[1+:13]
blanks := repl(" ",13)
Cmap := denom || repl(blanks,3)
```

The strings denom and blanks are used here in place of a more direct construction of Cmap, since they are useful in producing maps for the other suits.

Now,

```
clubs := map(hand,deckimage,Cmap)
```

assigns to clubs a string in which all the characters corresponding to clubs are left unchanged, while all other characters are blanks. This string still is 13 characters long, and probably contains a lot of blanks. The clubs can be obtained by constructing a cset with the blank removed:

```
clubs --:= ' '
```

(There's an augmented assignment operation you don't see very often. It does not appear in the actual program, where the result is computed in a single expression.)

At this point, clubs contains all the clubs in the hand, but they are in a cset and unordered. The desired string with the clubs in order and mapped into their denominations is produced by

```
clubs := map(clubs,denom,rank)
```

The result comes out correctly, since the automatic conversion of the cset to a string in the first argument to map puts the characters in alphabetical order.

That's about all there is to it, except for the mechanics of handling all of the suits in all of the hands and formatting the output in a manner that is customary, with the four hands arranged according to the points of the compass. Here's the complete program, arbitrarily set up to print five sets of hands. Comments have been removed to save space.

```
global deck, deckimage, handsize
global suitsize, denom, rank, blanks

procedure main()
   deck := deckimage := &lcase || &ucase
   handsize := suitsize := *deck / 4
   rank := "AKQJT98765432"
   blanks := repl(" ", suitsize)
   denom := &lcase[1+:suitsize]

   every 1 to 5 do display()
end

procedure display()
   local layout, i
   static bar, offset

   bar := "\n" || repl("-", 33)
   offset := repl(" ", 10)

   deck := shuffle(deck)
   layout := []
   every push(layout, show(deck[(0 to 3) *
      handsize + 1 +: handsize]))

   write()
   every write(offset, !layout[1])
   write()
   every i := 1 to 4 do
      write(left(layout[4][i], 20), layout[2][i])
   write()
   every write(offset, !layout[3])
   write(bar)
end

procedure shuffle(deck)
   local i
   every i := *deck to 2 by -1 do
      deck[?i] :=: deck[i]
   return deck
end

procedure show(hand)
   static Cmap, Dmap, Hmap, Smap
   initial {
      Cmap := denom || repl(blanks, 3)
      Dmap := blanks || denom || repl(blanks, 2)
      Hmap := repl(blanks, 2) || denom || blanks
      Smap := repl(blanks, 3) || denom
      }

   return [
      "S: " || arrange(hand, Smap),
      "H: " || arrange(hand, Hmap),
      "D: " || arrange(hand, Dmap),
      "C: " || arrange(hand, Cmap)
      ]
end
```

```
procedure arrange(hand, suit)
   return map(map(hand, deckimage, suit) -- ' ',
      denom, rank)
end
```

An example of the output from this program is:

```
               S: 3
               H: T7
               D: AKQ762
               C: QJ94

S: KQ987                    S: A652
H: 52                       H: AKQ4
D: T94                      D: 3
C: T82                      C: A653

               S: JT4
               H: J9863
               D: J85
               C: K7
```

A couple of things are left for you to consider, including the use of denom for all suits and the method used to provide the final layout.

Processing Command-Line Arguments

One final subject for this issue's programming corner concerns the use of command-line arguments and methods for processing them. In the program above, for example, it would be useful to be able to specify how many rounds of hands are to be produced and possibly to be able to set the seed for random number generation to produce different sets of hands. One way to do this is to provide an interactive interface, prompting the user for this information. An alternative method, and one that often is handier, is to allow the user to specify such matters as options on the command line when the program is executed.

For example, if the program above is named deal, it might be executed as

```
deal -h 10 -s 17
```

The argument following -h indicates that 10 rounds of hands are to be produced and the argument following -s indicates that the seed for random number generation is to be 17. The syntax shown here is the standard one for UNIX; other operating systems conventionally handle it differently. Such differences are inessential here. However, such command-line arguments, in whatever form, should be optional and defaults should be provided if they are omitted.

In the example above, there are four command-line arguments. These are passed as a list of strings to the main procedure of deal, as if the argument to main were

```
["-h", "10", "-s", "17"]
```

Thus, the main procedure of the program above might be rewritten as:

```
procedure main(a)
   local s, hands
         :
   hands := 5
   while s := get(a) do {
      case s of {
         "-h":  hands := integer(get(a)) | use()
         "-s":  &random := integer(get(a)) | use()
         default:  use()
         }
      }

   every 1 to hands do display()
                  :
```

The procedure use terminates program execution with an error message in case there is an erroneous command-line argument. UNIX favors a terse style for such error messages, and a version of use in this style might be

```
procedure use()
   stop("usage:  deal [-h n] [-s n]")
end
```

Note that get(a) provides an easy and concise way of obtaining the command-line arguments. Other possibilities are using an explicit index, as in a[i], or iterating over the list, as in !a. Try rephrasing the processing of the command-line arguments above to see why the use of get is better. Of course, get consumes the list. That does not matter in the case here. How could this be reformulated to avoid consuming the list while retaining the advantages that get provides?

Upcoming in the Newsletter

In addition to the usual features of this newsletter, we plan to include the following material in the next one:

- a tabulation of responses to the questionnaire
- what the Icon Project is all about
- descriptions of research in progress
- tips on efficient programming techniques in Icon

Ordering Information

General Information: The prices listed on the order form include media, printed documentation (but not the books on Icon), 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. Commercial express delivery will be provided if authorization and an account number to charge is given with the order.

Payment: Payment may be made by check or money order, but 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 drawn on a bank that does not have a branch in the United States. Payment should accompany the order. For organizations that cannot provide pre-payment, purchase orders will be accepted, but there is a $5 charge for processing such orders.

The UNIX Package: The UNIX implementation of Icon can be configured to run on most computers with UNIX-based operating systems, including the VAX, Sun Workstation, IBM RT PC, AT&T 3B, and Pyramid. The distribution contains source code, a test suite, the Icon program library, and documentation in machine-readable form. Object and executable files are not included, since they depend on the target computer.

The VMS Package: The VMS implementation requires VMS Release 4.2 or higher. The distribution contains source code, a test suite, the Icon program library, and documentation in machine-readable form. Object and executable files are included.

Note: The UNIX and VMS packages contain different system-specific configurations and support material. Neither will run on the other system.

The Macintosh Package: The Macintosh implementation runs as part of the MPW environment. It is not a stand-alone application; do not order this package unless you have MPW. The diskette includes executable files, a few sample programs, and documentation in machine-readable form. Source code and the Icon program library are not yet available.

The MS-DOS Packages: The MS-DOS implementation of Icon runs on computers with 8086/88/186/286-family processors. IBM hardware compatibility is not necessary. MS- or PC-DOS Version 2.0 or higher is required. There are both SMM and LMM implementations, and there are two forms of the LMM one. The SMM implementation uses 192k of memory. The LMM implementation requires at least 256k, but runs better with more. The SMM implementation is faster and more compact than the LMM one, but it cannot handle programs that need large amounts of storage. There are two forms of the LMM implementation, one that allows Icon's storage regions to expand as needed (referred to as LMM/ER on the order form) and a 'fixed-regions' one in which the regions cannot expand (referred to as LMM/FR on the order form). The LMM/FR implementation is considerably faster than the LMM/ER one, but it may not be able to use all of the available memory. It is inadvisable to run more than over version of Icon on the same computer because of file-naming conflicts. The SMM, LMM/ER, and LMM/FR diskettes include executable files, a few sample programs, and documentation in machine-readable form.

The source code for MS-DOS has been compiled successfully under Microsoft C Version 4.0 and Lattice C Version 3.2. It may be possible to use other C compilers, but this has not been done yet and certainly will require some work: *caveat emptor*. The distribution is in *arc* format and hence is suitable only for use on MS-DOS systems. It includes source files, a configuration system for different C compilers, a set of tools, a test suite, and documentation in machine-readable form.

The Icon program library for MS-DOS is the same for the SMM, and LMM/ER, and LMM/FR implementations and is distributed on a separate diskette.

The PC/IX Packages: The PC/IX implementation of Icon is a SMM one. Source code and the Icon program library are distributed separately. The source code diskettes contain configuration information and test suites for other UNIX installations, such as the IBM RT PC. All PC/IX diskettes are written in *dump/restore* format, which is the same format as *backup/restore* on the RT PC under AIX.

The XENIX Packages: The XENIX implementation of Icon is for SCO XENIX V. It will run on other XENIX systems, although the full extent of compatibility is not known. There are both SMM and LMM/FR implementations. It is inadvisable to run both the SMM and LMM/FR implementations on the same computer because of file-naming conflicts. The Icon program library is distributed separately. Source code for XENIX is not yet available. All XENIX diskettes are written in *tar* format.

The Porting Package: This package is intended for porting Icon to operating systems and computers on which it has not already been implemented. This version of Icon does not require a hierarchical file system. In addition to the source code, there are porting instructions and suites of test programs. The distribution is in MS-DOS ASCII format for ease of file transfer. Individual files are packaged in larger files to reduce the effort of file transfer and a C program for unpacking them is included. *Note:* Only 5-¼" diskettes are available. Persons requiring 3-½" diskettes will have to make arrangements for conversion.

The Documentation Package: This package contains a copy of the Icon language book, a description of recent additions to the language, a compilation of material from the programming corners of previous Icon Newsletters, and other material related to programming in Icon.

Order Form

Ship to:		Return this form with payment to:
Name	____________	Icon Project
Address	____________	Department of Computer Science
	____________	The University of Arizona
	____________	Tucson, AZ 85721
Telephone	____________	

UNIX Icon (9-track magnetic tape) options: ☐ tar ☐ cpio ☐ 1600 bpi ☐ 6250 bpi	$25	________
VAX/VMS Icon (9-track magnetic tape) options: ☐ 1600 bpi ☐ 6250 bpi	$25	________
Macintosh Icon (3-½" 1S diskette)	$15	________
MS-DOS Icon, SMM (5-¼" 2S/DD diskette)	$15	________
MS-DOS Icon, LMM/ER (5-¼" 2S/DD diskette)	$15	________
MS-DOS Icon, LMM/FR (5-¼" 2S/DD diskette)	$15	________
MS-DOS Icon, source code (2 5-¼" 2S/DD diskettes)	$25	________
MS-DOS Icon Program Library (5-¼" 2S/DD diskette)	$15	________
PC/IX Icon (5-¼" 2S/DD diskette)	$15	________
PC/IX Icon source code (4 5-¼" 2S/DD diskettes)	$35	________
PC/IX Icon Program Library (5-¼" 2S/DD diskette)	$15	________
XENIX Icon, small-memory model (5-¼" 2S/DD diskette)	$15	________
XENIX Icon, large-memory model (5-¼" 2S/DD diskette)	$15	________
XENIX Icon Program Library (5-¼" 2S/DD diskette)	$15	________
Icon Porting System (4 5-¼" 2S/DD diskettes)	$35	________
Documentation Package	$29	________
Other charges (see above)		________
Total		________