The Icon Newsletter

No. 41 – March 15, 1993

Contents

New Icon Program Library

The Icon program library is a major resource for Icon programmers. There are two main parts to the library: complete programs and collections of procedures. The programs are useful in their own right and do not require a knowledge of Icon, although several of them are useful in building and processing Icon programs. The procedures provide a large computational repertoire to augments Icon's built-in one.

In addition to the functionality of the library, it provides extensive examples of coding style and techniques used by many different programmers. As such, it is an excellent source of examples for novice Icon programmers.

The Icon program library has grown considerably since the last release in 1991. The new library contains 167 complete programs — everything from text utilities to games. There are 1,158 additional procedures for use in other programs — again, just about anything you could imagine. In all, the library contains more the 2.5MB of source material.

There is now a portion of the library devoted to X-Icon. The X-Icon applications include a direct-manipulation interface builder, several color selection tools, an intelligent font selector, a capable text editor, bitmapped graphics editors, and so on. Procedures include an interface toolkit, turtle graphics, geometrical transformations, and so on.

The library is the same for all platforms. We're providing three distribution formats: MS-DOS, Macintosh, and UNIX. If you're working on another platform, you should be able to convert one of these formats.

If you get the new library, you might also consider subscribing to the update service for it. Library updates are sent to subscribers three or four times a year (present subscribers to this update service received all the material in the new release some time ago). Subscribers to the update service also get additional material that's not quite in shape for the library but nonetheless may be useful.

Note: We've been distributing this new version of the library for several weeks. If you got the library recently, you may have received the new version. It's identified as Version 8.8.

Supporting the Icon Project

The Icon Project is expensive to run. In addition to the costs of printing and mailing the *Newsletter*, we sometimes need to upgrade the software we use to produce new versions of Icon. Occasionally we also have to repair or upgrade the personal

computers that are dedicated to the Icon Project.

Fortunately, we don't have personnel expenses. The Department of Computer Science here provides technical, secretarial, and clerical support, as well as working and storage space. In addition, many persons freely volunteer their time for tasks ranging from new implementations to documentation.

We get income from sales of Icon program material, books, and subscriptions. Since personnel costs for distribution are underwritten, our income from such sales exceeds the costs for media and shipping.

Of course, we also make Icon program material and most documentation freely available for electronic transfer via FTP and our RBBS. Recently many persons who formerly purchased Icon on diskettes and tapes have obtained FTP access. The effect on our distribution has been dramatic. The amount of Icon material downloaded by FTP has skyrocketed — over 4,500 files in January of this year alone. At the same time, purchase of Icon material has declined markedly. We're suffering "Death by FTP".

There are several things we could to try to do to deal with this problem, including reducing the frequency of the *Newsletter*, charging a subscription fee for it, or providing it only in machine-readable form for downloading. Such measures are unattractive for a variety of reasons and also are unlikely to solve the basic problem.

Instead, we hope that some of you who benefit from Icon and hope to see it continue will be willing to provide financial support.

We'd prefer support in a form that is beneficial both to you and the Icon Project. The best way to do this is to purchase something that's of value to you and also provides revenue for us. For example, a subscription to the Icon Analyst is only $25 a year and brings you interesting technical material about Icon every two months. A subscription to updates to the Icon program library gives you the latest programs and procedures long before a new version of the library is officially released. If you're interested in the source code for Icon and building your own version, a source-code update subscription gives you periodic updates with the latest improvements and new features.

If you've downloaded Icon material that you otherwise might have purchased, you can help by making a small payment toward supporting our electronic distribution facility. Or you can add a little extra when you're ordering Icon material. We've provided a place for this on our order form.

In the Works ...

We never leave things alone. We've now added a built-in preprocessor to Icon. It's comparatively simple, but it has several features to make writing Icon programs easier: file inclusion, constant definitions, and conditional compilation. We hope to have this new version of Icon available for distribution sometime this summer.

We've also been working on X-Icon, adding a few new features, removing redundant ones, and so on. We've put a lot of effort into reorganizing the implementation so that it will be easier to port to new platforms. We have our sights set on a 32-bit implementation of X-Icon that will run under Windows 3.1 and NT. Stay tuned.

Moving?

If you're moving, please let us know your new address. We use bulk rate for the *Newsletter* in the United States and we usually aren't informed of address changes or undeliverable mail. The situation is not much better for other countries. Although we use first-class mail outside the United States, notification of delivery problems is spotty. The result is that we not only lose track of subscribers who have moved, but we continue to send undeliverable mail, sometimes indefinitely.

You can let us know your new address any way that you like — by postal mail, electronic mail, fax, or telephone. See the box on page 5. But do let us know.

Downloading Icon Material

Most implementations of Icon are available for downloading electronically:

RBBS: (602) 621-2283

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

FTP Files by Electronic Mail

If you have access to electronic mail but not to FTP, you now can get files from our FTP area through electronic mail. Using a facility called ftpmail, you can send a script of commands to our FTP site. Material specified in these commands is sent back to you. You can get both directory listings and files. Large files are automatically split into smaller pieces to facilitate transfer.

To use ftpmail, send a message to

```
ftpmail@cs.arizona.edu
```

Your message must begin with the command open and end with the command quit. (The subject field of the message is ignored.) There are a dozen or so commands altogether. We'll list only the most basic ones here. You can get more information by sending the following message:

```
open
help
quit
```

You'll get a UNIX-style manual page with all the particulars by return e-mail.

If you want the reply to go to a different address than the one you're sending from, include the reply-to command in your message, as in

```
reply-to georgem@ece.foodom.edu
```

In fact, if you try ftpmail and don't get a reply, it's probably because the return address as it appears in your message header doesn't work. (We find that to be the case fairly frequently in mail sent to icon-project.) If this happens, use reply-to, being careful to give a proper address.

In order to get directory listings and files, you'll need to specify where they are. The cd (change directory) command does this. To get to Icon material, start with

```
cd /icon
```

A directory listing of this area then can be obtained with

```
dir
```

The command get is used to get a file. In the Icon FTP area, there are READ.ME files in all (or almost all) directories. Thus,

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

gets you our top-level READ.ME file.

Binary files, such as archives and executables, are automatically encoded as ASCII text to allow transfer via e-mail. The default encoding is uuencode. btoa encoding also is available. The commands btoa and uuencode can be used to specify the encoding you want. You'll need either uudecode or atob to convert encoded files back into their binary form. The Icon program library contains a program iidecode that has the same functionality as uudecode. It's slow, but it should work on any platform.

Give ftpmail a try. If you have problems, send e-mail to icon-project@cs.arizona.edu for assistance.

ProIcon Price Reduction

The ProIcon Group has reduced the price of ProIcon Version 2.0 to $95.

ProIcon includes:

- all features of Version 8.0 of Icon
- a standard Macintosh interface
- an integrated editor
- functions for creating dialog boxes, manipulating windows and the clip board, navigating through folders, and performing other Macintosh operations
- access to HyperCard XFCNs and XCMDs
- royalty-free applications
- a separate application for visualizing storage management
- a comprehensive, 367-page manual

ProIcon is 32-bit clean and is compatible with Version 7.0 of the Macintosh operating system.

ProIcon can be ordered from:

Catspaw, Inc.

P.O. Box 1123
Salida, CO 81201-1123

voice: (719) 539-3884
fax: (719) 539-4830

Add $5 domestic, $10 Canada, $30 all other for shipping.

Noun Stem Generation of Finnish

Editors' Note: The following article, which describes an application of Icon in linguistics, was contributed by Kimmo Kettunen. Kettunen's e-mail address is kettunen@delphi.com.

Introduction

The rich morphology of the Finnish language has been studied intensively in computational linguistics during the last decade or so. In the beginning of the 1980s, computational morphology had its major breakthrough in the work of Koskenniemi (1983), who introduced his so-called two-level (TWOL) model of computational morphology of Finnish. Since those times there have been at least 20 different kinds of morphological programs that at some level either analyze or synthesize Finnish word forms or do both (Kettunen 1992).

I have also contributed to this computerized morphological invasion by programming a prototype of a Finnish noun stem generator, which I have named Stemma (a colloquial Finnish form of the English word stem). The program is able to produce all the differing major and minor stems for an input noun that is given to it.

The program is implemented in Icon, which has not been used much in computational linguistics, although its characteristics are very well suited at least for prototypes and non-commercial research tools. Thus Stemma also shows the relevance of Icon to the CL community.

Structure of the Program

The working of the program is based on string and substring matching of the input word. The program analyzes the word from its end and, according to its characteristics, sends the processing to proper subprocedures. The most important subprocedures are grade alternation, plural formation, and 12 pattern procedures of which each can handle one or several types of nouns (mostly one type). The program has also seven small inherent vocabularies, which contain either all the tokens of exceptional small paradigms or exceptional words that are not affected by the grade-alternation rules. Altogether some 650 - 700 words are included in the vocabularies or in the rules themselves. Otherwise the program relies only on substring matching and pattern analysis and does not need comprehensive lexicons or paradigm markings of words.

The main program of Stemma reads the input, splits compounds to parts, counts syllables, makes the basic stem and character variable definitions, and calls the other procedures. The actual processing of the word is begun by the patterns procedure, which first decides whether the word needs to be handled in grade-alternation procedures. If grade alternation seems necessary, the word is sent to an appropriate alternation procedure, either for weakening or strengthening. Otherwise it is matched against different pattern rules, which decide whether any of the 12 special pattern procedures have to be invoked. After this, the word is given to plural formation, which may first define the nature of the first vowel, if the word ends with -a or -ä. After that, plural stems are formed according to the plural rules.

When plural stems have been formed, the processing is finished and the resulting stems can be given out. When the program stops, it also produces a short list of statistics, which states how many words it has processed, how many stems were produced, and the arithmetic mean of the amount of stems per input word. The processing time used and the mean processing time per word also are given. The statistical output of the program could be enriched very easily to cover different needs of usage. It was implemented in the first place to help in debugging the program, but since it proved useful also in other respects, it was embodied in the final version.

The maximal amount of different stem forms produced by Stemma for the input noun is five, which includes the input form in the nominative case. If the input noun is lapsuus ('childhood'), the program returns the stem forms lapsuus (nominative), lapsuude (weak stem), lapsuute (strong stem), lapsuut (consonant stem), and lapsuuksi (plural stem). Usually a Finnish noun has two or three different stem forms.

All 30 different possible case forms of Finnish (15 both in singular and plural) are formed using these stems, and once the alternations taking place in the stems have been produced, full noun forms can be produced easily by appending the case endings to proper stems. If the full possibilities of Finnish noun declination are considered, a noun

theoretically may have over 2,000 different forms, when different clitic and possessive forms and their combinations are used. But since all these different forms are formed using the small amount of stems, a vast number of different word forms can be covered by having the stems of a noun produced.

Linguistic Coverage of the Program

Stemma now covers the generation of Finnish noun stems almost completely. Adjectives also are covered fairly well. Only comparative stems and alternations that are category specific to some types of multisyllabic adjectives are not handled yet. These could be covered easily in the system, if the adjectival category was specified in the input word with a tag (for example, kova*, 'hard', matala*, 'shallow'). Pronouns and numerals are outside the program's present scope.

Normal compounds are handled correctly, provided the last part of the compound is separated from the rest with / (for example, avio/liitto, 'marriage'). If the last part has not been separated, the result may be right or wrong depending on the word. The program does not use comprehensive dictionaries. No other way of separating the parts of compounds is reliable, while formation of compounds is a pretty loosely structured process. Some other restrictions apply also, and they are described in Kettunen (1992).

I have made several test runs on different amounts of basic noun forms with Stemma. My largest test file includes 3,736 nouns, which have been randomly selected from a corpus of some 35,000 non-compound words. When this test file is run, Stemma gives 12,345 different forms as output (where of course 3,736 forms are the same as the input). The percentage of erroneous forms in this sample is less than one (disregarding the results of some adjectives, plurale tantums and opaque historical forms). Other smaller test corpora (with 1,000 and 133 different basic word forms) also have given similar results, and it seems that the accuracy and coverage of the program is somewhere between 98 - 99.6 per cent. Stemma is thus a very robust and reliable production-quality prototype program for Finnish noun stem generation.

Technical Information and Choice of the Implementation Language

Stemma is implemented in Icon. I chose Icon mainly because I was frustrated with other languages, which seemed to take too much effort in trivialities like variable typing and so forth and did not have good string-processing capabilities. Particularly of delight to me during the work were Icon's good ready-made string-manipulation functions, high-level design, ease of programming, and its rich repertoire of data structures. Untyped variables of the language were

The Icon Newsletter

Madge T. Griswold and Ralph E. Griswold
Editors

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

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

voice: (602) 621-8448
fax: (602) 621-4246

Electronic mail may be sent to:

icon-project@cs.arizona.edu

or

...uunet!arizona!icon-project

THE UNIVERSITY OF
ARIZONA
TUCSON ARIZONA

and

The Bright Forest Company
Tucson Arizona

also suitable for this kind of loosely structured work, where new variables often are needed in the midst of programming. I believe that programming Stemma was easier and faster using Icon than it would have been in other more conventional languages.

The size of Stemma is now about 23 kilobytes of source code, which makes some 46 kilobytes of compiled icode. The whole program has about 450 non-empty and non-commented lines which include also the seven inherent vocabularies. The amount of code could still be reduced heavily, if more abstract pattern-matching procedures were used (I used mainly find(), upto(), and match()). The matching rules are now very concrete and specific, which makes maintaining of the program easy. This may, however, lead to loss of generalization and inefficiency, and a more general approach could benefit the program. As Icon is a very high-level language, its characteristics certainly would be well suited to a more abstract or meta-level description of the linguistic rules and processes involved. This would also need deeper expertise in programming, however, and thus it is outside the present scope of my project. While I am a linguist, I am not a professional programmer.

The implementation of Stemma was done first with Icon's version 7.5, but it runs as well on any version from 8.0 up with no modifications up- or downwards. On an ordinary 12-MHz AT clone the program handles some 6 to 8 nouns per second depending on the machine and the complexity of processing caused by the words. Stemma runs on IBM compatible MS-DOS machines that have at least 640 kilobytes of RAM, but it is also easily portable to any computer that runs Icon, including mainframes and workstations. I have tried the program also in VAX/VMS and had no troubles.

An Example of Linguistic Rules Written in Icon

The procedure shown on the next page takes care of the normal grade alternation, where stops (k,p,t) and some two letter combinations that include stops (for example, lt, rt, mp) alternate under certain conditions. The alternation has been modeled in such a way that if alternation is possible (that is, a stop is found in the second last position and it is followed by a vowel and preceded by a voiced sound), the alternating sequence is firstly marked as an uppercase character (2). These are then processed by rules from 3 to 18 which specify what should happen to the sequence. (Either it weakens, which may occur as loss or change, or in exceptional cases it returns to the original sound.)

Those exceptional words, which do not undergo grade alternation, are included in the vocabulary named sanastot_6(). The procedure first checks it, and if the word is included there, it is returned unaltered (1). The vocabulary of exceptional words includes the most common words (such as auto, 'a car'), first names, colloquial words, and some loan words.

The grade-alternation rules include also some odd looking formulations like 4, 6, and 15, which are actually crude exception rules to reduce the size of the exception vocabulary. The rules may look awkward, but these kinds of "pragmatic" rules are the only possibility if you do not want to increase the size of the exception vocabulary.

The flow of the rule application is partially ordered such that the most general rule is tried last if no other rules have applied. Otherwise the application of the rules usually is not dependent on particular order if the rule contexts (that is, character strings including the alternating stop) differ from each other. For example, Rule 7 is the last and most common rule, which handles grade alternation of t, and it changes t to d (as in kita -> kida, 'jaws').

If the input word of Stemma were rotta ('a rat'), grade alternation Rule 3 would apply, and return the weak stem form rota. If the input word were lupa ('permission'), Rule 10 would match and return the weak form luva.

References

Kettunen, Kimmo. "Stemma, a Robust Noun Stem Generator for Finnish", *Humanistiske Data* 1: 26 - 31 (1991).

Kettunen, Kimmo. "Doing the Stem Generation with Stemma", *Proceedings of the 18th Finnish Linguistic Symposium*, Joensuu, Finland. 1992

Koskenniemi, Kimmo. *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production*. Publications of the Department of General Linguistics, University of Helsinki, No. 11 (1983).

```
procedure grade_alt2()
   change := ""; sanastot_6(); if member(gradevar_1, wo)                    # 1
   then return
   if (upto(klus, wo, -2) & upto(vow, wo, -1) & upto(sll, wo, -3) |         # 2
     (upto(klus, wo, -2) & wo[-2] == wo[-3]))
   then wstem[-2] := change := map(wstem[-2], "kpt", "KPT")
   if find("kK" | "tT" | "pP", wstem) & upto(reso, wstem, -4)               # 3
   then return(wstem[-2] := "")
   if wstem[-2] == "T" then
   {
     if find("tinTa" | "roTa", wstem) then                                  # 4
     return(wstem[-2] := "t")
     if find("lT" | "rT" | "nT", wstem, -3) & upto(vow, wstem, -4) then     # 5
        return(wstem [-2] := wstem[-3])
     if find("aaTi", wstem) & *wo >= 6 then                                 # 6
       return(wstem[-2] := "t")
     return(wstem[-2] := "d") }                                             # 7
     if wstem[-2] == "P" then
     {
       if find("aPi" | "oPi" , wstem) then                                  # 8
         return(wstem[-2] := "p")
       if find("mP", wstem) & upto(vow, wstem, -4) then                     # 9
         return(wstem [-2] := wstem[-3])
       return(wstem[-2] := "v") }                                           # 10
      if wstem[-2] == "K" then
      {
        if find("puKu" | "kyKy" | "suKu" | "luKu", wstem) then              # 11
          return(wstem[-2] := "v")
        if match("aiKa"|"poiKa", wstem) then                                # 12
          return(wstem[-3:-1] := "j")
        if find("nK", wstem) & upto(vow, wstem, -4, ) then                  # 13
          return(wstem[-2] := "g")
        if find("lKi" | "rKi" | "ylKā", wstem) then                         # 14
          return(wstem[-2] := "j")
   if find("hK" | "aaKi" | "haKi" | "kaKi" | "uuKi" | "yyKi" | "luKi" | "muKi" | # 15
     "ōōKi" | "iiKi" | "loKi" | "niKa" | "riKa" | "tiKa", wstem) then
       return(wstem[-2] := "k")
   if find("rK" | "lK", wstem) then                                         # 16
     return(wstem[-2] := "")
   if upto(vow, wstem, -1) & upto(vow, wstem, -3) &                         # 17
      wstem[-3] == wstem[-1] & find("VV", can, -4) then
        return(wstem[-2] := "\'")
   return(wstem[-2] := "") }                                                # 18
end
```

Procedure for Normal Grade Alternation

Programming Corner

It's common practice when debugging a program to insert a write() expression where trouble is expected, as in

```
write("x=", x)
```

That works well enough as long as you can be sure of the type of x and that it's something write() can convert to a string. But it's not very discriminating — you get a blank line if x is an empty string, an empty cset, or null. And there's no way to tell whether x is an integer or a string that happens to consist of digits. And if x happens to be something that can't be converted to a string, such as a list, you get a run-time error for your trouble.

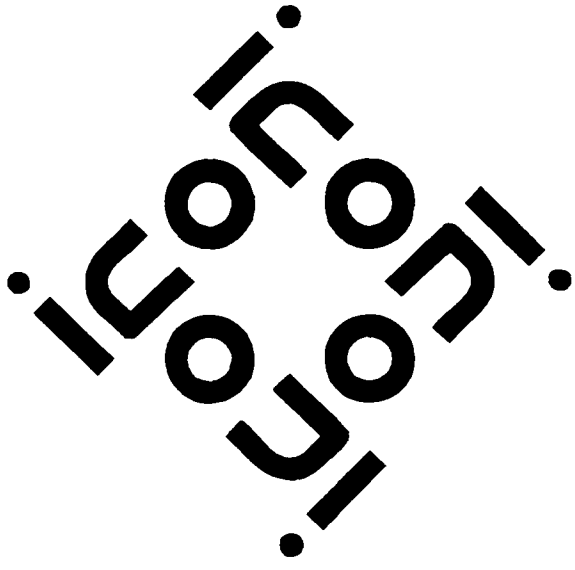
The function image() takes care of all of these problems. In the first place, it's safe: image(x) always produces a string, regardless of the type of x. Furthermore, image(x) is designed so that you can tell what the type of x is. Strings are enclosed in double quotes, csets in single quotes, while integers and real numbers are not quoted. A value that corresponds to a keyword is imaged with the keyword name. For example, the image of the null value is &null.

In the case of structures, the type, a serial number, and the size are given. For example, the image of an empty list might be list_15(0). The 15 is its serial number; the fifteenth list created since beginning of program execution. The number in parentheses is the size, zero in this case.

image() has other useful features. For example, the image of a value of type procedure shows whether it's built in (function) or declared (procedure). There are other things you might want to know about different kinds of images. See the Icon book [1]. In any event, it's worth casting diagnostic output in a form such as

```
write("x=", image(x))
```

Incidentally, several very capable extensions to the built-in image() function are included in the Icon program library. See fullimag.icn, image.icn, and ximage.icn.



Reference

1. *The Icon Programming Language*, second edition, Ralph E. Griswold and Madge T. Griswold, Prentice Hall, Englewood Cliffs, New Jersey, 1990. pp. 56-57, 128-129.

Third Icon Workshop

The Third Icon Workshop was held September 10-11, 1992 in La Jolla, California. Fifteen persons involved in the development and use of Icon met to present recent work, to discuss their interests, and to plan future directions.

Presentations included X-Icon, a multi-tasking version of Icon, the Icon compiler, ISIcon, program visualization, an X-Icon toolkit and interface builder, real-time garbage collection, and a dialect of Icon for instruction.

An eight-page report summarizing the meeting is available, free of charge, with any order of Icon material amounting to $15 or more. Just ask for IPD206 or the "Third Workshop Report".

We regret that we can't provide documents like this free of charge to everyone. There just are too many of you and costs of distributing reports add up quickly. All reports related to Icon are available, however, for the cost of reproduction and shipping. For a list of what's available, ask for IPD117 — it's free to anyone.

Ordering Icon Material

What's Available

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

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

Icon Program Material

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

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

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

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

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

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

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

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

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

Documentation

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

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

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

Payment

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

Prices

The prices quoted here are good until April 30, 1993. After that, prices are subject to change without further notice. Contact the Icon Project for current pricing information.

Extra Payment

If you wish to support the Icon Project by making an additional payment, a line is provided at the bottom of the order form for this.

Versions

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

Ordering Instructions

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

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

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

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

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

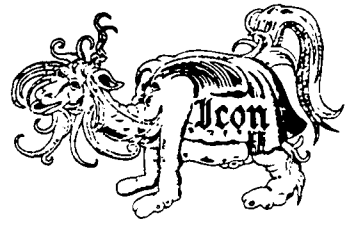
Ordering Codes: When filling out the order form, use the codes given in the second column of the list to the right (for example, AME, ATS, ...).

Executables				
Acorn Archimedes (8.0)	ARE	or	$15	[$5]
Amiga (8.0)	AME		$15	[$5]
Atari ST (8.0)	ATE	[1]	$15	[$5]
MS-DOS (8.8)	DE	or	$15	[$5]
MS-DOS 386/486 (8.8)	DE-386	or	$15	[$5]
Macintosh (8.0)	MET		$15	[$5]
Macintosh/MPW (8.8)	MEM		$15	[$5]
OS/2 (8.8)	OE	or	$15	[$5]
Source				
Amiga (8.0)	AMS		$15	[$5]
Atari ST (8.0)	ATS		$15	[$5]
MS-DOS & OS/2 (8.8)	DS	or	$30	[$5]
Macintosh (8.0)	MST		$15	[$5]
Macintosh/MPW (8.8)	MSM		$25	[$5]
MS-DOS updates (5)	SU	or	$60	[$15]
Complete Systems (these contain earlier versions of the library)				
CMS (8.0)	CT		$30	[$10]
MVS (8.0)	MT		$30	[$10]
UNIX (8.7)	UD	[2]	$25	[$5]
UNIX (8.7)	UT		$30	[$10]
UNIX (8.7)	UC		$45	[$10]
VMS (8.7)	VT		$32	[$11]
Program Library				
MS-DOS (8.8)	DL ↤	or	$15	[$5]
Macintosh (8.8)	ML ↤		$15	[$5]
UNIX (8.8)	UL ↤	or	$15	[$5]
MS-DOS updates (4)	LU-D	or	$30	[$12]
Macintosh updates (4)	LU-M		$30	[$12]
UNIX updates (4)	LU-U	[2]	$30	[$12]
Books				
The Icon Programming Language	LB		$40	[$13]
The Implementation of Icon + update	IB		$53	[$14]
Icon Programming for Humanists + diskette	HB		$38	[$10]
Newsletters				
The Icon Newsletter (complete, 1-40)	INC		$18	[$5]
The Icon Newsletter (back issues, each)	INS		$1	[$2[3]]
The Icon Analyst (1 year, 6 issues)	IA		$25	[$10]
The Icon Analyst (back issues, each)	IAS		$5	[$2[3]]

[1] 400K.

[2] 1.44M.

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



Order Form

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

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

name _____

address _____

_____

city _____ state _____ zipcode _____

(country) _____ telephone _____

❒ check if this is a new address

qty.	code	description	price	shipping*	total
	XP	Support for the Icon Project			

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

Make checks payable to The University of Arizona

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

❒ Visa ❒ MasterCard ❒ Discover ❒ check or money order

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

card number | | | | | | | | | | | | | | | | exp. date | | | |

name on card (please print) _____

signature _____

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