Version 8 of Icon for MS-DOS

Ralph E. Griswold

Department of Computer Science, The University of Arizona

1. Introduction

Version 8 of Icon for MS-DOS should run on any computer running MS-DOS 3.0 or higher. Math coprocessors are supported and used if present; otherwise software emulation is used. Approximate 500K of free RAM is needed to run Icon satisfactorily. Two versions of Icon's executor system are provided. One supports arithmetic on integers of unlimited magnitude and the other does not. The latter is smaller and can be used if you do not have enough RAM for the former.

Version 8 of Icon for MS-DOS is distributed on 5.25" or 3.5" diskettes, which include executable binary files, a few test programs, and documentation in machine-readable form. Printed documentation is included with diskettes distributed by the Icon Project at the University of Arizona.

This implementation of Icon is in the public domain and may be copied and used without restriction. The Icon Project makes no warranties of any kind as to the correctness of this material or its suitability for any application. The responsibility for the use of Icon lies entirely with the user.

2. Documentation

The basic reference for the Icon programming language is the book

> *The Icon Programming Language*, second edition, Ralph E. Griswold and Madge T. Griswold, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1990. 365 pages. ISBN 0-13-447889-4. $29.95.

This book is available from the Icon Project at the University of Arizona. It also can be ordered through any bookstore that handles special orders or by telephone directly from Prentice-Hall: (201) 767-9520.

Note that the first edition of this book, published in 1983, describes an older version of Icon and does not contain information about many of the features of Version 8.

A brief overview of Icon is contained in technical report TR 90-6 [1] (tr90-6.doc on the distribution diskette). Features that have been added to Icon since the book was written are described in TR 90-1 [2] (tr90-1.doc on the distribution diskette). These technical reports, together with this document (ipd132.doc on this diskette), provide enough information to write and run simple Icon programs, but persons who intend to use Icon extensively will need the book.

3. Installing MS-DOS Icon

Two executable binary files are needed to run Icon:

icont.exe	translator
iconx.exe	executor

These files should be located at a place on your PATH specification. As mentioned in Section 1, there are two forms of iconx: one without large-integer arithmetic and one with it. The former is named iconx.exe as distributed, while the latter is named iconxl.exe.

The distribution is contained in several files in ARC format. A copy of pkxarc.com is included for dearchiving. The distribution files are:

pkxarc.com	dearchiving utility
icon.arc	executable binary files [474KB]
samples.arc	Icon programs and data [3KB]
docs.arc	documents [97KB]
readme	installation overview and recent notes

The figures in brackets give the approximate amount of disk space needed when the files are extracted from their archives. (It is not necessary to keep both executors.)

To install the .exe files, set your current directory to the desired place, place the second distribution diskette in drive A, and dearchive the files there using pkxarc.com on the distribution diskette:

```
a:pkxarc a:icon.arc
```

The same technique can be used for extracting the remaining files on the other diskette.

The simplest way to use Icon is to pick the executor that fits your needs, place it at a location on your path, and rename it to iconx.exe if necessary. For example, if you want the executor that supports large-integer arithmetic, the following will do:

```
rename iconxl.exe iconx.exe
```

However, if you want to use both executors, you can either name them differently (such as they are named above) and use them by their different names, or you can name them the same but put them in different locations. In this case, you will have to change your path depending on which one you use.

4. Running MS-DOS Icon — Basic Information

Files containing Icon programs must have the extension .icn. Such files should be plain text files (without line numbers or other extraneous information). The icont translator produces an ‘‘icode’’ file that can be executed by iconx. For example, an Icon program in the file prog.icn is translated by

```
icont prog.icn
```

The result is an icode file with the name prog.icx. This file can be run by

```
iconx prog.icx
```

If your executor is named differently, simply use that name. For example, if your executor is named iconxl.exe, use

```
iconxl prog.icx
```

Alternatively, icont can be instructed to execute the icode file after translation by appending a −x to the command line, as in

```
icont prog.icn −x
```

This only works if your executor is named iconx.exe, since the −x option looks for this name. In the sections that follow, it is assumed that the executor is named iconx.exe.

If icont is run with the −x option, the file prog.icx is left and can be run subsequently using an explicitly named executor as described above.

The extensions .icn and .icx are optional. For example, it is sufficient to use

```
icont prog
```

and

```
iconx prog
```

5. Testing the Installation

There are a few programs on the distribution diskette that can be used for testing the installation and getting a feel for running Icon:

hello.icn — This program prints the Icon version number, time, and date. Run this test as

```
icont hello
iconx hello
```

Note that this can be done in one step with

```
icont hello -x
```

cross.icn — This program prints all the ways that two words intersect in a common character. The file cross.dat contains typical data. Run this test as

```
icont cross -x <cross.dat
```

meander.icn — This program prints the ''meandering strings'' that contain all subsequences of a specified length from a given set of characters. Run this test as

```
icont meander -x <meander.dat
```

roman.icn — This program converts Arabic numerals to Roman numerals. Run this test as

```
icont roman -x
```

and provide some Arabic numbers from your console.

If these tests work, your installation is probably correct and you should have a running version of Icon.

6. More on Running Icon

For simple applications, the instructions for running Icon given in Section 4 may be adequate. The icont translator supports a variety of options that may be useful in special situations. There also are several aspects of execution that can be controlled with environment variables. These are listed here. If you are new to Icon, you may wish to skip this section on the first reading but come back to it if you find the need for more control over the translation and execution of Icon programs.

6.1 Arguments

Arguments can be passed to the Icon program by appending them to the command line. Such arguments are passed to the main procedure as a list of strings. For example,

```
iconx prog text.dat log.dat
```

runs the icode file prog.icx, passing its main procedure a list of two strings, "text.dat" and "log.dat". The program also can be translated and run with these arguments with a single command line by putting the arguments after the -x:

```
icont prog -x text.dat log.dat
```

These arguments might be the names of files that prog.icn reads from and writes to. For example, the main procedure might begin as follows:

```
procedure main(a)
   in := open(a[1]) | stop("cannot open input file")
   out := open(a[2],"w") | stop("cannot open output file")
                        :
                        :
```

See also the information about the processing of command-line arguments given in Section 6.4.

6.2 The Translator

The icont translator can accept several Icon source files at one time. When several files are given, they are translated and combined into a single icode file whose name is derived from the name of the first file. For example,

```
icont prog1 prog2
```

translates the files prog1.icn and prog2.icn and produces one icode file, prog1.icx.

A name other than the default one for the icode file produced by MiconfR can be specified by using the −o option, followed by the desired name. For example,

```
icont −o probe.icx prog
```

produces the icode file named probe.icx rather than prog.icx.

If the −c option is given to icont, the translator stops before producing an icode file and intermediate ''ucode'' files with the extensions .u1 and .u2 are left for future use (normally they are deleted). For example,

```
icont −c prog1
```

leaves prog1.u1 and prog1.u2, instead of producing prog1.icx. These ucode files can be used in a subsequent icont command by using the .u1 name. This saves translation time when the program is used again. For example,

```
icont prog2 prog1.u1
```

translates prog2.icn and combines the result with the ucode files from a previous translation of prog1.icn. Note that only the .u1 name is given. The extension can be abbreviated to .u, as in

```
icont prog2 prog1.u
```

Ucode files also can be added to a program using the link declaration in an Icon source program as described in [2].

Icon source programs may be read from standard input. The argument − signifies the use of standard input as a source file. In this case, the ucode files are named stdin.u1 and stdin.u2 and the icode file is named stdin.icx.

The informative messages from the translator can be suppressed by using the −s option. Normally, both informative messages and error messages are sent to standard error output.

The −t option causes &trace to have an initial value of −1 when the icode file is executed. Normally, &trace has an initial value of 0.

The option −u causes warning messages to be issued for undeclared identifiers in the program.

Icon has several tables related to the translation of programs. These tables are large enough for most programs, but translation is terminated with an error message, indicating the offending table, if there is not enough room. If this happens, larger table sizes can be specified by using the −S option. This option has the form −S[cfgilnrstCFL]*n*, where the letter following the S specifies the table and *n* is the number of storage units to allocate for the table.

c	constant table	100
f	field table	100
g	global symbol table	200
i	identifier table	500
l	local symbol table	100
n	line number table	1000
r	record table	100
s	string space	20000
t	tree space	15000
C	code buffer	15000
F	file names	10
L	labels	500

The units depend on the table involved, but the default values can be used as guides for appropriate settings of −S options without knowing the units. For example,

```
icont −Sc200 −Sg600 prog
```

translates prog.icn with twice the constant table space and three times the global symbol table space that ordinarily would be provided.

6.3 Environment Variables

When an icode file is executed, several environment variables are examined to determine execution parameters. The values assigned to these variables should be numbers.

Environment variables are particularly useful in adjusting Icon's storage requirements. This may be necessary if your computer does not have enough memory to run programs that require an unusually large amount of data. Particular care should be taken when changing default sizes: unreasonable values may cause Icon to malfunction.

The following environment variables can be set to affect Icon's execution parameters. Their default values are listed in parentheses after the environment variable name:

TRACE (undefined). This variable initializes the value of &trace. If this variable has a value, it overrides the translation-time −t option.

NOERRBUF (undefined). If this variable is set, &errout is not buffered.

STRSIZE (65000). This variable determines the initial size, in bytes, of the region in which strings are stored.

HEAPSIZE (65000). This variable determines the initial size, in bytes, of the region in which Icon allocates lists, tables, and other objects.

COEXPSIZE (2000). This variable determines the size, in 32-bit words, of each co-expression block.

MSTKSIZE (10000). This variable determines the size, in words, of the main interpreter stack.

QLSIZE (5000). This variable determines the size, in bytes, of the region used by the garbage collector for pointers to strings.

The maximum region size that can be specified is 65000.

6.4 Details of Command-Line Processing

The processing of command-line arguments normally is important only in running iconx. Some details follow:

- An argument containing white space must be enclosed in double quotes ("). For example, "abc def" is a valid argument. The quotes are deleted in the string passed to iconx.
- If a quoted argument contains a double quote, the embedded quote must be preceded by a backslash. For example, "abc de\"f" is a valid argument. The outer quotes and the backslash are deleted in the string passed to iconx.
- If a non-quoted argument contains a double quote, the embedded quote can be preceded by a backslash or not. For example, abc\"d and abc"d are equivalent arguments.
- To pass a double quote as an argument, precede it with a backslash.
- Wild-card characters are not expanded.
- Quote-balancing errors are not checked. The start-up routine scans the residual command line from left to right.

7. Features of MS-DOS Icon

MS-DOS Icon supports all the features of Version 8 of Icon, with the following exceptions and additions:

- Pipes are not supported. A file cannot be opened with the "p" option.
- There are two additional options for open: "t" and "u". The "t" option, which is the default, indicates that the file is to be translated into UNIX[*] format. All carriage-return/line-feed sequences are translated into

[*]UNIX is a trademark of AT&T Bell Laboratories.

line-feed characters on both input and output. The "u" option indicates that the file is to be untranslated. Examples are:

```
untranfile := open("test.fil","ru")
tranfile := open("test.new","wt")
```

For files opened in the translate mode, the position produced by seek() may not reflect the actual byte position because of the translation of carriage-return/line-feed sequences to line-feed characters.

- Path specifications can be entered using either a / or a \. Examples are:

```
A:\ICON\TEST.ICN
A:/ICON/TEST.ICN
```

- The following MS-DOS device names can be used as file names:

console	CON
printer	PRN LST LPT LPT1
auxiliary port	AUX COM RDR PUN
null	NUL NULL

For example,

```
prompt := open("CON","w")
```

causes strings written to prompt to be displayed on the console. Use of a null file name means no file is created.

MS-DOS Icon also has some functions in addition to the standard repertoire:

MS-DOS Interface Functions

Disclaimer: The following functions provide a gateway to functions provided by MS-DOS and ROM BIOS. They should be used with care, since Icon maintains strict control over its environment (although it uses standard MS-DOS interfaces and does not bypass MS-DOS or ROM BIOS in any way). The descriptions that follow are low-level descriptions. They assume knowledge of the Intel 8086 (8088, 80x86 etc.) architecture.

Int86(a):

Int86(a) generates a hardware interrupt. The input is a list of integer values: [interrupt number, ax, bx, cx, dx, si, di, es, ds]. It returns a list of the form [flags, ax, bx, cx, dx, si, di, es, ds].

Great care must be taken in using this function. Some things to watch out for are:

- Interrupt functions that alter the stack registers (SS or SP) or alter the MS-DOS memory chain (that is, allocate or deallocate memory in the MS-DOS memory region). When Icon expands memory, it expects the next allocation to be contiguous with the region it currently owns.
- Interrupt functions that operate on files opened by Icon's open(s) function — that is, closing, seeking, reading, or writing.
- The values of ES and DS may not be valid on return.

InPort(i):

InPort(i) returns an integer from hardware port i.

OutPort(i1,i2):

OutPort(i1,i2) writes value i2 to hardware port i1.

GetSpace(i):

GetSpace(i) allocates a static block of storage outside of Icon's direct control (that is, it is not be affected by garbage collection). This function simply calls the malloc() allocation routine and returns the address of the resulting block as a long integer.

FreeSpace(A):

FreeSpace(A) frees a static block of storage. A is a value returned by GetSpace(i). No check is make to verify that the block was allocated by GetSpace(i). The results are unpredictable if it was not. Arithmetic should not be performed directly on a value returned by GetSpace(i).

Peek(A,i):

Peek(A,i) builds a string pointing to the address specified by A with a length of i, where A is either an integer that specifies a linear address value or a list of the form [segment, offset] and i is a length (default 1).

This is the only way of 'seeing' the contents of a block of storage allocated by GetSpace(i). Consider the following example:

```
block := Peek(addr := GetSpace(100),100)
```

The value of block is a string of length 100 and addr is the linear address of the block.

Peek(A,i) does not move data into Icon's memory region. Instead, it builds a string qualifier that points to the data.

Poke(A,s):

Poke(A,s) copies a string s to location A, where A is an address specified in the same way as for Peek(A,s) and s is a string to be copied to that storage location. The string is copied directly into storage, byte by byte. No conversion is performed. This is the only way of assigning data to a block of storage allocated by GetSpace(i).

8. Known Bugs

A list of known bugs in Icon itself is given in [2].

Under MS-DOS some programs that require a lot of memory may hang when run by the −x option to icont. This can be avoided by using iconx in a separate step.

Also, if there is not enough free RAM, the system() function may fail silently or hang.

9. Reporting Problems

Problems with Icon should be noted on a trouble report form (included with the distribution) and sent to

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

(602) 621-4049

icon-project@cs.arizona.edu (Internet)
... {uunet, allegra, noao}!arizona!icon-project (uucp)

10. Registering Copies of Icon

If you received your copy of Version 8 of Icon directly from the Icon Project, it has been registered in your name and you will receive the Icon Newsletter without charge. This Newsletter contains information about new implementations, updates, programming techniques, and information of general interest about Icon.

If you received your copy of Version 8 of Icon from another source, please fill out the registration form that is included in the documents in the distribution) and send it to the Icon Project at the address listed above. This will entitle you to a free subscription to the Icon Newsletter and assure that you receive information about updates.

Acknowledgements

The design and development of the Icon programming language was supported, in part, by the National Science Foundation under grants MCS75-01397, MCS79-03890, MCS81-01916, DCR-8320138, DCR-8401831, and DCR-8502015.

Many individuals contributed to the design and implementation of Icon. The principal ones are Cary Coutant, Dave Gudeman, Dave Hanson, Tim Korb, Bill Mitchell, Kelvin Nilsen, Janalee O'Bagy, Gregg Townsend, Ken Walker, and Steve Wampler.

The implementation of Icon was adapted to MS-DOS by Cheyenne Wills in Mechanicsburg, Pennsylvania. He also provided some of the technical material contained in this document.

References

1. R. E. Griswold, *An Overview of Version 8 of the Icon Programming Language*, The Univ. of Arizona Tech. Rep. 90-6, 1990.
2. R. E. Griswold, *Version 8 of Icon*, The Univ. of Arizona Tech. Rep. 90-1, 1990.