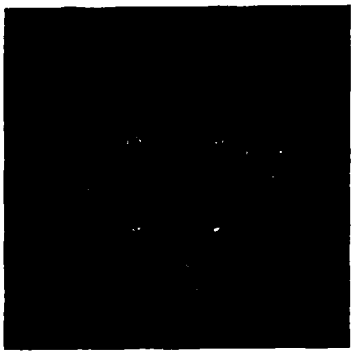
Version 8 of Icon for the Amiga

Clinton L. Jeffery



Department of Computer Science
The University of Arizona
Tucson, Arizona

IPD128a
April 11, 1996
http://www.cs.arizona.edu/icon/docs/ipd128.html

---

1. Introduction

This is an implementation of Version 8 of the Icon programming language for Amiga computers.

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.

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. 367 pages. ISBN 0-13-447889-4.

This book is available from the Icon Project at the University of Arizona.

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 (`ipd128.doc` on the distribution diskette), provide enough information to write simple Icon programs, but if you are going to use Icon seriously, you will need the book. You may also find it useful to examine the sample programs located on the distribution diskette.

2. Hardware and Software Requirements

This version was designed to run under V1.3 of the Amiga operating system software, and the rest of this document assumes you are using it. Amiga Icon version 8 has not been tested at all under V1.1 or V1.2. There is no fundamental reason it cannot be used with the older versions; if you want to run Icon with an older version of the operating system you may have problems

if Icon can't find a path to its executables or if you need to use environment variables.

This version of Icon is command-line based. It does not run under the workbench and does not have any support for Workbench icons or any of the special Amiga features. You should be able to use your choice of: the CLI (for older versions of AmigaDOS), the new AmigaShell, or some third party shell such as Matt Dillon's shell.

This version does not run on 512K machines. It was compiled and runs comfortably on a machine with 1 megabyte of memory (more is better). A hard disk is also essential for any serious programming work.

Since `icont` and `iconx` use the default system stack, they will not run if the stack is left at the default 4000 bytes AmigaDOS allocates for created windows. Before running icont or iconx from a CLI or Shell, you should increase the stack size. This is done by issuing a command such as

```
stack 15000
```

to the CLI from which you are going to run Icon. `icont` and `iconx` will halt with an error message if you try to run them with a stack size of less than 10000 bytes. A larger stack size will allow larger programs to be compiled and to run successfully.

3. Location of icont and iconx

The Icon system naturally follows the standard AmigaDOS search path to find executable files. That is, it looks first in the current directory, then in the `C:` directory, and finally in any other directories that you have added to the search path by means of the `PATH` command.

This means that you probably will want to place your icont and iconx commands in your `C:` directory if you have room there, or you can add the directory where the Icon binaries are located to your search path if you are running from the `ICON:diskette`.

Note that the Icon interpreter `iconx` has the same name as one of the AmigaDOS utilities. Care should be taken when locating `iconx` and setting the AmigaDOS search path; one can always rename Icon's `iconx` program to something else if the AmigaDOS `iconx` is needed. The AmigaDOS `iconx` is usually invoked from the Workbench, so this should not be too much of a problem.

4. Running Icon

To translate and link an Icon source program file named `myfile.icn` first you must add `Icon8.0:c` to your path (or copy the executables to some directory where AmigaDOS can find them), and then type

```
icont myfile
```

which translates and links `myfile.icn` and produces the file named `myfile`. You can then execute this by typing

```
iconx myfile
```

This can be done all in one step as

```
icont myfile -x
```

The usual standard input and standard output redirection is supported by the CLI, but note that the AmigaDOS standard is for the redirection to be given before the other arguments, thus:

```
iconx <infile >outfile myprogram myarg1 myarg2 ...
```

If the redirection arguments are not given immediately following the `iconx`, they do not work.

It is not possible to redirect the standard error output in this version but it is possible to combine the standard error output with the standard output by using the command line switch `-e -` as follows:

```
iconx >outfile -e - myprogram args args ...
```

All the output from the program including error messages is sent to outfile. Note that it is necessary to type the dash (-) following the switch. Just typing `-e` has no effect; you must type `-e -`. This switch is only allowed on the `iconx` command. It does not work on the `icont` command.

5. Testing the Installation

There are a few programs on the distribution diskette that may be used for testing the installation and getting a feel for running Icon. The programs and appropriate data files are located in the samples directory. Note that files containing Icon programs must have the extension `.icn`. Such files should be plain text files (without line numbers or other extraneous information). The command processor `icont` produces an icode file that can be executed by `iconx`. For example, an Icon program in the file `prog.icn` is translated and linked by

```
icont prog
```

The result is an icode file (binary) with the name `prog` (with the extension removed). This file can be run by

```
iconx prog
```

Alternatively, `icont` can be instructed to execute the program after translating and linking by appending a `-x` to the command line, as in

```
icont prog -x
```

In this case the file `prog` also is left and can be run subsequently using `iconx`.

Note that the `icont` and `iconx` programs are executable binary files and, as such their locations can be discovered by the normal AmigaDOS executable file path search mechanism. But the icode file produced by icont from the Icon source file is not a directly executable file. It essentially is a data file that is read by the Icon interpreter `iconx`. Therefore, the location of the

icode file is not known to AmigaDOS. There is no path search for data files. So your icode file (prog in the above examples) must either be in the current directory, or you must specify its location by giving the drive or directory path as part of the filename.

For example, if your current directory is RAM:, and you wish to run the Icon program prog.icn, which might be located in the directory df1:src, and you have placed icont in a location where AmigaDOS can find it, you would type

```
icont df1:src/prog -x
```

In this case, the icode file prog would be left behind in ram:, which was your current directory, after it has been run. You would then be able to run prog again by simply typing

```
iconx prog
```

The sample programs are:

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.dat cross -x
```

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.dat meander -x
```

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.

seive.icn
: This program produces prime numbers. Run this test as

```
icont <sieve.dat sieve -x
```

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

6. Special I/O

This version supports the standard AmigaDOS device names as files. Thus, to write directly to the printer from your Icon program do this:

```
fd := open("PRT:","w")
write(fd,"this will appear on your printer")
close(fd)
```

The existing CLI window is used for standard i/o unless these are redirected from the command line.

To use the CLI window for i/o when standard input or output has been redirected elsewhere, you can open it as file * as explained in the AmigaDOS manual, thus:

```
tty := open("*","b")
line := read(tty)
write(tty,line)
close(tty)
```

You can also open additional CON: or RAW: windows as described in the AmigaDOS manual, thus:

```
con := open("CON:10/10/100/100/my window","b")
write(con,"This appears in a new window")
close(con)
```

Note that as soon as you close a special window it disappears.

If you open a RAW: window for reading, notice that there is no echoing done for you and you must handle all the raw input as described in the AmigaDOS manual. RAW: i/o has not been extensively tested yet in this version, so it is possible that you may have problems. The AmigaDOS documentation is also notoriously misleading.

7. More on Running Icon

For simple applications, the instructions for running Icon given in Section 6 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.

7.1 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 uses. 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 8.4.

7.2 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 icont 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 link declarations in Icon source programs 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.

7.3 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 size, in bytes, of the region in which strings are stored.

BLKSIZE (65000)
: This variable determines the size, in bytes, of the region in which lists, tables, and other objects are stored.

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.

The special variable CHECKBREAK is used only in the Amiga version of Icon. If the line

```
CHECKBREAK=1
```

appears in your environment file, you should be able to interrupt the execution of an Icon program by typing Control-C or Control-D.

If this environment variable is not set, you may still be able to interrupt the program, but only if it is doing i/o. Icon runs significantly slower when CHECKBREAK is set but it is useful when developing programs.

An example of memory region resizing may be seen in the sample environment file `SmallIconEnv`.

9. Unsupported Feature

Large integer arithmetic is not provided for Amiga Icon because of the increased size of the executor needed to support this feature.

10. Known Bugs

A list of known bugs in Icon itself is given in [2]. Known bugs that are specific to the Amiga implementation of Icon are:

- The `atan2()` function simply divides its first argument by its second and calls `atan()`.

If this division causes overflow, the call to `atan2()` will produce incorrect results. This is due to the C compiler used to create Amiga Icon Version 8.

- Floating-point numbers are occasionally printed rather strangely. In addition, floating-point numbers in this implementation of Icon have a smaller range of legal values than in some other implementations of Icon. This is a result of the use of the Motorola Fast Floating Point (FFP) format in Amiga Icon Version 8.

- Specifying a directory name as an argument to `icont` or `iconx`, or in a call to Icon's `open()` function, may cause disaster. The implementation does not check to see if the file is a directory as it should.

11. 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
The University of Arizona
P.O. Box 210077
Tucson, AZ85721-0077
U.S.A.

(520) 621-6613 (voice)
(520) 621-4246 (fax)
icon-project@cs.arizona.edu

Acknowledgements

The design and development of the Icon programming language was supported, in part, by the National Science Foundation grants.

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

Rob McConeghy originally adapted Version 6 of Icon to run on the Amiga. Much of the Version 8 configuration for the Amiga is based on his work.

References

1. R. E. Griswold, *An Overview of Version 8 of the Icon Programming Language*, The Univ. of Arizona Tech. Rep. 90-6, 1994.

2. R. E. Griswold, *Version 8 of Icon*, The Univ. of Arizona Tech. Rep. 90-1, 1990.

Icon home page