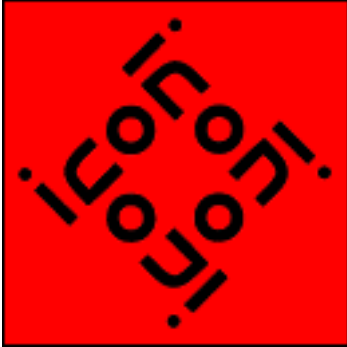


# Version 9 of Icon for 32-bit MS-DOS Platforms

Ralph E. Griswold



Department of Computer Science  
The University of Arizona  
Tucson, Arizona

IPD248b

March 24, 1996

<http://www.cs.arizona.edu/icon/docs/ipd248.html>

---

## 1. Overview

This implementation of Icon runs on MS-DOS in 32-bit protected mode. It was built using the Watcom C/389 9.0 compiler.

The minimum configuration to run this version is a 386SX with 2MB of RAM. 4MB or more is recommended. If only 2MB of RAM are available, the use of disk caching (for example, SMARTDRV) may cause problems and should be disabled.

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 Version 8 of Icon is the second edition of the book *The Icon Programming Language* [1]. This book is available from the Icon Project at The University of Arizona.

The new features of Version 9 of Icon are described in an accompanying technical report [2].

## 2. Installing MS-DOS Icon

Three executable binary files are needed to run Icon:

<code>icont.exe</code>	translator
<code>iconx.exe</code>	executor

These files should be located at a place on your `PATH` specification.

The distribution is contained in several files in LHA format. These files have the extension `lzh`. If you do not have a copy of `lha.exe`, execute the self-extracting archive `lha213.exe` on the distribution diskette. This will produce `lha.exe` and documentation.

The distribution files are:

```
docs.lzh      documents
icon.lzh     executable binary files
readme      installation overview and recent notes
samples.lzh  Icon programs and data
```

To install the `.exe` files, set your current directory to the desired place, place the appropriate distribution diskette in a drive, and dearchive the files there using `lha.exe`. For example, using drive A to dearchive the executable binary files, the following will do:

```
a:lha x a:icon.lzh
```

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

### 3. Running 32-Bit 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 Icon translator, `icont`, produces an "icode" file that is executed using `iconx.exe`. There are two forms of icode files:

1. Executable icode files that invoke `iconx.exe` automatically when they are run. These icode files have the extension `exe`.
2. Non-executable icode files that are run by `iconx` with the icode file as an argument. These icode files have the extension `icx`.

Non-executable icode files require about 8K less RAM to run than executable files, but they are more awkward to run. Note: Executable icode files do not stand alone; they require `iconx.exe` to run.

#### Producing Executable Icode Files

Executable icode files are produced by default. An Icon program in the file `prog.icn` is translated by

```
icont prog.icn
```

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

```
prog
```

The extension `.icn` is optional on the command line. For example, it is sufficient to use

```
icont prog
```

#### Producing Non-Executable Icode Files

To produce a non-executable icode file, use the option `-I`, as in

```
icont -I prog
```

The result is an icode file named `prog.icx`. It can be run by

```
iconx prog
```

Note that the `icx` extension is not necessary.

`iconx` will find an icode file if it is in the current directory or at place given on your `PATH` specification.

Except for the icode file extension and the method of running non-executable icode files, the remarks in the previous section apply.

## 4. 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
hello
```

`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
cross <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
meander <meander.dat
```

`roman.icn`

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

```
icont roman
roman
```

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.

## 5. More on Running Icon

For simple applications, the instructions for running Icon given in Section 3 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.

## 5.1 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. For example, the main procedure might begin as follows:

```
procedure main(args)
in := open(args[1]) | stop("cannot open file")
out := open(args[2]) | stop("cannot open file")
.
.
.
```

## 5.2 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.exe`.

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 prog
```

produces the icode file named `probe.exe` rather than `prog.exe`.

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

```
icont -c prog1
```

leaves `prog1.u1` and `prog1.u2`, instead of producing `prog1.exe`. These ucode files can be used in a subsequent `icont` command by using the `.u1` name. This saves translation time subsequently. 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 `.u2` name is implied. The extension can be abbreviated to `.u`, as in

```
icont prog2 prog1.u
```

Ucode files also can be added to a program using a link declaration.

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.exe`.

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.

### 5.3 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. Particular care should be taken when changing default values: unreasonable values may cause Icon to malfunction.

The following environment variables can be set to adjust 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` (500000)

This variable determines the size, in bytes, of the initial region in which strings are stored. If additional string regions are needed, they may be smaller.

`BLKSIZE` (500000)

This variable determines the size, in bytes, of the initial region in which lists, tables, and other objects are stored. If additional block regions are needed, they may be smaller.

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.

## 6. Features of MS-DOS Icon

MS-DOS Icon supports all the features of Version 9 of Icon, with the following exceptions and additions:

- Pipes are not supported. A file cannot be opened with the "p" option.
- 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.

## 7. 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, AZ 85721-0077  
 U.S.A.

(520) 621-6613 (voice)  
 (520) 621-4246 (fax)

[icon-project@cs.arizona.edu](mailto:icon-project@cs.arizona.edu)

## **Acknowledgements**

Many individuals contributed to the design and implementation of Icon. Clint Jeffery and Gregg Townsend collaborated with the author in the development of Version 9. This 32-bit MS-DOS implementation was contributed to the Icon Project by persons who wish to remain anonymous.

## **References**

1. R. E. Griswold and M. T. Griswold, The Icon Programming Language, Prentice-Hall, Inc., Englewood Cliffs, NJ, second edition, 1990.
2. R. E. Griswold, C. L. Jeffery and G. M. Townsend, Version 9.0 of the Icon Programming Language, The Univ. of Arizona Icon Project Document [IPD267](#), 1995.

---

[Icon home page](#)