Version 9.0 of the Icon Programming Language

Ralph E. Griswold, Clinton L. Jeffery, and Gregg M. Townsend

Department of Computer Science, The University of Arizona

1. Introduction

The current version of Icon is Version 9.0. The second edition of the Icon book [1] describes Version 8.0. This description is a supplement to that book.

Most of the language extensions in Version 9.0 are upward-compatible with previous versions of Icon and most programs written for earlier versions work properly under Version 9.0. The language additions to Version 9.0 are:

- a preprocessor
- an optional interface to graphic facilities (for platforms that support them)
- new functions and keywords
- several other changes and enhancements

There also are several improvements to the implementation. See Section 3.

2. Language Features

2.1 Preprocessing

All Icon source code passes through a preprocessor before translation. The effects of preprocessing can be seen by running icont or iconc with the –E flag.

Preprocessor directives control the actions of the preprocessor and are not passed to the Icon translator or compiler. If no preprocessor directives are present, the source code passes through the preprocessor unaltered.

A source line is a preprocessor directive if its first non-whitespace character is a $ and if that $ is not followed by another punctuation character. The general form of a preprocessor directive is

$ *directive arguments # comment*

Whitespace separates tokens when needed, and case is significant, as in Icon proper. The entire preprocessor directive must appear on a single line which cannot be continued. The comment portion is optional. An invalid preprocessor directive produces an error except when skipped by conditional compilation.

Preprocessor directives can appear anywhere in an Icon source file without regard to procedure, declaration, or expression boundaries.

Include Directives

An include directive has the form

$include *filename*

An include directive causes the contents of another file to be interpolated in the source file. The file name must be quoted if it is not in the form of an Icon identifier. #line comments are inserted before and after the included file to allow proper identification of errors.

Included files may be nested to arbitrary depth, but a file may not include itself either directly or indirectly. File names are looked for first in the current directory and then in the directories listed in the environment variable LPATH. Relative paths are interpreted in the preprocessor's context and not in relation to the including file's location.

Line Directives

A line directive has the form

$line *n* [*filename*]

The line containing the preprocessing directive is considered to be line *n* of the given file (or the current file, if unspecified) for diagnostic and other purposes. The line number is a simple unsigned integer. The file name must be quoted if it is not in the form of an Icon identifier.

Note that the interpretation of *n* differs from that of the C preprocessor, which interprets it as the number of the *next* line.

$line is an alternative form of the older, special comment form #line. The preprocessor recognizes both forms and produces the fully specified older form for the lexical analyzer.

Define Directives

A define directive has the form

$define *name text*

The define directive defines the text to be substituted for later occurrences of the identifier *name* in the source code. *text* is any sequence of characters except that any string or cset literals must be properly terminated within the definition. Leading and trailing whitespace are not part of the definition. The text can be empty.

Redefinition of a name is allowed only if the new text is exactly the same as the old text. For example, 3.0 is not the same as 3.000.

Redefinition of Icon's reserved words and keywords is allowed but not advised.

Definitions remain in effect through the end of the current original source file, crossing include boundaries, but they do not persist from file to file when names are given on the command line.

If the text of a definition is an expression, it is wise to parenthesize it so that precedence causes no problems when it is substituted. If the text begins with a left parenthesis, it must be separated from the name by at least one space. Note that the Icon preprocessor, unlike the C preprocessor, does not provide parameterized definitions.

Undefine Directives

An undefine directive has the form

$undef *name*

The current definition of *name* is removed, allowing its redefinition if desired. It is not an error to undefine a nonexistent name.

Predefined Symbols

At the start of each source file, several symbols are automatically defined to indicate the Icon system configuration. Each potential predefined symbol corresponds to one of the values produced by the keyword &features. If a feature is present, the symbol is defined with a value of 1. If a feature is absent, the symbol is not defined. See the appendix for a list of predefined symbols.

Predefined symbols have no special status: like other symbols, they can be undefined and redefined.

Substitution

As input is read, each identifier is checked to see if it matches a previous definition. If it does, the value replaces the identifier in the input stream.

No whitespace is added or deleted when a definition is inserted. The replacement text is scanned for defined identifiers, possibly causing further substitution, but recognition of the original identifier name is disabled to prevent infinite recursion.

Occurrences of defined names within comments, literals, or preprocessor directives are not altered.

The preprocessor is ignorant of multi-line literals and can potentially be fooled this way into making a substitution inside a string constant.

The preprocessor works hard to get line numbers right, but column numbers are likely to be rendered incorrect by substitutions.

Substitution cannot produce a preprocessor directive. By then it is too late.

Conditional Compilation

Conditional compilation directives have the form

$ifdef *name*

and

$ifndef *name*

$ifdef or $ifndef cause subsequent code to be accepted or skipped depending on whether *name* has been previously defined. $ifdef succeeds if a definition exists; $ifndef succeeds if a definition does *not* exist. The value of the definition does not matter.

A conditional block has this general form:

```
$ifdef name    or    $ifndef name
   ... code to use if test succeeds ...
$else
   ... code to use if test fails ...
$endif
```

The $else section is optional. Conditional blocks can be nested provided that all of the $if/$else/$endif directives for a particular block are in the same source file. This does not prevent the conditional inclusion of other files via $include as long as any included conditional blocks are similarly self-contained.

Error Directives

An error directive has the form

$error *text*

An $error directive forces a fatal compilation error displaying the given text. This is typically used with conditional compilation to indicate an improper set of definitions.

Subtle Points

Because substitution occurs on replacement text but not on preprocessor directives, either of the following sequences is valid:

```
$define x 1          $define y x
$define y x          $define x 1
write(y)             write(y)
```

It is possible to construct pathological examples of definitions that combine with the input text to form a single Icon token, as in

```
$define X e3          $define Y 456e
write(123X)           write(Y+3)
```

2.2 Graphics Facilities

Version 9.0 provides support for graphics facilities through a combination of high-level support and a repertoire of functions. Not all platforms support graphics. *Note:* There are numerous changes to the graphics facilities in Version 9.0. Persons who used an earlier version of Icon should consult the current reference manual [2].

2.3 New Functions and Keywords

The new functions and keywords are described briefly here. At the end of this report there also is a sheet with more complete descriptions in the style of the second edition of the Icon book. This sheet can be trimmed and used as an insert to the book.

There are six new functions:

chdir(s)	Changes the current directory to s but fails if there is no such directory or if the change cannot be made.
delay(i)	Delays execution i milliseconds. Delaying execution is not supported on all platforms; if it is not, there is no delay and delay() fails.
flush(f)	Flushes the output buffers for file f.
function()	Generates the names of the Icon (built-in) functions.
loadfunc(s1, s2)	Dynamically loads a C function. This function presently is supported on Suns and DEC Alpha systems running UNIX. See [3] for details.
sortf(X, i)	Produces a sorted list of the elements of X. The results are similar to those of sort(X, i), except that among lists and among records, structure values are ordered by comparing their ith fields.

There are six new keywords:

&allocated	Generates the number of bytes allocated since the beginning of program execution. The first result is the total number of bytes in all regions, followed by the number of bytes in the static, string, and block regions.
&dump	If the value of &dump is nonzero at program termination, a dump in the style of display() is provided.
&e	The base of the natural logarithms, 2.71828 ...
&phi	The golden ratio, 1.61803 ...
&pi	The ratio of the circumference of a circle to its diameter, 3.14159 ...
&progname	The file name of the executing program. &progname is a variable and a string value can be assigned to it to replace its initial value.

The graphics facilities add additional new keywords [2].

Some UNIX platforms now support the keyboard functions getch(), getche(), and kbhit(). Whether or not these functions are supported can be determined from the values generated by &features. *Note:* On UNIX platforms, ''keyboard'' input comes from standard input, which may not necessarily be the keyboard. *Warning:* The keyboard functions under UNIX may not work reliably in all situations and may leave the console in a strange mode if inter-

rupted at an unfortunate time. These potential problems should be kept in mind when using these functions.

2.4 Other Language Enhancements

Lists

The functions push() and put() now can be called with multiple arguments to add several values to a list at one time. For example,

```
put(L, x1, x2, x3)
```

appends the values of x1, x2, and x3 to L. In the case of push(), values are prepended in order that they appear from left to right. Consequently, as a result of

```
push(L, x1, x2, x3)
```

the first (leftmost) item on L is the value of x3.

Records

Records can now be sorted by sort() and sortf() to produce sorted lists of the record fields.

A record can now be subscripted by the string name of one of its fields, as in

```
z["r"]
```

which is equivalent to

```
z.r
```

If the named field does not exist for the record, the subscripting expression fails.

Records can now be used to supply arguments in procedure invocation, as in

```
p ! R
```

which invokes p with arguments from the fields of R.

Multiple Subscripts

Multiple subscripts are now allowed in subscripting expressions. For example,

```
X[i, j, k]
```

is equivalent to

```
X[i][j][k]
```

X can be a string, list, table, or record.

Integers

The sign of an integer is now preserved when it is shifted right with ishift().

The form of approximation for large integers that appear in diagnostic messages now indicates a power of ten, as in 10^57. The approximation is now accurate to the nearest power of 10.

Named Functions

The function proc(x, i) has been extended so that proc(x, 0) produces the built-in function named x even if the global identifier having that name has been assigned another value. proc(x, 0) fails if x is not the name of a function.

2.5 Other Changes

- The ability to configure Icon so that Icon procedures can be called from a C program has been eliminated.
- Memory monitoring and the functions associated with it no longer are supported.
- The dynamic declaration, a synonym for local, is no longer supported.
- Real literals that are less than 1 no longer need a leading zero. For example, .5 now is a valid real literal instead of being the dereferencing operator applied to the integer 5.
- A reference to an unknown record field now produces a linker warning message rather than a fatal error. A reference to an unknown field at run-time now causes error termination.
- The identifiers listed by display() are now given in sorted order.
- In sorting structures, records now are first sorted by record name and then by age (serial number).
- Some of the values generated by &features have been changed, and some former values corresponding the features that are present in all implementations of Icon have been deleted. See the appendix.
- The text of some run-time error messages has been changed and a few new error numbers have been added. A complete list is available on request.

3. Implementation Changes

Linker Changes

By default, unreferenced globals (including procedures and record constructors) are now omitted from the code generated by icont. This may substantially reduce the size of icode files, especially when a package of procedures is linked but not all the procedures are used.

The invocable declaration and the command-line options –f s and –v *n* are now honored by icont as well as iconc [4]. The invocable declaration can be used to prevent the removal of specific unreferenced procedures and record constructors that are invoked by string invocation. The –f s option prevents the removal of all unreferenced declarations and is equivalent to invocable all.

The command line option –v *n* to icont controls the verbosity of its output:

–v 0 is the same as icont –s
–v 1 is the default
–v 2 reports the sizes of the icode sections (procedures, strings, and so forth)
–v 3 also lists discarded globals

Note: Programs that use string invocation may malfunction if the default removal of declarations is used. The safest and easiest approach is to add

```
invocable all
```

to such programs.

Other Changes

- The tables used by icont now expand automatically. The –S option is no longer needed. As a side effect of this change, the sizes of procedures are no longer listed during translation.
- Most implementations of Icon now use fixed-sized storage regions. Multiple regions are allocated if needed.
- Under UNIX, shell headers are now produced instead of bootstrapping code in icode files. This substantially reduces the size of icode files on some platforms.
- Under MS-DOS, iconx now finds icode files at any place on the PATH specification as well as in the current directory. The MS-DOS translator now is also capable of producing .exe files.

4. Limitations, Bugs, and Problems

- Line numbers sometimes are wrong in diagnostic messages related to lines with continued quoted literals.
- Large-integer arithmetic is not supported in i to j and seq(). Large integers cannot be assigned to keywords.
- Large-integer literals are constructed at run-time. Consequently, they should not be used in loops where they would be constructed repeatedly.
- Conversion of a large integer to a string is quadratic in the length of the integer. Conversion of very a large integer to a string may take a very long time and give the appearance of an endless loop.
- Right shifting of large negative integers by ishift() is inconsistent with the shifting of ordinary integers.
- Integer overflow on exponentiation may not be detected during execution. Such overflow may occur during type conversion.
- In some cases, trace messages may show the return of subscripted values, such as &null[2], that would be erroneous if they were dereferenced.
- If a long file name for an Icon source-language program is truncated by the operating system, mysterious diagnostic messages may occur during linking.
- Stack overflow checking uses a heuristic that is not always effective.
- If an expression such as

  ```
  x := create expr
  ```

  is used in a loop, and x is not a global variable, unreferenceable co-expressions are generated by each successive create operation. These co-expressions are not garbage collected. This problem can be circumvented by making x a global variable or by assigning a value to x before the create operation, as in

  ```
  x := &null
  x := create expr
  ```

- Stack overflow in a co-expression may not be detected and may cause mysterious program malfunction.

Acknowledgements

The design and implementation of Version 9.0 of Icon was supported in part by National Science Foundation Grant CCR-8901573.

References

1. R. E. Griswold and M. T. Griswold, *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, NJ, second edition, 1990.
2. C. L. Jeffery, G. M. Townsend and R. E. Griswold, *Graphics Facilities for the Icon Programming Language; Version 9.0*, The Univ. of Arizona Icon Project Document IPD255, 1994.
3. R. E. Griswold and G. M. Townsend, *Calling C Functions from Version 9.0 of Icon*, The Univ. of Arizona Icon Project Document IPD240, 1994.
4. R. E. Griswold, *Version 9.0 of the Icon Compiler*, The Univ. of Arizona Icon Project Document IPD237, 1994.

Appendix — Predefined Symbols

predefined symbol	&features value
_AMIGA	Amiga
_ACORN	Acorn Archimedes
_ATARI	Atari ST
_CMS	CMS
_MACINTOSH	Macintosh
_MSDOS_386	MS–DOS/386
_MSDOS	MS–DOS
_MVS	MVS
_OS2	OS/2
_PORT	PORT
_UNIX	UNIX
_VMS	VMS
_COMPILED	compiled
_INTERPRETED	interpreted
_ASCII	ASCII
_EBCDIC	EBCDIC
_EXPANDABLE_REGIONS	expandable regions
_FIXED_REGIONS	fixed regions
_CO_EXPRESSIONS	co–expressions
_DIRECT_EXECUTION	direct execution
_DYNAMIC_LOADING	dynamic loading
_EVENT_MONITOR	event monitoring
_EXECUTABLE_IMAGES	executable images
_EXTERNAL_FUNCTIONS	external functions
_GRAPHICS	graphics
_KEYBOARD_FUNCTIONS	keyboard functions
_LARGE_INTEGERS	large integers
_MEMORY_MONITOR	memory monitoring
_MULTITASKING	multiple programs
_MULTIREGION	multiple regions
_PIPES	pipes
_RECORD_IO	record I/O
_STRING_INVOKE	string invocation
_SYSTEM_FUNCTION	system function
_VISUALIZATION	visualization support
_ARM_FUNCTIONS	Archimedes extensions
_DOS_FUNCTIONS	MS–DOS extensions
_PRESENTATION_MGR	Presentation Manager
_X_WINDOW_SYSTEM	X Windows

In addition, the symbol _V9 is defined in Version 9.0.

---

chdir(s) : n change directory

chdir(s) changes the directory to s but fails if there is no such directory or if the change cannot be made. Whether the change in the directory persists after program termination depends on the operating system on which the program runs.

Error: 103 s not string

---

delay(i) : n delay execution

delay(i) delays execution i milliseconds. This function is not supported on all platforms; if it is not, there is no delay and delay() fails.

Error: 101 i not integer

---

flush(f) : n flush buffer

flush(f) flushes the output buffers for f.

Error: 105 f not file

---

function() : s1, s2, ..., sn generate function names

function() generates the names of the Icon (built-in) functions.

---

loadfunc(s1, s2) : p load external function

loadfunc(s1, s2) loads the function named s2 from the library file s1. s2 must be a C or compatible function that provides a particular interface expected by loadfunc(). loadfunc() is not available on all systems.

---

proc(x, i) : p convert to procedure

proc(x, i) produces a procedure corresponding to the value of x, but fails if x does not correspond to a procedure. If x is the string name of an operator, i specifies the number of arguments: 1 for unary (prefix), 2 for binary (infix) and 3 for ternary. proc(x, 0) produces the built-in function named x even if the global identifier having that name has been assigned another value. proc(x, 0) fails if x is not the name of a function.

Default: i 1

Errors: 101 i not integer
205 i not 0, 1, 2, or 3

---

push(L, x1, x2, ..., xn) : L push onto list

push(L, x1, x2, ..., xn) pushes x1, x2, ..., onto the left end of L. Values are pushed in order from left to right, so xn becomes the first (leftmost) value on L. push(L) with no second argument pushes a null value onto L.

Errors: 108 L not list
307 inadequate space in block region

See also: get(), pop(), pull(), and put()

---

put(L, x1, x2, ..., xn) : L put onto list

put(L, x1, x2, ..., xn) puts x1, x2, ..., onto the right end of L. Values are pushed in order from left to right, so xn becomes the last (rightmost) value on L. put(L) with no second argument puts a null value onto L.

Errors: 108 L not list
307 inadequate space in block region

See also: get(), pop(), pull(), and push()

---

sort(X, i) : L sort structure

sort(X, i) produces a list containing values from x. If X is a list, record, or set, sort(X, i) produces the values of X in sorted order. If X is a table, sort(X, i) produces a list obtained by sorting the elements of X, depending on the value of i. For i = 1 or 2, the list elements are two-element lists of key/value pairs. For i = 3 or 4, the list elements are alternative keys and values. Sorting is by keys for i odd, by value for i even.

Default: i 1

Errors: 101 i not integer
115 X not structure
205 i not 1, 2, 3, or 4
307 inadequate space in block storage region

See also: sortf()

sortf(X, i) : L sort structure by field

sortf(X, i) produces a sorted list of the values in X. Sorting is primarily by type and in most respects is the same as with sort(X, i). However, among lists and among records, two structures are ordered by comparing their ith fields. i can be negative but not zero. Two structures having equal ith fields are ordered as they would be in regular sorting, but structures lacking an ith field appear before structures having them.

Default: i 1

Errors:

	101	i not integer
	126	X not list, record, or set
	205	i = 0
	307	inadequate space in block region

See also: sort()

&allocated : i1, i2, i3, i4 cumulative allocation

&allocated generates the total amount of space, in bytes, allocated since the beginning of program execution. The first value is the total for all regions, followed by the totals for the static, string, and block regions, respectively. The space allocated in the static region is always given as zero. *Note:* &allocated gives the cumulative allocation; &storage gives the current allocation; that is, the amount that has not been freed by garbage collection.

&dump : i termination dump

If the value of &dump is nonzero when program execution terminates, a dump in the style of display() is provided.

&e : r base of natural logarithms

The value of &e is the base of the natural logarithms, 2.71828 ... .

&phi : r golden ratio

The value of &phi is the golden ratio, 1.61803 ... .

&pi : r ratio of circumference to diameter of a circle

The value of &pi is the ratio of the circumference of a circle to its diameter, 3.14159 ... .

&progname : s program name

The value of &progname is the file name of the executing program. A string value can be assigned to &progname to replace its initial value.