The Icon Program Library; Version 5.10*

Compiled by Ralph E. Griswold

TR 85-18

August 9, 1985

Corrected September 3, 1985

Department of Computer Science

The University of Arizona

Tucson, Arizona 85721

*This work was supported by the National Science Foundation under Grant DCR-8401831.

The Icon Program Library; Version 5.10

Preface

The Icon program library consists of Icon programs, procedures, C functions, and support material. The library is distributed as part Version 5.10 of Icon [1-3].

The format of the library mimics the UNIX* Programmer's Manual [4]. Section 1 contains application programs. Section 2 contains collections of Icon procedures. C functions, which can be used to augment the repertoire of built-in Icon functions, are contained in Section 3. Sections 4 and 5 are reserved for future use. Section 6 contains demonstration programs and games. Section 7 contains miscellaneous material, while Icon program library maintenance information is included in Section 8. See the table of contents for details.

1. Using the Library

See *i-hier(7)* for the structure of the Icon system. The program material in the library is divided into three components. Two of the components consists of Icon source code. The third consists of C source code. The prefix *i-* is used to identify programs in the library that are related to Icon program development. Some procedures in the Icon program library depend on particular UNIX environments. Such dependencies are note in the *REQUIREMENTS* sections of the manual pages.

Application Programs and Games

The programs in Sections 1 and 6 are on a par with programs in the corresponding sections of the UNIX manual. The fact that the programs in the Icon library are written in Icon is irrelevant to the user; Icon application programs and games can be moved to a local library without modification.

Procedures

The procedures in Section 2 can be linked into an Icon program in two ways: by specifying them with their .u1 suffix on the *icont(1)* command line when the program is translated [5], or by including an appropriate link directive in the program itself [2]. See the library manual page for *i-hier(7)* for the path that is needed.

Some procedures have global symbols in addition to the procedures names themselves. There is, therefore, the possibility of collisions between names in user programs and names in the library procedures.

C Functions

C functions can be added to the built-in repertoire of Icon by building a personalized interpreter [6]. Alternatively, they can be incorporated into Icon directly [3]. Some of the C functions require the loading of libraries. Check the library manual pages on specific functions.

2. The UNIX Manual Versus the Library Manual

Library manual pages, as included in this report, are slightly different from UNIX manual pages in their head and foot formats. See *lman(8)* for the production of such manual pages. Library manual pages with head and foot formats in the style of the UNIX manual can be obtained as described in *uman(8)*.

Reference to manual pages that are not in the library, such as *ed(1)*, apply to the UNIX manual.

---

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

3. Bug Reports and Submissions of New Material

Bug reports should be mailed to

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

or send electronic mail to

icon-project.arizona@csnet-relay	(CSNET or ARPANET)
arizona!icon-project	(Usenet and uucpnet)

There are currently uucp connections to Arizona through noao, mcnc, ihnp4, and utah-cs.

New material for the library is welcome, although the final decision on inclusion in the Icon library is at the discretion of the Icon Project. Submissions should be sent in machine-readable form, including documentation, either on *tar*-format magnetic tape or via electronic mail.

Acknowledgements

Several persons have contributed material to the Icon program library, as noted by the attributions on the manual pages. In order to make program material easier to read, contributions have been reformatted to provide a degree of uniformity. Program layout therefore does not necessarily reflect the author's preference.

In addition to contributing material to the library, Tom Hicks, Bill Mitchell, and Steve Wampler have made numerous helpful suggestions about the library and its documentation.

References

1. Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 1983.
2. Griswold, Ralph E. and William H. Mitchell. *Version 5.10 of Icon.* Technical Report TR 85-16, Department of Computer Science, The University of Arizona. August 1985.
3. Griswold, Ralph E. and William H. Mitchell. *Installation and Maintenance Guide for Version 5.10 of Icon.* Technical Report TR 85-15, Department of Computer Science, The University of Arizona, Tucson, Arizona. August 1985.
4. *UNIX Programmer's Manual; Seventh Edition, Volume 1.* Bell Laboratories, Murray Hill, New Jersey. 1979.
5. Griswold, Ralph E. and William H. Mitchell. *ICONT(1),* manual page for *UNIX Programmer's Manual,* Department of Computer Science, The University of Arizona. August 1985.
6. Griswold, Ralph E. and William H. Mitchell. *Personalized Interpreters for Version 5.10 of Icon.* Technical Report TR 85-17, Department of Computer Science, The University of Arizona. August 1985.

CONTENTS

PERMUTED INDEX

NAME

cppp - cp preprocessor

SYNOPSIS

cppp [**-d** identifiers] [**-u** identifiers]

DESCRIPTION

Cppp filters standard input to standard output, processing the C preprocessor control lines **#ifdef**, **#ifndef**, and **#else**. If the controlling identifier is in the arguments following the **-d** option, the corresponding lines are included, while if the controlling identifier is in the arguments following the **-u** option, the corresponding lines are deleted. Otherwise, the preprocessor control lines are included. The control lines **#define** and **#undef** are deleted if the identifier is in either list.

For example, suppose that a program contains code for 8-bit, 16-bit, and 32-bit processors under control of the identifiers BIT8, BIT16, and BITB2. A version of the program for 16-bit processors only can be obtained by

```
cppp -d BIT16 -u BIT8 BIT32
```

LIMITATION

Cppp assumes the input file is syntactically correct with respect to control lines.

AUTHOR

Ralph E. Griswold

NAME

csgen - context-sensitive sentences

SYNOPSIS

csgen [-t]

DESCRIPTION

Csgen accepts a context sensitive production grammar from standard input and generates randomly selected sentences from the corresponding language.

Uppercase letters stand for nonterminal symbols, -> indicates the lefthand side can be rewritten by the righthand side. Other characters are considered to be terminal symbols. Lines beginning with # are considered to be comments and are ignored. A line consisting of a nonterminal symbol followed by a colon and a nonnegative integer i is a generation specification for *i* instances of sentences for the language defined by the nonterminal (goal) symbol. An example is:

```
# a(n)b(n)c(n)
# Salomaa, p. 11. Attributed to M. Soittola.
#
X
X->abc
X->aYbc
Yb->bY
Yc->Zbcc
bZ->Zb
aZ->aaY
aZ->aa
X:10
```

A positive integer followed by a colon can be prefixed to a production to replicate that production, making its selection more likely. For example,

```
3:X->abc
```

is equivalent to

```
X->abc
X->abc
X->abc
```

The **-t** option writes a trace of the derivations to standard error output.

LIMITATIONS

Nonterminal symbols can only be represented by single uppercase letters and there is no way to represent uppercase terminal symbols.

There can be only one generation specification and it must appear as the last line of input.

Generation of context-sensitive strings is a slow process. It may not terminate, either because of a loop in the rewriting rules of the grammar or because of the progressive accumulation of nonterminal symbols. *Csgen*, however, avoids deadlock, in which there are no possible rewrites for a string in the derivation.

SEE ALSO

Salomaa, Arto. *Formal Languages*, Academic Press, New York, 1973.

AUTHOR

Ralph E. Griswold

NAME

delam - delaminate a file using column positions

SYNOPSIS

delam fieldspec {input file | -} {output file | -} ...

DESCRIPTION

Delam delaminates an input file into several output files according to the specified fields. *Delam* reads the input file and writes the fields in each line to the corresponding output files as individual lines. If no data occurs in the specified position for a given input line an empty output line is written. This insures that all output files contain the same number of lines as the input file.

If '-' is used for the input file, the standard input is read. If '-' is used as an output file name, the corresponding field is written to the standard output.

The fields are defined by a list of field specifications, separated by commas, colons, or semicolons, of the following form:

n	the character in column *n*
n-m	the characters in columns *n* through *m*
n+*m*	*m* characters beginning at column *n*

where the columns in a line are numbered from 1 to the length of the line.

The use of *delam* is illustrated by the following examples.

```
delam 1-10,5 foo x y
```

reads file foo and writes characters 1 through 10 to file x and character 5 to file y.

```
delam 10+5:1-10:1-10:80 - middle left1 left2 end
```

reads the standard input and writes characters 10 through 14 to middle, 1 through 10 to left1 and left2, and character 80 to end.

```
delam 1-80;1-80 - - -
```

copies the standard input to the standard output, replicating the first eighty columns of each line twice.

NOTES

The functionality of the Software Tools *delam* has been divided. The Icon version of *delam* uses an extended fieldlist syntax.

SEE ALSO

lam(1), delamc(1)

Hanson, David R. *Software Tools User's Manual*, Technical Report TR 81-20, Department of Computer Science, The University of Arizona. 1981.

AUTHOR

Thomas R. Hicks

NAME

delamc - delaminate a file using separator characters

SYNOPSIS

delamc [**-t***chars*] {input file | **-**} {output file | **-**} ...

DESCRIPTION

Delamc delaminates an input file into several output files according to the separator characters specified by the string following the **-t** flag. *Delamc* writes the fields in each line to the corresponding output files as individual lines. If no data occurs in the specified position for a given input line an empty output line is written. This insures that all output files contain the same number of lines as the input file.

If '**-**' is used for the input file, the standard input is read. If '**-**' is used as an output file name, the corresponding field is written to the standard output. If the **-t** flag is not used, an ascii horizontal tab character is assumed as the default field separator.

The use of *delamc* is illustrated by the following examples.

```
delamc tst.asm labels opcodes operands
```

reads the file tst.asm and writes the fields, each of which is separated by a tab character, to the output files labels, opcodes, and operands.

```
delamc -t: scores names matric ps1 ps2 ps3
```

reads the file scores and writes the fields, each of which is separated by a colon, to the indicated output files.

```
delamc -t,:; oldata f1 f2
```

reads the file oldata and separates the fields using either a comma, a colon, or a semicolon.

NOTES

The functionality of the Software Tools *delam* has been divided. *Delamc* differs from the Tools syntax by allowing a set of separation characters to be specified.

SEE ALSO

lam(1), delam(1)

Hanson, David R. *Software Tools User's Manual*, Technical Report TR 81-20, Department of Computer Science, The University of Arizona. 1981.

AUTHOR

Thomas R. Hicks

NAME

edscript - generate script for ed(1)

SYNOPSIS

edscript

DESCRIPTION

Edscript takes specifications for global edits from standard input and outputs an edit script for *ed(1)* to standard output. *Edscript* is primarily useful for making complicated literal substitutions that involve characters that have syntactic meaning to *ed(1)* and hence are difficult to enter in *ed(1)*.

Each specification begins with a delimiter, followed by a target string, followed by the delimiter, followed by the replacement string, followed by the delimiter. For example

```
|...|**|
|****||
```

specifies the replacement of all occurrences of three consecutive periods by two asterisks, followed by the deletion of all occurrences of four consecutive asterisks. Any character may be used for the delimiter, but the same character must be used in all three positions in any specification, and the delimiter character cannot be used in the target or replacement strings.

DIAGNOSTICS

Any line that does not have proper delimiter structure is noted and does not contribute to the edit script.

SEE ALSO

ed(1)

AUTHOR

Ralph E. Griswold

NAME

fset - perform set operations on UNIX file specifications

SYNOPSIS

fset argument

DESCRIPTION

The UNIX shell provides for the specification of filenames using "wildcards". Each wildcard specification may be thought of as defining a set of names (that is, those that match the specification). *Fset* allows the user to apply the set operations of intersection, union, and difference to these filename sets. The resultant list may then be used as an argument to other shell commands.

Fset's argument is an expression composed of legal UNIX file specifications, parenthesis, and the following set operators:

&&	intersection
++	union
--	difference

Because characters that have special meaning to the shell occur frequently in the arguments used for *fset*, it is advisable to quote the arguments consistently.

The use of *fset* is illustrated by the following examples:

```
fset 'g*--*.icn'
```

produces the list (set) of filenames for files beginning with g, excluding those ending with .icn.

Similarly,

```
fset '*'
```

produces all files in the current directory excluding the . and .. files.

```
fset '((*--*.icn)++c*)'
```

and

```
fset '(*--*.icn)++c*'
```

produces the complement of all filenames ending with .icn in addition to all filenames beginning with c.

```
fset '(((c? && c*)))'
```

is a redundant, but legal, specification for all two-character filenames that begin with c, while

```
fset '.*'
```

produces the set of filenames for all hidden files, excluding the . and .. files.

LIMITATIONS

Multiple command line arguments, formed by omitting the quotes around the file set expression, are permitted. Their use is limited, however, since parentheses do not get past the shell's command-line expansion.

Almost any legal file specification will work when enclosed in quotes except that the simple grammar that is used cannot handle blanks adjacent to parentheses.

File names that begin or end in "questionable" characters such as *, ?, +, -, and &, probably will not work.

A file specification that, when interpreted by the shell, produces no matching filename will be placed (unchanged) in the result.

SEE ALSO

gcomp(1)

AUTHOR

Thomas R. Hicks

NAME

gcomp - global complement of file names

SYNOPSIS

gcomp files

DESCRIPTION

Gcomp produces a list of the files in the current directory that do not appear among the arguments. For example,

```
gcomp *.c
```

produces a list of files in the current directory that do not end in .c. As another example, to remove all the files in the current directory that do not match Makefile, *.c, or *.h, the following can be used:

```
rm `gcomp Makefile *.c *.h`
```

The files . and .. are not included in the output, but other 'dot files' are.

BUGS

Gcomp reads the current directory as a file and assumes that all directory entires are 16 bytes in length.

SEE ALSO

fset(1)

AUTHOR

William H. Mitchell

NAME

groupsort - sort groups of lines using the first line as the key

SyNOPSIS

groupsort [-o] [separator string]

DESCRIPTION

Groupsort sorts standard input containing "records" defined to be groups of consecutive lines. Output is written to standard output. Each input record is separated by one or more repetitions of a demarcation line (a line beginning with the separator string). The first line of each record is used as the key.

If no separator string is specified on the command line, the default is the empty string. Because all input lines are trimmed of whitespace (blanks and tabs), blank lines are default demarcation lines. The separator string specified can be an initial substring of the string used to demarcate lines, in which case the resulting partition of the input file may be different from a partition created using the entire demarcation string.

The -o flag sorts the input file but does not produce the sorted records. Instead it lists the keys (in sorted order) and line numbers defining the extent of the record associated with each key.

The use of *groupsort* is illustrated by the following examples. The command

```
groupsort "catscatscatscatscats" <x >y
```

sorts the file x, whose records are separated by lines containing the string "catscatscatscatscats", into the file y placing a single line of "catscatscatscatscats" between each output record. Similarily, the command

```
groupsort "cats" <x >y
```

sorts the file x as before but assumes that any line beginning with the string "cats" delimits a new record. This may or may not divide the lines of the input file into a number of records different from the previous example. In any case, the output records will be separated by a single line of "cats". Another example is

```
groupsort -o <Bibliography >Bibkeys
```

which sorts the file Bibliography and produces a sorted list of the keys and the extents of the associated records. Each output line is of the form:

```
[s-e] keystring
```

where

s is the line number of the key line.
e is the line number of the last line.
keystring is the actual key of the record.

AUTHOR

Thomas R. Hicks

NAME

i-psort - sort procedures in Icon program

SYNOPSIS

i-psort

DESCRIPTION

I-psort reads an Icon program from standard input and writes an equivalent program to standard output with the procedures sorted alphabetically. Global, external, and record declarations come first in the order they appear in the original program. The main procedure comes next followed by the remaining procedures in alphabetical order.

Comments and white space between declarations are attached to the next following declaration.

LIMITATIONS

I-psort only recognizes declarations that start at the beginning of lines.

Comments and interline white space between declarations may not come out as intended.

AUTHOR

Ralph E. Griswold

NAME

i-split - split Icon program into separate files

SYNOPSIS

i-split [-g file]

DESCRIPTION

I-split reads an Icon program from standard input and writes each procedure to a separate file. The output file names consist of the procedure name with **.icn** appended. If the **-g** option is specified, any global, external, and record declarations are written to that file. Otherwise they are written in the file for the procedure that immediately follows them.

Comments and white space between declarations are attached to the next following declaration.

LIMITATIONS

I-split only recognizes declarations that start at the beginning of lines.

Comments and interline white space between declarations may not come out as intended.

If the **-g** option is not specified, any global, external, or record declarations that follow the last procedure are discarded.

AUTHOR

Ralph E. Griswold

NAME

i-trfil - Icon trace filter

SYNOPSIS

i-trfil [options]

DESCRIPTION

I-trfil filters standard input using user-designated strings as keys.

The following options may appear more than once and in any order:

+string	Consider only lines containing *string*; otherwise all lines are considered.
^string	Remove lines containing *string* from consideration (except those containing a string specified by the ! option).
!string	Retain lines containing *string*.
-c*n*	Print *n* context lines before (if *n* is negative) or after (if *n* is positive). Nonsequential context lines are separated by a blank line. The default is 0.

LIMITATIONS

Trace output from Icon programs goes to standard error output, which cannot be piped directly into *i-trfil*. This problem can be circumvented by combining output streams. In *sh(1)*, this can be accomplished by

```
prog 2>&1 | i-trfil ...
```

while in *csh(1)*, it can be accomplished by

```
prog |& i-trfil ...
```

BUG

If the environment variable **TRACE** is set to a nonzero value when *i-trfil* is running, one line of tracing is produced from *i-trfil* in addition to any other tracing.

SEE ALSO

sh(1), csh(1)

AUTHOR

Allan J. Anderson

NAME

i-xref - Icon program cross reference

SYNOPSIS

i-xref [file] [options]

DESCRIPTION

I-xref provides a cross-reference facility for Icon programs. It lists the occurrences of each variable by line number. Variables are listed by procedure or separately as globals. The options specify the formatting of the output and whether or not to cross-reference quoted strings and non-alphanumerics. Variables that are followed by a left parenthesis are listed with an asterisk following the name. If a file is not specified, then standard input is cross-referenced.

The following options change the format defaults. They may appear in any order:

-c *n* The column width per line number. The default is 4 columns wide.

-l *n* The starting column (i.e. left margin) of the line numbers. The default is column 40.

-w *n* The column width of the whole output line. The default is 80 columns wide.

Normally only alphanumerics are cross-referenced. These options expand what is considered:

-q Include quoted strings.

-x Include all non-alphanumerics.

LIMITATION

This program assumes the subject file is a valid Icon program. For example, quotes are expected to be matched.

AUTHOR

Allan J. Anderson

NAME

labels - format labels

SYNOPSIS

labels [options]

DESCRIPTION

Labels writes labels onto the standard output using coded information taken from standard input. In the input, a line beginning with # is a label header. Subsequent lines up to the next header or end-of-file are accumulated and output so as to be centered horizontally and vertically on pin-feed label forms. Lines beginning with * are treated as comments and are ignored.

The following options may appear in any order:

-n	Print *n* copies of each label.
-s *string*	Select only those labels whose headers contain a character in *string*.
-t	Format for curved tape labels (the default is to format for rectangular mailing labels).
-w *n*	Limit line width to *n* characters. The default width is 40.
-l *n*	Limit the number of printed lines per label to *n*. The default is 8.
-d *n*	Limit the depth of the label to *n*. The default is 9 for rectangular labels and 12 for tape labels (**-t**).

Options are processed from left to right. If the number of printed lines is set to a value that exceeds the depth of the label, the depth is set to the number of lines. If the depth is set to a value that is less than the number of printed lines, the number of printed lines is set to the depth. Note that the order in which these options are specified may produce different results.

Label forms should be used with a pin-feed platen. For mailing labels, the carriage should be adjusted so that the first character is printed at the leftmost position on the label and so that the first line of the output is printed on the topmost line of the label. For curved tape labels, some experimentation may be required to get the text positioned properly.

DIAGNOSTICS

If the limits on line width or the number of lines per label are exceeded, a label with an error message is formatted onto standard error output. Thus, if standard output and standard error output are directed to the same file, an erroneous label does not result in misformatting of the other labels.

SEE ALSO

zipsort(1)

AUTHOR

Ralph E. Griswold

NAME

lam - laminate files

SYNOPSIS

lam file [file | - string] ...

DESCRIPTION

Lam laminates the named files onto the standard output. The resulting output is the line-by-line concatenation of corresponding lines from each file named. If the files are different lengths, empty lines are substituted for missing lines in the shorter files.

Each *string* argument is placed in the output line at the point that it appears in the argument list. For example, lines from *file1* and *file2* can be laminated with a colon between each line from *file1* and the corresponding line from *file2* by the command

```
lam file1 -: file2
```

Filenames and strings may appear in any order in the argument list. If '-', is given for a filename the standard input is read at that point. If a file is named more than once, each of its lines will be duplicated on the output line, except that if standard input is named more than once, its lines will be read alternately. For example, each pair of lines from standard input can be joined onto one line with a space between them by the command

```
lam - "- " -
```

while the command

```
lam file1 "- " file1
```

replicates each line from *file1*.

SEE ALSO

delam(1), delamc(1)

Hanson, David R. *Software Tools User's Manual*, Technical Report TR 81-20, Department of Computer Science, The University of Arizona. 1981.

AUTHOR

Thomas R. Hicks

NAME

ll - line length

SYNOPSIS

ll [filename ...]

DESCRIPTION

Ll prints the lengths of the shortest and longest lines in the named files. If no filename argument appears, the standard input is used. '-' may be used to explicitly specify the standard input.

SEE ALSO

Hanson, David R. *Software Tools User's Manual*, Technical Report TR 81-20, Department of Computer Science, The University of Arizona. 1981.

AUTHOR

Thomas R. Hicks

NAME

loadmap - detail the symbols in a compiled file

SYNOPSIS

loadmap [options] file

DESCRIPTION

Loadmap produces a formatted listing of selected symbol classes from a compiled file. The listing is by class, and gives the name, starting address, and length of the region associated with each symbol.

The options are:

-a Display the absolute symbols.

-b Display the BSS segment symbols.

-c Display the common segment symbols.

-d Display the data segment symbols.

-t Display the text segment symbols.

-u Display the undefined symbols.

If no options are specified, **-t** is assumed.

If the address of a symbol cannot be determined, ???? is given in its place.

DEFICIENCIES

The size of the last region in a symbol class is suspect and is usually given as rem.

Output is not particularly exciting on a stripped file.

SEE ALSO

nm(1), size(1)

AUTHOR

Stephen B. Wampler

NAME

parens - produce strings of balanced parentheses

SYNOPSIS

parens [**-b** *n*] [**-n** *n*] [**-l** *s*] [**-r** *s*] [**-v**]

DESCRIPTION

Parens produces parenthesis-balanced strings in which the parentheses are randomly distributed.

The following options may appear in any order:

-b *n* Bound the length of the strings to *n* left and right parentheses each. The default is 10.

-n *n* Produce *n* strings. The default is 10.

-l *s* Use *s* for the left parenthesis. The default is (.

-r *s* Use *s* for the right parenthesis. The default is) .

-v Randomly vary the length of the strings between 0 and the bound. In the absence of this option, all strings are the exactly as long as the specified bound.

SEE ALSO

Arnold, D. B. and M. R. Sleep. "Uniform Random Generation of Balanced Parenthesis Strings", *ACM Transactions on Programming Languages and Systems*, Vol. 2. No. 1 (1980), pp. 122-128.

AUTHOR

Ralph E. Griswold

NAME

roffcmds - usage of *nroff/troff* commands and defined strings

SYNOPSIS

roffcmds

DESCRIPTION

Roffcmds processes standard input and writes a tabulation of *nroff/troff* commands and defined strings to standard output.

LIMITATIONS

Roffcmds only recognizes commands that appear at the beginning of lines and does not attempt to unravel conditional constructions. Similarly, defined strings buried in disguised form in definitions are not recognized.

SEE ALSO

Ossana, Joseph F. *Nroff/Troff User's Manual*, Bell Laboratories, Murray Hill, New Jersey. October 11, 1976.

AUTHOR

Ralph E. Griswold

NAME

rsg - generate random sentences

SYNOPSIS

rsg [-l *n*] [-l *n*] [-t]

DESCRIPTION

Rsg generates randomly selected sentences from a grammar specified by the user.

The following options may appear in any order:

-s *n* Set the seed for random generation to *n*. The default seed is 0.

-l *n* Terminate generation if the number of symbols remaining to be processed exceeds *n*. There is no default limit.

-t Trace the generation of sentences. Trace output goes to standard error output.

Rsg works interactively, allowing the user to build, test, modify, and save grammars. Input to *rsg* consists of various kinds of specifications, which can be intermixed:

Productions define nonterminal symbols in a syntax similar to the rewriting rules of BNF with various alternatives consisting of the concatenation of nonterminal and terminal symbols.

Generation specifications cause the generation of a specified number of sentences from the language defined by a given nonterminal symbol.

Grammar output specifications cause the definition of a specified nonterminal or the entire current grammar to be written to a given file.

Source specifications cause subsequent input to be read from a specified file.

In addition, any line beginning with # is considered to be a comment, while any line beginning with = causes the rest of that line to be used as a prompt to the user whenever *rsg* is ready for input (there normally is no prompt). A line consisting of a single = stops prompting.

Productions

Examples of productions are:

```
<expr>::=<term>|<term>+<expr>
<term>::=<element>|<element>*<term>
<element>::=x|y|z|(<expr>)
```

Productions may occur in any order. The definition for a nonterminal symbol can be changed by specifying a new production for it.

There are a number of special devices to facilitate the definition of grammars, including eight predefined, built-in nonterminal symbols:

symbol	definition
<lb>	<
<rb>	>
<vb>	|
<nl>	newline
<>	empty string
<&lcase>	any single lowercase letter
<&ucase>	any single uppercase letter
<&digit>	any single digit

In addition, if the string between a < and > begins and ends with a single quotation mark, that construction stands for any single character between the quotation marks. For example,

```
<'xyz'>
```

is equivalent to

```
x|y|z
```

Finally, if the name of a nonterminal symbol between the < and > begins with ?, the user is queried during generation to supply a string for that nonterminal symbol. For example, in

```
<expr>::=<term>|<term>+<expr>|<?expr>
```

if the third alternative is encountered during generation, the user is asked to provide a string for <expr>.

Generation Specifications

A generation specification consists of a nonterminal symbol followed by a nonnegative integer. An example is

```
<expr>10
```

which specifies the generation of 10 <expr>s. If the integer is omitted, it is assumed to be 1. Generated sentences are written to standard output.

Grammar Output Specifications

A grammar output specification consists of a nonterminal symbol, followed by ->, followed by a file name. Such a specification causes the current definition of the nonterminal symbol to be written to the given file. If the file is omitted, standard output is assumed. If the nonterminal symbol is omitted, the entire grammar is written out. Thus,

```
->
```

causes the entire grammar to be written to standard output.

Source Specifications

A source specification consists of @ followed by a file name. Subsequent input is read from that file. When an end of file is encountered, input reverts to the previous file. Input files can be nested.

DIAGNOSTICS

Syntactically erroneous input lines are noted, but ignored.

Specifications for a file that cannot be opened are noted and treated as erroneous.

If an undefined nonterminal symbol is encountered during generation, an error message that identifies the undefined symbol is produced, followed by the partial sentence generated to that point. Exceeding the limit of symbols remaining to be generated as specified by the **-l** option is handled in similarly.

CAVEATS

Generation may fail to terminate because of a loop in the rewriting rules or, more seriously, because of the progressive accumulation of nonterminal symbols. The latter problem can be identified by using the **-t** option and controlled by using the **-l** option. The problem often can be circumvented by duplicating alternatives that lead to fewer rather than more nonterminal symbols. For example, changing

```
<expr>::=<term>|<term>+<expr>
```

to

```
<expr>::=<term>|<term>|<term>+<expr>
```

increases the probability of selecting <term> from 1/2 to 2/3. See the second reference listed below for a discussion of the general problem.

SEE ALSO

Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983. pp. 211-219, 301-302.

Wetherell, C. S. "Probabilistic Languages: A Review and Some Open Questions", *Computer Surveys*, Vol. 12, No. 4 (1980), pp. 361-379.

AUTHOR

Ralph E. Griswold

NAME

shuffile - shuffle file

SYNOPSIS

shuffile [-s *n***]**

DESCRIPTION

Shuffile reads a text file from standard input and writes a version with the lines shuffled to standard output.

The option **-s** *n* sets the seed for random generation to *n*. The default seed is 0.

LIMITATIONS

Shuffile is designed to handle text files only.

Shuffile stores the input file in memory and shuffles pointers to the lines; there must be enough memory available to store the entire file.

AUTHOR

Ralph E. Griswold

NAME

tablc - tabulate characters

SYNOPSIS

tablc [-a] [-n] [-u]

DESCRIPTION

Tablc tabulates the characters in standard input and writes a summary to standard output in which each character and the number of times it occurs is listed. Characters are written using Icon's escape conventions.

The following options may appear in any order:

-a Write the summary in alphabetical order of the characters. This is the default.

-n Write the summary in numerical order of the counts.

-u Write only the characters that occur just once.

AUTHOR

Ralph E. Griswold

NAME

tablw - tabulate words

SYNOPSIS

tablw [-a] [-i] [-n] [-s *n*] [-u]

DESCRIPTION

Tablw tabulates the words in standard input and writes a summary to standard output in which each word and the number of times it occurs is listed. A word is defined to be a string of consecutive upper- and lowercase letters with at most one interior occurrence of a dash or apostrophe.

The following options may appear in any order:

-a Write the summary in alphabetical order of the words. This is the default.

-i Ignore case distinctions among letters (uppercase letters are mapped into corresponding lowercase letters on input).

-n Write the summary in numerical order of the counts.

-s *n* Tabulate only words longer than *n* characters. The default value for *n* is 0.

-u Write only the words that occur just once.

AUTHOR

Ralph E. Griswold

NAME

trim - trim lines

SYNOPSIS

trim [*n*] [**-f**]

DESCRIPTION

Trim copies lines from standard input to standard output, truncating the lines at *n* characters and removing any trailing blanks. The default value for *n* is 80.

The **-f** option causes all lines to be *n* characters long; otherwise shorter lines are left as is.

AUTHOR

Ralph E. Griswold

NAME

zipsort - sort labels by zip code

SYNOPSIS

zipsort [-d *n***]**

DESCRIPTION

Zipsort sorts labels produced by *labels*(1) in the order of their postal zip codes.

The option **-d** *n* sets the number of lines per label to *n*. The default is 9. This value must agree with the value used to format the labels; see *labels*(1).

The zip code must be the last nonblank string at the end of the label. It may consist of digits with an embedded dash for extended zip codes.

DIAGNOSTICS

If a label does not end with a legal zip code, the label is printed to standard error output, but it also is included in standard output after all labels with legal zip codes.

SEE ALSO

labels(1)

AUTHOR

Ralph E. Griswold

NAME

bitops – operations on bit strings

DESCRIPTION

These procedures perform operations on characters strings of zeros and ones ('bit strings')

SYNOPSIS

and(b1,b2)	logical 'and' of b1 and b2
bitstring(i)	convert integer i to bit string
bsum(b1,b2)	arithmetic sum of b1 and b2 (used by other procedures)
decimal(b)	convert b to integer
exor(b1,b2)	'exclusive-or' of b1 and b2
neg(b)	negation of b
or(b1,b2)	logical 'or' of b1 and b2

NOTE

If i in bitstring(i) is negative, the value produced is the corresponding unsigned 32-bit bit string.

BUGS

Integer values that exceed those allowable in Icon may produce bogus results or spurious diagnostics.

AUTHOR

Ralph E. Griswold

NAME

bold - boldface and underscored text

DESCRIPTION

These procedures produce text with interspersed characters suitable for printing to produce the effect of boldface (by overstriking) and underscoring (using backspaces).

SYNOPSIS

bold(s)	bold version of s
uscore(s)	underscored version of s

AUTHOR

Ralph E. Griswold

NAME

collate - collate and decollate strings

DESCRIPTION

These procedures collate (interleave) respective characters of two strings and decollate by selecting every other character of a string.

SYNOPSIS

collate(s1,s2)	produce a string consisting of interleaved characters of s1 and s2. For example, collate("abc","def") produces "adbecf".
decollate(s,i)	produce a string consisting of every other character of s. If i is odd, the odd-numbered characters are selected, while if i is even, the even-numbered characters are selected.

DIAGNOSTICS

Run-time error 208 occurs if the arguments to collate are not of the same size.

AUTHOR

Ralph E. Griswold

NAME

complex - arithmetic on complex numbers

DESCRIPTION

These procedures perform operations on complex numbers.

SYNOPSIS

complex(r,i)	create complex number with real part r and imaginary part i
cpxadd(x1,x2)	add complex numbers x1 and x2
cpxdiv(x1,x2)	divide complex number x1 by complex number x2
cpxmul(x1,x2)	multiply complex number x1 by complex number x2
cpxsub(x1,x2)	subtract complex number x2 from complex number x1
cpxstr(x)	convert complex number x to string representation
strcpx(s)	convert string representation s of complex number to complex number

SEE ALSO

Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983. pp. 60, 285-286.

AUTHOR

Ralph E. Griswold

NAME

escape - interpret Icon literal string

DESCRIPTION

The procedure **escape(s)** produces a string in which Icon quoted literal escape conventions in **s** are replaced by the corresponding characters.

LIMITATIONS

Octal and hexadecimal literal specifications cannot be abbreviated by omitting leading zeros.

SEE ALSO

Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983. p. 232.

AUTHOR

William H. Mitchell

NAME

gener – generators

DESCRIPTION

These procedures generate sequences of results.

SYNOPSIS

hex()	sequence of hexadecimal codes for numbers from 0 to 255
label(s,i)	sequence of labels with prefix **s** starting at **i**
octal()	sequence of octal codes for numbers from 0 to 255
star(s)	sequence consisting of the closure of **s** starting with the empty string and continuing in lexical order as given in **s**

SEE ALSO

Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983. pp. 145.

AUTHOR

Ralph E. Griswold

NAME

gpack - graphics package for the Chromatics CG 3999

SYNOPSIS

bckgrnd(colr)	set background color
_char()	switch to character mode
clip(mode)	turn clip mode ON/OFF
clipped(object)	clip an object to window size (internal)
color(colr)	set foreground color
curcol(colr)	set cursor color
draw(object)	display object
enable(colr)	enable particular color guns
erase()	clear the screen
_fill()	turn on fill mode
ginit()	initialize graphics package
mode(newmode)	set plot submode (internal)
movcur(x,y)	position cursor (internal)
_nofill()	turn off fill mode
_noroll()	turn off roll mode
_number(n)	output number (n) (internal)
_plot()	switch to plot mode
_point(x,y)	output point (x,y) (internal)
_restore()	reset terminal windows
_roll()	turn on roll mode
setscale(xmin,ymin,xmax,ymax,colmin,rowmin,colmax,rowmax)	scaling
scale(pt)	scale a point
_scale(mode)	turn ON/OFF scaling
text(x,y,s)	output string s at point (x,y)
window(w)	switch to window w (1 to 4)
wsize(xmin,ymin,xmax,ymax)	set window size
xfit(x)	scale x-coordinate (internal)
_xydel(xdelta,ydelta)	output incremental motion (internal)
yfit(y)	scale y-coordinate (internal)

Gpack is a package designed to interface to the Chromatics CG 3999 Color Graphics Terminal. The user must be familiar with the Chromatics terminal and its capabilities. *Gpack* maintains information on the state of each hardware window and avoids the transmission of redundant commands. The following objects, defined as records, are known to *gpack*:

point(x,y)
dot(x,y)
line(a,b)
box(a,b)
circle(center,radius)
arc(center,radius,start,stop)

points(pts)	a collection of points.
lines(pts)	a collection of points, joined by vectors
polygon(pts)	a collection of points, joined and closed by vectors
incdots(start,motions)	start point and list of motions
motion(xdel,ydel)	an incremental motion

The procedure **draw** can display any of these objects, with or without scaling. Additionally, **draw** accepts a co-expression that generates these objects.

Two record types are used internally:

```
wind(pmode,smode,cmode,fmode,rmode,psubmode,fc,bc,lowerleft,upperright)
scaling(xslope,xinter,yslope,yinter)
```

where

pmode is ON if window is in plot mode.
smode is ON if scaling is active.
cmode is ON if clipping is active.
fmode is ON if fill is active.
rmode is ON if roll is active.
psubmode is the plot submode.
fc is the foreground color.
bc is the background color.
lowerleft, upperright are the window bounds.
xslope, xinter are the x-coordinate scaling constants
yslope, yinter are the y-coordinate scaling constants

The defaults for all windows are:

```
wind(OFF,OFF,OFF,OFF,ON,"",NOCOLOR,NOCOLOR,point(0,0),point(XMAX,YMAX))
scaling(1,0,1,0)
```

The following values are predefined globals, and should not be reassigned:

MODE, ESC	used internally
ON, OFF	mode settings
XMAX, YMAX	maximum screen addresses
DOT,VECTOR,RECTANGLE,CIRCLE,ARC,CONCVECT,INCDOT	plot submodes
BLACK,BLUE,GREEN,CYAN,RED,MAGENTA,YELLOW,WHITE,BLINK	colors

The following globals are used internally:

_wno	current window (1 to 4), initially 1
window	list of window attributes
wscale	list of window scaling factors

The procedure *ginit()* must be called to initialize global constants and set window attributes at the start of any program using *gpack*. Procedures that are marked 'internal' are intended primarily for internal use by *gpack*. Some procedures are generators that reverse their effects.

SEE ALSO

worm(6)

DEFICIENCIES

Object clipping has not been implemented. **radius** is not scaled for **circle** and **arc**.

AUTHOR

Stephen B. Wampler

NAME

Image - generalized string image of Icon value

SYNOPSIS

```
Image(x)
Imagex(x)
```

DESCRIPTION

The procedure Image(x) produces a string image of the value x. The value produced is a generalization of the value produced by the Icon function image(x), providing detailed information about structures.

Tags are used to uniquely identify structures. A tag consists of a letter identifying the type followed by an integer. The tag letters are L for lists, R for records, and T for tables. The first time a structure is encountered, it is imaged as the tag followed by a colon, followed by a representation of the structure. If the same structure is encountered again, only the tag is given.

An example is

```
a := ["x"]
push(a,a)
t := table()
push(a,t)
t[a] := t
t["x"] := []
t[t] := a
write(Image(t))
```

which produces

```
T1:["x"->L1:[],L2:[T1,L2,"x"]->T1,T1->L2]
```

Note that a table is represented as a list of entry and assigned values separated by ->s.

The procedure Imagex(x) is similar to Image(x), except that newlines and spaces are inserted to that the printed result is displayed on multiple lines with indentation. There are other formatting details that differ between the two procedures. For the example given above, the result of

```
write(Imagex(t))
```

is

```
T1:{
  "x"
  ---
L1:[]
  ]
------
L2:[
  T1
  L2
```

```
  "x"
  ]
---
T1
------
T1
---
L2
------
}
```

AUTHOR

Ralph E. Griswold

NAME

lmap - list map

SYNOPSIS

lmap(a1,a2,a3) map elements of a1 according to a2 and a3.

DESCRIPTION

This procedure is the analog for lists of the built-in string-mapping function map(s1,s2,s3). Elements in a1 that are the same as elements in a2 are mapped into the corresponding elements of a3. For example, given the lists

```
a1 := [1,2,3,4]
a2 := [4,3,2,1]
a3 := ["a","b","c","d"]
```

then

```
lmap(a1,a2,a3)
```

changes a1 to

```
["d","c","b","a"]
```

Note that the value of a1 is modified. Lists that are mapped can have any kinds of elements. The operation

```
x === y
```

is used to determine if elements x and y are equivalent.

All cases in lmap are handled as they are in map, except that no defaults are provided for omitted arguments. As with map, lmap can be used for transposition as well as substitution.

SEE ALSO

Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983. pp. 37-38, 181-186.

AUTHOR

Ralph E. Griswold

NAME

patterns - SNOBOL4-style pattern matching

DESCRIPTION

These procedures are adapted from TR 80-25 and TR 81-6. They provide procedural equivalents for most SNOBOL4 patterns and some extensions.

SYNOPSIS

Anchor()	&ANCHOR = 1 if Mode := Anchor
Any(s)	ANY(S)
Apply(s,p)	S ? P
Arb()	ARB
Arbno(p)	ARBNO(P)
Arbx(i)	ARB(I)
Bal()	BAL
Break(s)	BREAK(S)
Breakx(s)	BREAKX(S)
Cat(p1,p2)	P1 P2
Discard(p)	/P
Exog(s)	\S
Find(s)	FIND(S)
Float()	&ANCHOR = 0 if Mode := Float
Len(i)	LEN(I)
Limit(p,i)	P \ i
Locate(p)	LOCATE(P)
Marb()	longest-first ARB
Mode()	Anchored or unanchored matching (see Anchor and Float)
Notany(s)	NOTANY(S)
Pos(i)	POS(I)
Replace(p,s)	P ≡ S
Rpos(i)	RPOS(I)
Rtab(i)	RTAB(I)
Span(s)	SPAN(S)
String(s)	S
Succeed()	SUCCEED
Tab(i)	TAB(I)
Xform(f,p)	F(P)

In addition to the procedures above, the following expressions can be used:

p1() | p2()	P1 | P2
v <- p()	P . V (approximate)
v := p()	P $ V (approximate)
fail	FAIL
=s	S (in place of String(s))
p1() || p2()	P1 P2 (in place of Cat(p1,p2))

Using this system, most SNOBOL4 patterns can be satisfactorily transliterated into Icon procedures and expressions. For example, the pattern

```
SPAN("0123456789") $ N "H" LEN(*N) $ LITERAL
```

can be transliterated into

```
(n <- Span('0123456789')) || ="H" || (literal <- Len(n))
```

Concatenation of components is necessary to preserve the pattern-matching properties of SNOBOL4. See the documents listed below for details and limitations.

CAVEATS

Simulating SNOBOL4 pattern matching using the procedures above is inefficient.

SEE ALSO

Ralph E. Griswold. *Pattern Matching in Icon*, TR 80-25, The University of Arizona, 1980.

Ralph E. Griswold. *Models of String Pattern Matching*, TR 81-6, Department of Computer Science, The University of Arizona, 1981.

Ralph E. Griswold. "Implementing SNOBOL4 Pattern Matching in Icon", *Computer Languages*, Vol. 8, No. 8 (1983), pp. 77-92.

AUTHOR

Ralph E. Griswold

NAME

pdae - programmer-defined argument evaluation

DESCRIPTION

These procedures are taken mainly from TR 82-16, which describes how co-expressions can be used to model the built-in argument evaluation regime of Icon and also provide new ones. Some procedures have been corrected or improved.

SYNOPSIS

Allpar{e1,e2, ...}	parallel evaluation with last result used for short sequences
Call(a)	utility procedure to call functions
Extract{e1,e2, ...}	extract results of even-numbered arguments according to odd-numbered values
Lifo{e1,e2, ...}	models standard Icon 'lifo' evaluation
Parallel{e1,e2, ...}	parallel evaluation terminating on shortest sequence
Reverse{e1,e2, ...}	left-to-right reversal of lifo evaluation
Rotate{e1,e2, ...}	parallel evaluation with shorter sequences re-evaluated
Simple{e1,e2, ...}	simple evaluation with only success or failure

BUGS AND LIMITATIONS

Because of the handling of the scope of local identifiers in co-expressions, expressions in programmer-defined argument evaluation regimes cannot communicate through local identifiers. Some constructions, such as **break** and **return**, cannot be used in arguments to programmer-defined argument evaluation regimes. See TR 82-8 for details of these problems.

At most 10 arguments can be used in the invocation of a programmer-defined argument evaluation regime. This limit can be increased by modifying **Call**.

SEE ALSO

pdco(2)

Griswold, Ralph E. and Michael Novak. "Programmer-Defined Control Operations", *The Computer Journal*, Vol. 26, No. 2 (1983), pp. 175-183.

Novak, Michael and Ralph E. Griswold. *Programmer-Defined Argument Evaluation Regimes*, TR 82-16, Department of Computer Science, The University of Arizona, 1982.

AUTHORS

Ralph E. Griswold and Michael Novak

NAME

pdco - programmer-defined control operations

DESCRIPTION

These procedures are taken mainly from TR 82-8, which describes how co-expressions can be used to model the built-in control structures of Icon and also provide new ones. Some procedures have been corrected or improved and there are additions.

SYNOPSIS

Alt{e1,e2}	models e1 | e2
Colseq{e1,e2, ...}	produces results of e1, e2, ... alternately
Comseq{e1,e2}	compares result sequences of e1 and e2
Cond{e1,e2, ...}	generalized Lisp conditional
Every{e1,e2}	models every e1 do e2
Galt{e1,e2, ...}	generalized alternation: e1 | e2 | ...
Lcond{e1,e2, ...}	Lisp conditional
Limit{e1,e2}	models e1 \ e2
Ranseq{e1,e2, ...}	produces results of e1, e2, ... at random
Repalt{e}	models |e
Resume{e1,e2,e3}	models every e1 \ e2 do e3
Select{e1,e2}	produces results from e1 by position according to e2

BUGS AND DEFICIENCIES

Because of the handling of the scope of local identifiers in co-expressions, expressions in programmer-defined control operations cannot communicate through local identifiers. Some constructions, such as **break** and **return**, cannot be used in arguments to programmer-defined control operations. See TR 82-8 for details of these problems.

SEE ALSO

pdae(2)

Griswold, Ralph E. and Michael Novak. "Programmer-Defined Control Operations", *The Computer Journal*, Vol. 26, No. 2 (1983), pp. 175-183.

AUTHORS

Ralph E. Griswold and Michael Novak

NAME

radcon – radix conversion procedures

DESCRIPTION

These procedures convert numbers from one radix to another. The letters from **a** to **z** are used for 'digits' greater than 9. All the conversion procedures fail if the conversion cannot be made.

SYNOPSIS

exbase10(i,j)	convert base-10 integer i to base j
inbase10(s,i)	convert base-i integer s to base 10.
radcon(s,i,j)	convert base-i integer s to base j.

LIMITATION

The maximum base allowed is 36.

SEE ALSO

Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983. pp. 67, 286.

AUTHOR

Ralph E. Griswold

NAME

segment - segment string

DESCRIPTION

The procedure **segment(s,c)** generates consecutive substrings of **s** consisting of characters that respectively do/do not occur in **c**. For example,

```
segment("This is a sentence.",&lcase ++ &ucase)
```

generates

```
"This"
" "
"is"
" "
"a"
" "
"sentence"
"."
```

AUTHOR

William H. Mitchell

NAME

Seqimage - produce string image of result sequence

DESCRIPTION

The procedure Seqimage{e,i,j} produces a string image of the result sequence for the expression e. The first i results are printed. If i is omitted, there is no limit. If there are more than i results for e, ellipses are provided in the image after the first i. If j is specified, at most j results from the end of the sequence are printed after the ellipses. If j is omitted, only the first i results are produced.

For example, the expressions

```
Seqimage{1 to 20}
Seqimage{1 to 20,10}
Seqimage{1 to 20,10,5}
```

produce, respectively,

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ...}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ..., 16, 17, 18, 19, 20}
```

WARNING

If j is not omitted and e has a infinite result sequence, Seqimage does not terminate.

SEE ALSO

Griswold, Ralph E. and Michael Novak. "Programmer-Defined Control Operations", *The Computer Journal*, Vol. 26, No. 2 (1983), pp. 175-183.

AUTHOR

Ralph E. Griswold

NAME

shuffle - shuffle string or list

DESCRIPTION

The procedure **shuffle(x)** shuffles a string or list. In the case that **x** is a string, a corresponding string with the characters randomly rearranged is produced. In the case that **x** is a list, the values in the list are randomly rearranged.

DIAGNOSTIC

Run-time time error 102 occurs if **x** is not a list, a string, or a value that can be converted to a string.

SEE ALSO

Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983. p. 187.

AUTHORS

Ward Cunningham and Ralph E. Griswold

NAME

Size - size of Icon object

DESCRIPTION

The procedure Size(x) produces the size, in bytes, of x.

There is an optional second argument, which specifies the machine word size, in bits. For example, Size(x,16) produces the size of x on a 16-bit machine. Only word sizes of 16 and 32 are supported; the default, in the absence of the second argument, is 32.

CAVEATS

The sizes given are approximate for many objects. The following interpretations are made in determining sizes:

Strings are counted as distinct; no allowance is made for shared usage of storage.

The size of procedures may be underestimated.

The size of co-expressions is approximate and does not take into account space for the local identifiers and related information.

The size of lists that are augmented by stack and queue functions may be underestimated.

Sizes are based on parameters as given in the Icon system as it is distributed; local changes may affect the sizes of some objects.

SEE ALSO

Griswold, Ralph E. and William H. Mitchell. *A Tour Through the C Implementation of Icon; Version 5.10*, Department of Computer Science, The University of Arizona, 1985.

AUTHOR

Ralph E. Griswold

NAME

snapshot - snapshot of state of string scanning

DESCRIPTION

The procedure **snapshot()** writes a snapshot of the state of string scanning, showing the value of **&subject** and **&pos**. For example,

```
"((a+b)-delta)/(c*d))" ? (tab(bal('+-/*')) & snapshot())
```

produces

```
-------------------------------------
|                                   |
| &subject = "((a+b)-delta)/(c*d))" |
|                          |        |
-------------------------------------
```

Note that the bar showing the **&pos** is positioned under the **&pos**th character (actual positions are between characters). If **&pos** is at the end of **&subject**, the bar is positioned under the quotation mark delimiting the subject. For example,

```
"abcdefgh" ? (tab(0) & snapshot())
```

produces

```
-------------------------
|                       |
| &subject = "abcdefgh" |
|                     | |
-------------------------
```

Escape sequences are handled properly. For example,

```
"abc\tdef\nghi" ? (tab(upto('\n')) & snapshot())
```

produces

```
-----------------------------
|                           |
| &subject = "abc\tdef\nghi" |
|                    |      |
-----------------------------
```

AUTHOR

Ralph E. Griswold and Randal L. Schwartz

NAME

structs – structure operations

DESCRIPTION

These procedures for manipulating structures are taken from *The Icon Programming Language.*

SYNOPSIS

depth(a)	compute maximum depth of tree a
eq(x,y)	compare list structures x and y
ldag(s)	construct a dag from the string s
lgraph(s)	construct a graph from the string sgraph
ltree(s)	construct a tree from the string s
stree(a)	construct a string from the list a
tcopy(a)	copy tree a
teq(a1,a2)	compare trees a1 and a2
visit(a)	visit, in preorder, the nodes of a

NOTE

The procedure ldag has a second argument that is used on internal recursive calls; a second argument must not be supplied by the user.

SEE ALSO

Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983. pp. 170-180, 295.

AUTHOR

Ralph E. Griswold

NAME

strutil - string utilities

DESCRIPTION

These procedures perform simple operations on strings.

SYNOPSIS

compress(s,c)	compress consecutive occurrences of c in s.
delete(s,c)	delete occurrences of c in s
rotate(s,i)	rotate s i characters to the left (negative i produces rotation to the right); the default value of i is 1.

SEE ALSO

Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983. pp. 46, 283-284.

AUTHOR

Ralph E. Griswold

NAME

ttyinit - initialize predefined terminal control attributes

DESCRIPTION

The procedure ttyinit() initializes a number of global identifiers for terminal attributes:

TANDEM
CBREAK
LCASE
ECHO
CRMOD
RAW
ODDP
EVENP

These identifiers serve as useful attributes for **stty** in *ttyctl(3.icon).*

SEE ALSO

ttyctl(3.icon)

AUTHOR

Stephen B. Wampler

NAME

getenv - get value for environment variable

SYNOPSIS

getenv(s)

DESCRIPTION

getenv searches the environment list for a string of the form s=value and returns value if such a string is present; otherwise, it fails.

SEE ALSO

getenv(3)

AUTHOR

Stephen B. Wampler

NAME

Descr, Indir, Word1, Word2, Symbol, Efp, Gfp, Pfp - examine Icon internals

DESCRIPTION

These functions deal with Icon internals, producing values that exist in memory and registers as Icon is executing. Internal values are represented as Icon integers. A knowledge of the implementation of Icon is needed to use these functions properly.

SYNOPSIS

Descr(i,j)	compose a descriptor whose first word is i and whose second word is j.
Indir(i)	return an integer corresponding to the value where the descriptor at i points.
Word1(x)	return Icon integer whose value is the first word of the descriptor x.
Word2(x)	like Word1, except that the value is for the second word of the descriptor x.
Symbol(s)	returns the address of the symbol s. The allowable values of s are:

globals	address of the global vector
eglobals	address of the end of the global vector
gnames	address of the global name vector
strbase	beginning of the allocated string region
strfree	string region free pointer
blkbase	beginning of the allocated block region
blkfree	block region free pointer
stkbase	beginning of the co-expression stack region
stkfree	co-expression stack region free pointer

Symbol(s) fails if s is not one of these strings.

Efp()	return the address of the expression frame pointer
Gfp()	return the address of the generator frame pointer
Pfp()	return the address of the frame pointer

CAVEATS

Iscope is inherently dangerous. For example, the composition of an arbitrary descriptor using Descr may cause the Icon system to malfunction in mysterious ways.

Addresses that are represented by integers are not relocated during garbage collection; such addresses generally are invalidated by a garbage collection.

REQUIREMENTS

Efp, Gfp, and Pfp are only implemented for the PDP-11, Sun Workstation, and VAX-11 implementations of Icon.

SEE ALSO

Griswold, Ralph E. and William H. Mitchell. *A Tour Through the C Implementation of Icon; Version 5.10*, Department of Computer Science, The University of Arizona. 1985.

Griswold, Ralph E. "A Portable Diagnostic Facility for SNOBOL4", *Software—Practice and Experience*, Vol. 5 (1975), pp. 93-105.

Griswold, Ralph E. "Linguistic Extension of Abstract Machine Modelling to Aid Software Development", *Software—Practice and Experience*, Vol. 10 (1980), pp. 1-9.

AUTHORS

Ralph E. Griswold and William H. Mitchell

NAME

exp, log, log10, sqrt - miscellaneous math functions

SYNOPSIS

```
exp(x)
log(x)
log10(x)
sqrt(x)
```

DESCRIPTION

exp returns e^x.

log returns the natural logarithm of x.

log10 returns the base-10 logarithm of x.

sqrt returns the square root of x.

The C math library (-lm)must be loaded with these functions.

DIAGNOSTICS

Run-time error 251 occurs if x is zero or negative in log or log10.

Run-time error 252 occurs for overflow in exp.

SEE ALSO

exp(3m)

AUTHORS

Ralph E. Griswold

NAME

seek - seek to position in stream

SYNOPSIS

seek(file,offset,ptrname)

DESCRIPTION

seek sets the position of the next input or output operation on **file**. The new position is at the signed distance **offset** bytes from the beginning, the current, or the end of the file, depending on whether **ptrname** is 0, 1, or 2. **seek** returns the current value of the offset relative to the beginning of **file**.

SEE ALSO

fseek(3s)

AUTHOR

Stephen B. Wampler

NAME

sin, cos, tan, asin, acos, atan, atan2, dtor, rtod - trigonometric functions

SYNOPSIS

```
sin(x)
cos(x)
tan(x)
asin(x)
acos(x)
atan(x)
atan2(x,y)
dtor(x)
rtod(x)
```

DESCRIPTION

sin, cos, and tan return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure that the result is meaningful.

asin returns the arc sin in the range $-\pi/2$ to $\pi/2$.

acos returns the arc cosine in the range 0 to π.

atan returns the arc tangent in the range $-\pi/2$ to $\pi/2$.

atan2 returns the tangent of x/y in the range $-\pi$ to π.

dtor converts degrees to radians, while rtod converts radians to degrees.

The C math library (-lm) must be loaded with these functions.

DIAGNOSTICS

Run-time error 251 occurs if the magnitude of the argument to asin or acos is greater than 1.

Run-time error 252 occurs for the singular points of tan.

BUGS

The value of tan for arguments greater than about 2^{31} is garbage.

SEE ALSO

sin(3m)

AUTHORS

Ralph E. Griswold and Stephen B. Wampler

NAME

ttyctl - primitive control of terminal attributes

SYNOPSIS

```
stty(x1,x2, ..., xn)
restty()
keyin()
```

DESCRIPTION

stty sets or clears the terminal attributes given as arguments. Positive attributes are set and negative attributes are cleared. **restty** attempts to restore terminal attributes to their original condition. A reasonable guess is made if the original attributes cannot be determined. **keyin** succeeds if there is any input waiting to be read from standard input but fails otherwise.

Attributes for **stty** may be any legitimate bit pattern. However, if the procedure **ttyinit** in *ttyinit(2)* has been executed, the following global identifiers are available as arguments:

```
TANDEM
CBREAK
LCASE
ECHO
CRMOD
RAW
ODDP
EVENP
```

REQUIREMENTS

The function **keyin** assumes Berkeley extensions.

SEE ALSO

ttyinit(2)

AUTHOR

Stephen B. Wampler

NAME

cross - intersection of words

SYNOPSIS

cross

DESCRIPTION

Cross displays an intersection of words given one per line on standard input. Uppercase letters are mapped into lowercase on input.

BUGS

Cross only produces one possible intersection and does not attempt to obtain the most compact result.

DIAGNOSTICS

Cross objects if the input contains a nonalphabetic character.

AUTHOR

William P. Malloy

NAME

deal - deal bridge hands

SYNOPSIS

deal [**-h** *n*] [**-s** *n*]

DESCRIPTION

Deal shuffles, deals, and displays hands in the game of bridge.

The following options may appear in either order:

-h *n* Display *n* hands. The default is 1.

-s *n* Set the seed for random generation to *n*. The default seed is 0.

NOTE

The letter T is used to represent the 10.

AUTHOR

Ralph E. Griswold

NAME

farb - produce random farberism

SYNOPSIS

farb

NOTE

A few of the farberisms may be objectionable to some persons.

AUTHOR

Ralph E. Griswold (with compliments to David J. Farber)

NAME

queens - solutions to n-queens problem

SYNOPSIS

queens [*n*]

DESCRIPTION

Queens displays the solutions to the nonattacking queens problem. The number of queens is given by *n*; the default is 6.

AUTHOR

Stephen B. Wampler

NAME

worm - display random worm on the Chromatics CG 3999

SYNOPSIS

worm [-b*n* -s*n* -f*n* -l*n* -r*n*]

DESCRIPTION

Worm displays the movement of a random worm on the Chromatics CG 3999 color graphics terminal. The following options may be given in any order:

-b*n*	set background color (0-7, default: 1 {blue})
-c*n*	set duration of display (default: 1000)
-f*n*	set foreground color (0-7, default: 7 {white})
-l*n*	set length of worm (default: 10)
-r*n*	set random number seed (default: 0)

SEE ALSO

gpack(2)

AUTHOR

Stephen B. Wampler

NAME

i-hier - Icon hierarchy

DESCRIPTION

The following outline gives a sketch of the Icon hierarchy.

v5	root of the Icon system (location may vary from site to site)		
	/library	Icon program library	
		/src	source code for Icon library programs
			/cmd source code for programs
			/lib source code for procedure libraries
		/ibin	executable binaries for programs
		/ilib	linkable code for procedure libraries
		/libtest	Icon library test programs
		/man	manual
			/man0 front matter
			/man1 application programs
			/man2 procedures
			/man3 C functions
			... ...
			/man7 miscellaneous
			/man8 library maintenance
			/cat0 formatted front matter for manual
			/cat1 formatted pages for application programs
			... ...
	/rtlib	code for building personalized interpreters	
	/docs	Icon documentation	
	/book	source code for procedures from the Icon book	
	/bin	executable binaries for Icon	
	/src	source code for the Icon system	
		/tran	source code for the Icon translator
		/link	source code for the Icon linker
		/h	header files for the Icon system
		/fncs	source code for built-in functions
		/ops	source code for operators
		/rt	source code for run-time support routines
		/lib	source code for routines called by the Icon interpreter
		/iconx	source code for the Icon interpreter
		/icont	source code for the Icon command processor
		/sys	source code for target machine
		/proto	source code for prototype implementation
		/att3b	source code for AT&T 3B implementation
		/pdp11	source code for PDP-11 implementation
		/ridge	source code for Ridge 32 implementation
		/mc68000	source code for Sun Workstation implementation
		/vax	source code for VAX implementation
		/pifncs	source code for Icon library C functions
	/pidem	sample personalized interpreter	
	/samples	Icon installation test programs	
	/test	Icon test suite	
	/port	Icon porting test suite	

SEE ALSO

Griswold, Ralph E. and William H. Mitchell. *Installation and Maintenance Guide for Version 5.10 of Icon*, TR 85-15, Department of Computer Science, The University of Arizona, 1985.

NAME

lman - format manual page in library style

DESCRIPTION

The manual pages for the Icon Library can be produced with the head and foot format of the Icon Library manual by using the macros in v5/library/man/tmac.an, as in

```
cd v5/library/man
troff tmac.an man1/rsg.1
```

NOTE

The text for the manual pages includes files in v5/library/man and must be formatted in this directory.

NAME

uman - format manual page in UNIX manual style

DESCRIPTION

The manual pages for the Icon Library can be produced with the head and foot format of the UNIX manual by using the standard UNIX manual macros, as in

```
cd v5/library/man
troff -man man1/rsg.1
```

NOTE

The text for the manual pages includes files in v5/library/man and must be formatted in this directory.