

Scheme→C Index to the Revised⁴ Report on the Algorithmic Language Scheme

Joel F. Bartlett

15 March 1993

Implementation Notes

Scheme→C is an implementation of the language Scheme as described in the *Revised⁴ Report on the Algorithmic Language Scheme (LISP Pointers, Volume IV, Number 3, July-September 1991)*.

The implementation is known to not conform to the required portions of the report in the following ways:

- The syntax for numbers reflects the underlying C implementation. Scheme programs may not use the numeric prefixes #i and #e, and numbers may not contain # as a digit.
- Numerical input and output uses the facilities of the underlying C implementation. As a result, the constraints of section 6.5.6 may not be satisfied.
- As /, quotient, and remainder depend upon C's behavior for negative fixed arguments (which is undefined), those doing ports must verify their correct operation.
- Implementations that do not handle arithmetic overflow traps may return incorrect results when an overflow occurred during the operation.
- The control flow of compiled programs is constrained by the underlying C implementation. As a result, some tail calls are not compiled as tail calls.

The implementation has been extended beyond the report in the following ways:

- Additional procedures:

```
%list->record  
%record  
%record->list
```

```
%record-length  
%record-lookup-method  
%record-methods  
%record-methods-set!  
%record-ref      %record-set!  
%record?  
after-collect  
backtrace  
bit-and          bit-lsh  
bit-not          bit-or  
bit-rsh          bit-xor  
c-byte-ref       c-byte-set!  
c-double-ref     c-double-set!  
c-float-ref      c-float-set!  
c-int-ref        c-int-set!  
c-longint-ref    c-longint-set!  
c-longunsigned-ref  
c-longunsigned-set!  
c-s2cuint-ref   c-s2cuint-set!  
c-shortint-ref  
c-shortint-set!  
c-shortunsigned-ref  
c-shortunsigned-set!  
c-sizeof-double  
c-sizeof-float  
c-sizeof-int     c-sizeof-long  
c-sizeof-s2cuint  
c-sizeof-short  c-sizeof-tscp  
c-string->string  
c-tscp-ref      c-tscp-set!  
c-unsigned-ref  
c-unsigned-set!  
catch-error     close-port  
collect         collect-all  
collect-info    cons*  
define-system-file-task  
echo  
enable-sytem-file-tasks  
error           eval  
exit
```

```

expand                expand-once
fixed->float          fixed?
float->fixed          float?
flush-buffer          format
get-output-string
getprop              getprop-all
implementation-information
last-pair
open-file
open-input-string
open-output-string
optimize-eval
port->stdio-file
pp
proceed              proceed?
putprop
read-eval-print
remove              remove!
remq                remq!
remv                remv!
remove-file          rename-file
reset
reset-bpt            reset-error
scheme-byte-ref
scheme-byte-set!
scheme-int-ref
scheme-int-set!
scheme-s2cuint-ref
scheme-s2cuint-set!
scheme-tscp-ref
scheme-tscp-set!
set-gcinfo!
set-generation-limit!
set-maximum-heap!
set-stack-size!
set-time-slice!
set-top-level-value!
set-write-circle!
set-write-length!
set-write-level!
set-write-pretty!
set-write-width!
signal
stack-size
string->uninterned-symbol
system
time-of-day
time-slice
top-level
top-level-value
uninterned-symbol?

```

```

wait-system-file
weak-cons
when-unreferenced
write-circle          write-count
write-length          write-level
write-pretty          write-width

```

- **Additional syntax:**

```

bpt
define-c-external
define-constant
define-external
define-in-line
define-macro          include
module
trace                  untrace
unbpt
unless                  when

```

- **Additional variables:**

```

%record-prefix-char
%record-read
*args*                  *bpt-env*
debug-output-port
*error-env*
*error-handler*
*frozen-objects*
*obarray*                *result*
*scheme2c-result*
stderr-port              stdin-port
stdout-port
trace-output-port

```

Index

" delimits strings. Inside a string constant, a " is represented by \", and a \ is represented by \\. R⁴RS 25.

#(denotes the start of a vector. R⁴RS 26.

#\character written notation for characters. R⁴RS 24.

#\formfeed ASCII form feed character (#o14). R⁴RS 24.

#\linefeed ASCII line feed character (#o12). R⁴RS 24.

#\newline new line character (#o12). R⁴RS 24.

#\return ASCII carriage return character (#o15). R⁴RS 24.

#\space ASCII space character (#o40). R⁴RS 24.

#\tab ASCII tab character (#o11). R⁴RS 24.

#b binary radix prefix. R⁴RS 20.

#d decimal radix prefix. R⁴RS 20.

#f boolean constant for false. Note that while the empty list () is also treated as a false value in conditional expressions, it is not the same as #f. R⁴RS 13.

#o octal radix prefix. R⁴RS 20.

#t boolean constant for true. R⁴RS 13.

#x hex radix prefix. R⁴RS 20.

(%list->record *list*) returns a newly created *record* whose elements are the members of *list*.

(%record *expression* ...) returns a newly created *record* whose elements contain the given arguments.

(%record->list *record*) returns a newly created *list* of the objects contained in the elements of *record*.

(%record-length *record*) returns the number of elements in *record*.

(%record-lookup-method *record* *method*) returns either the *record*'s method *procedure* or #f when no method is defined for the method named *method*. All records have defaults for the following methods: %to-display, to-equal?, %to-eval, and %to-write.

(%record-methods *record*) returns a list of *pairs* that denote the methods for *record*. Each *pair* is composed of a *symbol* denoting the method name and the method *procedure*.

(%record-methods-set! *record* *methods*) sets the methods associated with *record* to *methods*, a list of method *pairs*.

%record-prefix-char is the character that denotes a *record*.

%record-read is a *procedure* that is called to read a *record*. When read encounters the value of %read-prefix-char following a #, it calls %record-read with the current *input-port* as its

argument to input the record. The value read is the value returned by this procedure.

(%record-ref *record* *integer*) returns the contents of element *integer* of *record*. The first element is 0.

(%record-set! *record* *integer*) sets element *integer* of *record* to *expression*.

(%record? *expression*) *predicate* that returns #t when *expression* is a *record*.

%to-display method to display a *record*. When display encounters a record, it calls the record's %to-display method with the following arguments: the record, the output port, the number of spaces to indent (or #f), the number of levels to print (or #f), the length of lists, vectors, or records to print (or #f), and a list of pairs, vectors, and records already seen (or #f). The method returns either #f indicating no further action is to be taken, or a pair indicating that the car of the pair is to be output. For example, if %record-prefix-char is #\~, the method could be: (lambda (r port . ignore) (display "#~" port) (list (%record->list r)))).

%to-equal? method to compare a *record* to any value using equal?. The method *prediate* is called with the *record* and the comparison value as its arguments. The default method is eq?.

%to-eval method to evaluate a *record*. Eval evaluates a *record* by returning the value of calling the *record*'s %to-eval method with the *record* as the argument. The default method is (lambda (x) x).

%to-write method to write a *record*. When write encounters a record, it calls the record's %to-write method with the following arguments: the record, the output port, the number of spaces to indent (or #f), the number of levels to print (or #f), the length of lists, vectors, or records to print (or #f), and a list of pairs, vectors, and records already seen (or #f). The method returns either #f indicating no further action is to be taken, or a pair indicating that the car of the pair is to be output. For example, if %record-prefix-char is #\~, the method could be: (lambda (r port . ignore) (display "#~" port) (list (%record->list r)))).

'*expression* abbreviation for (quote *expression*).

R⁴RS 7, 16.

(** number ...*) returns the product of its arguments. R⁴RS 21.

args arguments of the *procedure* when a breakpoint has been hit. The value of this symbol will be used as the arguments when the user continues from the breakpoint. See *bpt*, *proceed*.

bpt-env list of environments when a breakpoint is encountered in an embedded Scheme→C system.

error-env list of environments when an error occurs in an embedded Scheme→C system.

error-handler the error handling *procedure*. See *error*.

frozen-objects list of objects that are never moved by the garbage collector. Scheme programs can use this to “lock down” objects in memory before passing them to programs written in other languages.

obarray is a vector of lists of symbols. It is used by *read* to assure that symbols written and then read back in are *equiv?*. See *interned*, R⁴RS 18.

result result of the *procedure* when a breakpoint has been hit. The value of this symbol be returned as the value of the *procedure* after the user continues from the breakpoint. See *bpt*, *proceed*.

scheme2c-result normal result of computation in an embedded Scheme→C system.

'back-quote-template abbreviation for (quasiquote *back-quote template*). R⁴RS 11.

(used to group and notate lists. R⁴RS 5.

() the empty list. R⁴RS 15.

) used to group and notate lists. R⁴RS 5.

(+ *number ...*) returns the sum of its arguments. R⁴RS 21.

, *expression* abbreviation for (unquote *expression*) that causes the expression to be replaced by its value in the *back-quote-template*. R⁴RS 11.

, @*expression* abbreviation for (unquote-splicing *expression*) that causes

the expression to be evaluated and “spliced” into the *back-quote-template*. R⁴RS 11.

(- *number number ...*) with two or more arguments, this returns the difference of its arguments, associating to the left. With one argument it returns the additive inverse of the argument. R⁴RS 21.

-C command line flag to *s2cc* that will cause the compiler to compile the Scheme files *source.sc* to C source in *source.c*. No further processing is performed.

-I *directory* command line flag to *s2cc* to supply a directory to be searched by *include* when it is looking for a source file. When multiple flags are supplied, the directories are searched in the order that the flags are specified.

-LIBDIR *directory* command line flag to *s2cc* to supply the *directory* containing the files: *predef.sc*, *objects.h*, *libs2c.a*, and optionally *libs2c.p.a*.

-Ob command line flag to *s2cc* that controls bounds checking. When it is supplied to the compiler, no bounds checking code for *vector* or *string* accesses will be generated. Supplying this flag is equivalent to supplying the flags -f '*bounds-check*' '#f'.

-Og command line flag to *s2cc* that controls the generation of stack-trace debugging code. When it is supplied to the compiler, stack-trace code will not be generated.

-On command line flag to *s2cc* that controls number representation. When it is supplied to the compiler, all numbers will be assumed to be *fixed* integers. Supplying this flag is equivalent to supplying the flags -f '*fixed-only*' '#t'.

-Ot command line flag to *s2cc* that controls type error checking. When it is supplied, no error checking code will be generated. Supplying this flag is equivalent to supplying the flags -f '*type-check*' '#f'.

-e command line flag to *s2ci*. When it is supplied, all text read on the standard input file will be echoed on the standard output file.

-emacs command line flag to *s2ci*. When supplied, the interpreter assumes that it is being run by GNU emacs.

`-i` command line flag to `s2cc` that will combine the source and object files into a Scheme interpreter. Module names for files other than Scheme source files must be supplied using the `-m` command line flag.

`-log` command line flag to `s2cc` to log information internal to the compiler. Each type of compiler information is denoted by one of the flags: `-source`, `-macro`, `-expand`, `-closed`, `-transform`, `-lambda`, `-tree`, `-lap`, `-peep`. The flag `-log` is equivalent to specifying the flags: `-source`, `-macro`, `-expand`, `-closed`, `-transform`, `-lambda`, and `-tree`.

`-m` *module* command line flag to `s2cc` to specify the name of a module that must be initialized by calling the procedure `module_init`. Note that the Scheme compiler will downshift the alphabetic characters in module names supplied in the `module` directive. Modules are initialized in the order that the `-m` command flags are specified.

`-nh` command line flag to `s2ci`. When it is supplied, the interpreter version header will not be printed on the standard output file.

`-np` command line flag to `s2ci`. When it is supplied, prompts for input from standard input will not be printed on standard output.

`-q` command line flag to `s2ci`. When it is supplied, the result of each expression evaluation will not be printed on standard output.

`-pg` command line flag to `s2cc` that will cause it to produce profiled code for run-time measurement using *gprof*. The profiled library will be used in lieu of the standard Scheme library.

`-scgc` *flag* command line flag to any Scheme program that controls the reporting of garbage collection statistics. If *flag* is set to 1, then garbage collection statistics will be printed on `stderr`. This flag will override `SCGCINFO`.

`-sch` *integer* command line flag to any Scheme program to set the initial heap size in megabytes. If it is not supplied, and the `SCHEAP` environment variable was not set, and the program did not have a default, then the implementation dependent default is used. This flag will override `SCHEAP`.

`-scl` *integer* command line flag to any Scheme program to set the full collection limit. When more

than this percent of the heap is allocated following a generational garbage collection, then a full garbage collection will be done. The default value is 40. This flag will override `SCLIMIT`.

`-scm` *symbol* command line flag to any Scheme program to cause execution to start at the procedure that is the value of *symbol*, rather than at the main program. Note that the Scheme `read` procedure typically upshifts alphabetic characters. Thus, to start execution in the Scheme interpreter, one would enter `-scm READ-EVAL-PRINT` on the command line.

`-scmh` *integer* command line flag to any Scheme program to set the maximum heap size in megabytes. If it is not supplied, and the `SCMAXHEAP` environment variable was not set, then the maximum heap size is five times the initial heap size. This flag will override `SCMAXHEAP`.

`.` denotes a dotted-pair: (*obj* . *obj*). R⁴RS 15.

`.sc` file name extension for Scheme→C source files.

(/ *number* ...) with two or more arguments, this returns the quotient of its arguments, associating to the left. With one argument it returns the multiplicative inverse of the argument. R⁴RS 21.

`;` indicates the start of a comment. The comment continues until the end of the line. R⁴RS 5.

(< *number number number* ...) *predicate* that returns `#t` when the arguments are monotonically increasing. R⁴RS 21.

(<= *number number number* ...) *predicate* that returns `#t` when the arguments are monotonically nondecreasing. R⁴RS 21.

(= *number number number* ...) *predicate* that returns `#t` when the arguments are equal. R⁴RS 21.

`=>` used in a `cond` conditional clause. R⁴RS 9.

(> *number number number* ...) *predicate* that returns `#t` when the arguments are monotonically decreasing. R⁴RS 21.

(>= *number number number* ...) *predicate* that returns `#t` when the arguments are monotonically nonincreasing. R⁴RS 21.

`\` tells `read` to treat the character that follows it as a letter when reading a symbol. If the character

is a lower-case alphabetic character, it will not be upshifted. R⁴RS 18.

`\"` represents a `"` inside a string constant. R⁴RS 25.

`\\` represents a `\` inside a string constant. R⁴RS 25.

(*abs number*) returns the magnitude of its argument. R⁴RS 21.

(*acos number*) returns the arccosine of its argument. R⁴RS 23.

`after-collect` is a variable in the top level environment. Following each garbage collection, if its value is not `#f`, then it is assumed to be a procedure and is called with three arguments: the heap size in bytes, the currently allocated storage in bytes, and the allocation percentage that will cause a full garbage collection. The value returned by the procedure is ignored.

alist a list of *pairs*. R⁴RS 17.

(*and expression ...*) *syntax* for a conditional expression. R⁴RS 9.

(*append list ...*) returns a list consisting of the elements of the first *list* followed by the elements of the other *lists*. R⁴RS 17.

(*apply procedure arg-list*) calls the *procedure* with the elements of *arg-list* as the actual arguments. R⁴RS 27.

(*apply procedure obj ... arg-list*) calls the *procedure* with the list (*append (list obj ...) arg-list*) as the actual arguments. R⁴RS 27.

(*asin number*) returns the arcsine of its argument. R⁴RS 23.

(*assoc obj alist*) finds the first *pair* in *alist* whose *car* field is *equal?* to *obj*. If no such *pair* exists, then `#f` is returned. R⁴RS 17.

(*assq obj alist*) finds the first *pair* in *alist* whose *car* field is *eq?* to *obj*. If no such *pair* exists, then `#f` is returned. R⁴RS 17.

(*assv obj alist*) finds the first *pair* in *alist* whose *car* field is *eqv?* to *obj*. If no such *pair* exists, then `#f` is returned. R⁴RS 17.

(*atan number*) returns the arctangent of its argument. R⁴RS 23.

(*atan number number*) returns the arctangent of its arguments. R⁴RS 23.

(*backtrace*) displays the call stack where a breakpoint occurred.

back-quote-template list or vector structure that may contain *,expression* and *,@expression* forms. R⁴RS 11.

(*begin expression ...*) *syntax* where *expression*'s are evaluated left to right and the value of the last *expression* is returned. R⁴RS 10.

bindings a *list* whose elements are of the form: (*symbol expression*), where the *expression* is the initial value to place in the location bound to the *symbol*. R⁴RS 10.

(*bit-and number ...*) returns an unsigned number representing the bitwise and of its 32-bit arguments.

(*bit-lsh number integer*) returns an unsigned number representing the 32-bit value *number* shifted left *integer* bits.

(*bit-not number ...*) returns an unsigned number representing the bitwise not of its 32-bit argument.

(*bit-or number ...*) returns an unsigned number representing the bitwise inclusive or of its 32-bit arguments.

(*bit-rsh number integer*) returns an unsigned number representing the 32-bit value *number* shifted right *integer* bits.

(*bit-xor number ...*) returns an unsigned number representing the bitwise exclusive or of its 32-bit arguments.

body one or more *expressions* that are to be executed in sequence. R⁴RS 10.

(*boolean? expression*) *predicate* that returns `#t` if *expression* is `#t` or `#f`. R⁴RS 13.

(*bpt*) *syntax* to return a list of the procedures that have been breakpointed.

(*bpt symbol*) *syntax* to set a breakpoint on the *procedure* that is the value of *symbol*. Each entry and exit of the *procedure* will provide the user with an opportunity to examine and alter the current state of the computation. For interactive Scheme- $\rightarrow$ C systems, the computation is continued by entering

control-D. The computation may be terminated and a return made to the top level of the interpreter by entering (top-level). In embedded Scheme->C systems, (proceed) is used to continue the computation, and the computation is abandoned by evaluating (reset-error). See *args*, *result*, top-level, unbpt.

(bpt *symbol procedure*) syntax to set a conditional breakpoint on the *procedure* that is the value of *symbol*. A breakpoint occurs when (apply *procedure arguments*) returns a true value.

(c-byte-ref *c-pointer integer*) returns the byte at the *integer* byte of *c-pointer* as a *number*.

(c-byte-set! *c-pointer integer number*) sets the byte at the *integer* byte of *c-pointer* to *number* and returns *number* as its value.

(c-double-ref *c-pointer integer*) returns the double at the *integer* byte of *c-pointer* as a *number*.

(c-double-set! *c-pointer integer number*) sets the double at the *integer* byte of *c-pointer* to *number* and returns *number* as its value.

(c-float-ref *c-pointer integer*) returns the float at the *integer* byte of *c-pointer* as a *number*.

(c-float-set! *c-pointer integer number*) sets the float at the *integer* byte of *c-pointer* to *number* and returns *number* as its value.

(c-int-ref *c-pointer integer*) returns the int at the *integer* byte of *c-pointer* as a *number*.

(c-int-set! *c-pointer integer number*) sets the int at the *integer* byte of *c-pointer* to *number* and returns *number* as its value.

(c-longint-ref *c-pointer integer*) returns the long int at the *integer* byte of *c-pointer* as a *number*.

(c-longint-set! *c-pointer integer number*) sets the long int at the *integer* byte of *c-pointer* to *number* and returns *number* as its value.

(c-longunsigned-ref *c-pointer integer*) returns the unsigned long at the *integer* byte of *c-pointer* as a *number*.

(c-longunsigned-set! *c-pointer integer number*) sets the unsigned long at the *integer* byte of *c-pointer* to *number* and returns *number* as its value.

c-pointer a *number* that is the address of a structure outside the Scheme heap, or a *string* that is a C-structure within the Scheme heap.

(c-s2cuint-ref *c-pointer integer*) returns the S2CUINT at the *integer* byte of *c-pointer* as a *number*.

(c-s2cuint-set! *c-pointer integer number*) sets the S2CUINT at the *integer* byte of *c-pointer* to *number* and returns *number* as its value.

(c-shortint-ref *c-pointer integer*) returns the short int at the *integer* byte of *c-pointer* as a *number*.

(c-shortint-set! *c-pointer integer number*) sets the short int at the *integer* byte of *c-pointer* to *number* and returns *number* as its value.

(c-shortunsigned-ref *c-pointer integer*) returns the unsigned short at the *integer* byte of *c-pointer* as a *number*.

(c-shortunsigned-set! *c-pointer integer number*) sets the unsigned short at the *integer* byte of *c-pointer* to *number* and returns *number* as its value.

c-sizeof-double size (in bytes) of the C type double.

c-sizeof-float size (in bytes) of the C type float.

c-sizeof-int size (in bytes) of the C type int.

c-sizeof-long size (in bytes) of the C type long.

c-sizeof-s2cuint size (in bytes) of the C type S2CUINT that is defined by Scheme->C to be an unsigned integer the same size as a pointer.

c-sizeof-short size (in bytes) of the C type short.

c-sizeof-tscp size (in bytes) of the C type TSCP that is defined by Scheme->C to represent tagged Scheme pointers.

(c-string->string *c-pointer*) returns a Scheme *string* that is a copy of the null-terminated string *c-pointer*.

(c-tscp-ref *c-pointer integer*) returns the TSCP at the *integer* byte of *c-pointer*.

(c-tscp-set! *c-pointer integer expression*) sets

the TSCP at the *integer* byte of *c-pointer* to *expression* and returns *expression* as its value.

(c-unsigned-ref *c-pointer integer*) returns the unsigned at the *integer* byte of *c-pointer* as a *number*.

(c-unsigned-set! *c-pointer integer number*) sets the unsigned at the *integer* byte of *c-pointer* to *number* and returns *number* as its value.

c-type syntax for declaring the type of a non-Scheme procedure, procedure argument, or global. The allowed types are: `pointer`, `array`, `char`, `int`, `shortint`, `longint`, `unsigned`, `shortunsigned`, `longunsigned`, `float`, `double`, `tscp`, or `void`.

(car *pair*) returns the contents of the `car` field of the *pair*. R⁴RS 16.

(caar *pair*) returns (car (car *pair*)). R⁴RS 16.

(ca...r *pair*) compositions of `car` and `cdr`. R⁴RS 16.

(call-with-current-continuation *procedure*) calls *procedure* with the current continuation as its argument. R⁴RS 28.

(call-with-input-file *string procedure*) calls *procedure* with the *port* that is the result of opening the file *string* for input. R⁴RS 29.

(call-with-output-file *string procedure*) calls *procedure* with the *port* that is the result of opening the file *string* for output. R⁴RS 29.

(case *key clause clause ...*) *syntax* for a conditional expression where *key* is any expression, and each *clause* is of the form ((*datum ...*) *expression expression ...*). The last clause may be an “else clause” of the form (else *expression expression ...*). R⁴RS 9.

(catch-error *procedure*) calls *procedure* with no arguments. If an error occurs while executing *procedure*, return a string containing the error message. Otherwise return a *pair* whose `car` contains the procedure’s value.

(cdr *pair*) returns the contents of the `cdr` field of the *pair*. R⁴RS 16.

(cd...r *pair*) compositions of `car` and `cdr`. R⁴RS 16.

(cddddr *pair*) returns (cdr (cdr (cdr (cdr *pair*)))). R⁴RS 16.

(ceiling *number*) returns the smallest integer that is not smaller than its arguments. R⁴RS 22.

char syntax for declaring a non-Scheme procedure, procedure argument, or global variable as the C type `char`. When a `char` value must be supplied, an expression of type *character* must be supplied. When a `char` value is returned, a value of type *character* will be returned.

(char->integer *character*) returns an *integer* whose value is the ASCII character code of *character*. R⁴RS 25.

(char-alphabetic? *character*) *predicate* that returns #t when *character* is alphabetic. R⁴RS 25.

(char-ci<=? *character character*) *predicate* that returns #t when the first *character* is less than or equal to the second *character*. Upper case and lower case letters are treated as though they were the same character. R⁴RS 25.

(char-ci<? *character character*) *predicate* that returns #t when the first *character* is less than the second *character*. Upper case and lower case letters are treated as though they were the same character. R⁴RS 25.

(char-ci=? *character character*) *predicate* that returns #t when the first *character* is equal to the second *character*. Upper case and lower case letters are treated as though they were the same character. R⁴RS 25.

(char-ci>=? *character character*) *predicate* that returns #t when the first *character* is greater than or equal to the second *character*. Upper case and lower case letters are treated as though they were the same character. R⁴RS 25.

(char-ci>? *character character*) *predicate* that returns #t when the first *character* is greater than the second *character*. Upper case and lower case letters are treated as though they were the same character. R⁴RS 25.

(char-downcase *character*) returns the lower case value of *character*. R⁴RS 25.

(char-lower-case? *letter*) *predicate* that returns #t when *letter* is lower-case. R⁴RS 25.

(char-numeric? *character*) *predicate* that returns #t when *character* is numeric. R⁴RS 25.

(char-ready? *optional-input-port*) *predicate* that

returns #t when a character is ready on the *optional-input-port*. R⁴RS 30.

(char-upcase *character*) returns the upper case value of the *character*. R⁴RS 25.

(char-upper-case? *letter*) *predicate* that returns #t when *letter* is upper-case. R⁴RS 25.

(char-whitespace? *character*) *predicate* that returns #t when *character* is a whitespace character. R⁴RS 25.

(char<=? *character character*) *predicate* that returns #t when the first *character* is less than or equal to the second *character*. R⁴RS 24.

(char<? *character character*) *predicate* that returns #t when the first *character* is less than the second *character*. R⁴RS 24.

(char=? *character character*) *predicate* that returns #t when the first *character* is equal to the second *character*. R⁴RS 24.

(char>=? *character character*) *predicate* that returns #t when the first *character* is greater than or equal to the second *character*. R⁴RS 24.

(char>? *character character*) *predicate* that returns #t when the first *character* is greater than the second *character*. R⁴RS 24.

(char? *expression*) *predicate* that returns #t when *expression* is a *character*. R⁴RS 24.

character Scheme object that represents printed characters. See #\character, #\character-name, R⁴RS 24.

(close-input-port *input-port*) closes the file associated with *input-port*. R⁴RS 30.

(close-output-port *output-port*) closes the file associated with *output-port*. R⁴RS 30.

(close-port *port*) closes the file associated with *port*.

(collect) invokes the garbage collector to perform a generational collection. Normally, garbage collection is invoked automatically by the Scheme system.

(collect-all) invokes the garbage collector to perform a full collection. Normally, garbage collection is invoked automatically by the Scheme system.

(collect-info) returns a *list* containing information about heap and processor usage. The items in the list (and their position) are: number of bytes currently allocated (0), current heap size in bytes (1), application processor seconds (2), garbage collection processor seconds (3), maximum heap size in bytes (4), full collection limit percent (5).

complex number complex numbers are not supported in Scheme->C. R⁴RS 18.

(complex? *expression*) *predicate* that returns #t when *expression* is a *complex number*. All Scheme->C numbers are complex. R⁴RS 20.

(cond *clause clause ...*) *syntax* for a conditional expression where each *clause* is of the form (*test expression ...*) or (*test => procedure*). The last *clause* may be of the form (*else expression expression ...*). R⁴RS 9.

(cons *expression₁ expression₂*) returns a newly allocated *pair* that has *expression₁* as its car, and *expression₂* as its cdr. R⁴RS 16.

(cons* *expression expression ...*) returns an object formed by consing the *expressions* together from right to left. If only one *expression* is supplied, then that *expression* is returned.

(cos *number*) returns the cosine of its argument. R⁴RS 23.

(current-input-port) returns the current default input *port*. R⁴RS 30.

(current-output-port) returns the current default output *port*. R⁴RS 30.

debug-output-port port used for interactive debugging output. The default value is the same as stderr-port.

(define *symbol expression*) *syntax* that defines the value of *expression* as the value of either a top-level symbol or a local variable. R⁴RS 12.

(define (*symbol formals*) *body*) *syntax* that defines a *procedure* that is either the value of a top-level symbol or a local variable. R⁴RS 12.

(define (*symbol . formal*) *body*) *syntax* that defines a *procedure* that is either the value of a top-level symbol or a local variable. R⁴RS 12.

(define-c-external *symbol c-type string*) *syntax* for a compiler declaration that defines *symbol*

as a non-Scheme global variable with the name *string* and the type *c-type*.

(define-c-external (*symbol c-type₁...*) *c-type₂* *string*) *syntax* for a compiler declaration that defines *symbol* as a non-Scheme procedure with arguments of the type specified in the list *c-type₁*. The procedure's name is *string* and it returns a value of type *c-type₂*.

(define-c-external (*symbol c-type₁... c-type₂*) *c-type₃* *string*) *syntax* for a compiler declaration that defines *symbol* as a non-Scheme procedure that takes a variable number of arguments. The types of the initial arguments are specified by the list *c-type₁*. Any additional arguments must be of the type *c-type₂*. The procedure's name is *string* and it returns a value of type *c-type₃*.

(define-constant *symbol expression*) *syntax* that defines a macro that replaces all occurrences of *symbol* with the value of *expression*, evaluated at the time of the definition.

(define-external *symbol₁ symbol₂*) *syntax* for a compiler declaration that *symbol₁* is defined in module *symbol₂*.

(define-external *symbol* TOP-LEVEL) *syntax* for a compiler declaration that *symbol* is a top-level symbol. Its value is to be found via the **obarray**.

(define-external *symbol₁ TOP-LEVEL symbol₂*) *syntax* for a compiler declaration that *symbol₁* is a top-level symbol that is known to be defined in module *symbol₂*. Its value is to be found via the **obarray**.

(define-external *symbol* " *string*) *syntax* for a compiler declaration that *symbol* has the external name *string*.

(define-external *symbol string₁ string₂*) *syntax* for a compiler declaration that *symbol* is in the module *string₁* and has the external name *string₁string₂*.

(define-external (*symbol₁ formals*) *symbol₂*) *syntax* for a compiler declaration that *symbol₁* is a Scheme procedure defined in module *symbol₂*.

(define-external (*symbol₁ . formal*) *symbol₂*) *syntax* for a compiler declaration that *symbol₁* is a Scheme procedure defined in module *symbol₂*.

(define-external (*symbol formals*) " *string*) *syntax* for a compiler declaration that *symbol* is a procedure that has the external name *string*.

(define-external (*symbol . formal*) " *string*) *syntax* for a compiler declaration that *symbol* is a procedure that takes a variable number of arguments and has the external name *string*.

(define-external (*symbol formals*) *string₁ string₂*) *syntax* for a compiler declaration that *symbol* is a procedure in the module *string₁* that has the external name *string₁string₂*.

(define-external (*symbol . formal*) *string₁ string₂*) *syntax* for a compiler declaration that *symbol* is a procedure in the module *string₁* that has the external name *string₁string₂*.

(define-in-line (*symbol formals*) *body*) *syntax* that defines a procedure that is to be compiled "in-line".

(define-in-line (*symbol . formal*) *body*) *syntax* that defines a procedure that is to be compiled "in-line".

(define-macro *symbol* (lambda (*form expander*) *expression ...*) *syntax* that defines a macro expansion procedure. Macro expansion is done using the ideas expressed in *Expansion-Passing Style: Beyond Conventional Macros*, 1986 ACM Conference on Lisp and Functional Programming, 143-150.

(define-system-file-task *file idle-task file-task*) installs the *idle-task* and *file-task* procedures for system file number *file*. When a Scheme program reads from a port and no characters are internally buffered, the *idle-task* for each system file is called. Then, the *file-task* for each system file that has input pending is called. As long as no characters are available on the Scheme port, the Scheme system will idle, calling the *file-task* for each system file as input becomes available. A system file task is removed by supplying #f as the *idle-task* and *file-task*.

(delay *expression*) *syntax* used together with the procedure *force* to implement call by need. R⁴RS 11.

(display *expression optional-output-port*) writes a human-readable representation of *expression* to *optional-output-port*. R⁴RS 31.

(do (*var ...*) (*test expression ...*) *command ...*) *syntax*

for an iteration construct. Each *var* defines a local variable and is of the form (*symbol init step*) or (*symbol init*). R⁴RS 11.

double syntax for declaring a non-Scheme procedure, procedure argument, or global variable as the C type *double*. When a *double* value must be supplied, an expression of type *number* must be supplied. When a *double* value is returned, a value of type *number* is returned.

(*echo port*) turns off echoing on *port*.

(*echo port output-port*) echos *port* on *output-port*. All characters read from or written to *port* are also written to *output-port*.

else keyword in last *clause* of *cond* or *case* form.

environment the set of all variable bindings in effect at some point in the program. R⁴RS 6.

(*eof-object? expression*) *predicate* that returns #t if *expression* is equal to the end of file object. R⁴RS 30.

(*enable-system-file-tasks flag*) enables (*flag* is #t) or disables (*flag* is #f) system file tasking and returns the previous system file tasking state. When the value of *flag* is the symbol *wait*, system file tasking is enabled and the Scheme program is blocked until there are no system file tasks.

(*eq? expression₁ expression₂*) *predicate* that is the finest test for equivalence between *expression₁* and *expression₂*. R⁴RS 15.

(*equal? expression₁ expression₂*) *predicate* that is the coarsest test for equivalence between *expression₁* and *expression₂*. R⁴RS 15.

(*eqv? expression₁ expression₂*) *predicate* that is the medium test for equivalence between *expression₁* and *expression₂*. R⁴RS 13.

(*error symbol format-template expression ...*) reports an error. The procedure name is *symbol* and the error message is produced by the *format-template* and optional *expressions*. The *procedure* error is equivalent to (*lambda x (apply *error-handler* x)*). See **error-handler**.

(*eval expression*) evaluates *expression*. Any macros in *expression* are expanded before evaluation.

(*eval-when list expression ...*) *syntax* to evaluate *expressions* when the current situation is in *list*. When this form is evaluated by the Scheme interpreter and *eval* is a member of the situation *list*, then the expressions will be evaluated. When this form is evaluated by the Scheme compiler and *compile* is a member of the situation *list*, then the expressions will be evaluated within the compiler. When this form is evaluated by the Scheme compiler, and *load* is a member of the situation *list*, then the compiler will compile the form (*begin expression ...*).

(*even? integer*) *predicate* that returns #t if *integer* is even. R⁴RS 21.

exact *fixed* numbers are exact, all other numbers are not. R⁴RS 14.

(*exact->inexact number*) returns the *inexact* representation of *number*. R⁴RS 23.

(*exact? number*) *predicate* that returns #t if *number* is *exact*. R⁴RS 21.

(*exit*) returns from the current read-eval-print procedure.

(*exp number*) returns exponential function of *number*. R⁴RS 22.

(*expand expression*) returns the value of *expression* after all macro expansions. See *define-macro*.

(*expand-once expression*) returns the value of *expression* after one macro expansion. See *define-macro*.

expression a Scheme construct that returns a value. R⁴RS 7.

(*expt number₁ number₂*) returns *number₁* raised to the power *number₂*. R⁴RS 23.

fix *format descriptor* for compatibility with R³RS.

fixed Scheme->C internal representation for small *integers*. A *fixed* value is represented in a “pointer size” word with two bits used by the tag. With 32-bit pointers, this yields a maximum value of $2^{29} - 1$ or 536,870,911 and a minimum value of -2^{29} or -536,870,912. With 64-bit pointers, this yields a maximum value of $2^{61} - 1$ or 2,305,843,009,213,693,951 and a minimum value of -2^{61} or -2,305,843,009,213,693,952.

(fixed->float *fixed*) returns the *float* representation of *fixed*.

(fixed? *expression*) *predicate* that returns #t when *expression* is a *fixed*.

float *syntax* for declaring a non-Scheme procedure, procedure argument, or global variable as the C type float. When a float value must be supplied, an expression of type *number* must be supplied. When a float value is returned, a value of type *number* is returned.

float Scheme->C internal floating point representation. This is typically 64-bits.

(float->fixed *float*) returns the *fixed number* that best represents the value of *float*.

(float? *expression*) *predicate* that returns #t if *expression* is a *float* value.

(floor *number*) returns the largest *integer* not larger than *number*. R⁴RS 22.

(flush-buffer *optional-output-port*) forces output of all characters buffered in *optional-output-port*.

(for-each *procedure list list* ...) applies *procedure* to each element of the *lists* in order. R⁴RS 28.

(force *promise*) returns the forced value of a promise. R⁴RS 28.

formals a *symbol* or a *list* of *symbols* that are the arguments. R⁴RS 8.

(format #f *format-template expression* ...) returns a string that is the result of outputting the *expressions* according to the *format-template*.

(format *format-template expression* ...) returns a string that is the result of outputting the *expressions* according to the *format-template*.

(format *output-port format-template expression* ...) output the *expressions* to *output-port* according to the *format-template*.

(format #t *format-template expression* ...) output the *expressions* to the current output port according to the *format-template*.

format descriptor a *list* that describes the type of output conversion to be done by number->string. The supported forms are (int), (fix *integer*), and (s2ci *integer*). R⁴RS 21.

format-template a *string* consisting of format descriptors and literal characters. A format descriptor is ~ followed by some other character. When one is encountered, it is interpreted. Literal characters are output as is. See ~a, ~A, ~C, ~c, ~s, ~S, ~%, ~~.

(gcd *number* ...) returns the greatest common divisor of its arguments. R⁴RS 22.

(get-output-string *string-output-port*) returns the *string* associated with *string-output-port*. The *string* associated with the *string-output-port* is initially set to "".

(getprop *symbol expression*) returns the value that has the key eq? to *expression* from *symbol*'s property list. If there is no value associated with *expression*, then #f is returned.

(getprop-all *symbol*) returns the *symbol*'s property list.

(implementation-information) returns a list of string or #f values containing information about the Scheme implementation. The list is of the form (*implementation-name version machine processor operating-system filesystem features* ...).

(if *expression*₁ *expression*₂) *syntax* for a conditional expression. R⁴RS 8.

(if *expression*₁ *expression*₂ *expression*₃) *syntax* for a conditional expression. R⁴RS 8.

(include *string*) *syntax* to include the contents of the file *string* at this point in the Scheme compilation. Search directories may be specified by the -I command flag.

inexact float numbers are inexact. R⁴RS 14.

(inexact->exact *number*) returns the *exact* representation of *number*. R⁴RS 23.

(inexact? *number*) *predicate* that returns #t when *number* is *inexact*. R⁴RS 21.

input-port Scheme object that can deliver characters on command. R⁴RS 29.

(input-port? *expression*) *predicate* when returns #t when *expression* is an *input-port*. R⁴RS 29.

int *syntax* for declaring a non-Scheme procedure, procedure argument, or global variable as the C type int. When a int value must be supplied,

an expression of type *number* must be supplied. When a `int` value is returned, a value of type *number* is returned.

`int format descriptor` for compatibility with R³RS.

integer integers are represented by both *fixed* and *float* values. R⁴RS 18.

`(integer->char integer)` returns the *character* whose ASCII code is equal to *integer*. R⁴RS 25.

`(integer? expression)` *predicate* that returns `#t` when *expression* is an *integer*. R⁴RS 20.

interned symbols that are contained in `*obarray*` are interned.

`(lambda formal body)` the ultimate imperative, the ultimate declarative. R⁴RS 8.

`(last-pair list)` returns the last *pair* of *list*.

`(lcm number ...)` returns the least common multiple of its arguments. R⁴RS 22.

`(length list)` returns the length of *list*. R⁴RS 17.

`(let bindings body)` *syntax* for a binding construct that computes initial values before any bindings are done. R⁴RS 10.

`(let symbol bindings body)` *syntax* for a general looping construct. R⁴RS 11.

`(let* bindings body)` *syntax* for a binding construct that computes initial values and performs bindings sequentially. R⁴RS 10.

`(letrec bindings body)` *syntax* for a binding construct that binds the variables before the initial values are computed. R⁴RS 10.

letter an alphabetic character. R⁴RS 25.

list the empty list, or a *pair* whose `cdr` is a *list*. R⁴RS 16.

`(list expression ...)` returns a *list* of its arguments. R⁴RS 17.

`(list? expression)` *predicate* that returns `#t` when *expression* is a *list*. R⁴RS 16.

`(list->string list)` returns the string formed from the *characters* in *list*. R⁴RS 26.

`(list->vector list)` returns a *vector* whose elements are the members of *list*. R⁴RS 27.

`(list-ref list integer)` returns the *integer* element of *list*. Elements are numbered starting at 0. R⁴RS 17.

`(list-tail list integer)` returns the sublist of *list* obtained by omitting the first *integer* elements. R⁴RS 17.

`(load string)` loads the expressions in the file *string* into the Scheme interpreter. The results of the expressions are printed on the current output port. R⁴RS 31.

`(load string)` loads the expressions in the file *string* into the Scheme interpreter. The contents of the file and the results of the expressions are printed on the current output port.

`(loadq string)` loads the expressions in the file *string* into the Scheme interpreter. The results of the expressions are not printed.

`(log number)` returns the natural logarithm of *number*. R⁴RS 22.

longint syntax for declaring a non-Scheme procedure, procedure argument, or global variable as the C type `long int`. When a `long int` value must be supplied, an expression of type *number* must be supplied. When a `long int` value is returned, a value of type *number* is returned.

longunsigned syntax for declaring a non-Scheme procedure, procedure argument, or global variable as the C type `long unsigned`. When a `long unsigned` value must be supplied, an expression of type *number* must be supplied. When a `long unsigned` value is returned, a value of type *number* is returned.

`(make-string integer)` returns a string of length *integer* with unknown elements. R⁴RS 25.

`(make-string integer char)` returns a string of length *integer* with all elements initialized to *char*. R⁴RS 25.

`(make-vector integer)` returns a vector of length *integer* with unknown elements. R⁴RS 26.

`(make-vector integer expression)` returns a vector of length *integer* with all elements set to *expression*. R⁴RS 26.

`(map procedure list list ...)` returns a *list* constructed by applying *procedure* to each element of

the *lists*. The order of application is not defined. R⁴RS 27.

(max *number number ...*) returns the maximum of its arguments. R⁴RS 21.

(member *expression list*) returns the first *sublist* of *list* such that (equal? *expression* (car *sublist*)) is true. If no match occurs, then #f is returned. R⁴RS 17.

(memq *expression list*) returns the first *sublist* of *list* such that (eq? *expression* (car *sublist*)) is true. If no match occurs, then #f is returned. R⁴RS 17.

(memv *expression list*) returns the first *sublist* of *list* such that (eqv? *expression* (car *sublist*)) is true. If no match occurs, then #f is returned. R⁴RS 17.

(min *number number ...*) returns the minimum of its arguments. R⁴RS 21.

(module *symbol clause ...*) *syntax* to declare module information for the Scheme→C compiler. The *module* form must be the first item in the source file. The module name is a *symbol* that must be a legal C identifier. Using this information, the compiler is able to construct an object module that is similar in structure to a Modula 2 module. Following the module name come optional *clauses*. If the module is to provide the “main” program, then a *clause* of the form (main *symbol*) is provided that indicates that *symbol* is the initial *procedure*. It will be invoked with one argument that is a *list* of *strings* that are the arguments that the program was invoked with. A minimum (and default) heap size can be specified by the *clause* (HEAP *integer*), where the size is specified in megabytes. The user may control that top-level *symbols* in this module are visible as top-level *symbols* by including a *clause* of the form (top-level *symbol ...*). If this clause occurs, then only those *symbols* specified will be made top-level. All other top-level *symbols* in the module will appear at the top-level with names of the form: *module_symbol*. If a top-level clause is not provided, then all top-level *symbols* in the module will be made top-level. The final clause, (with *symbol ...*) indicates that this module will be linked with other modules. Normally the intermodule linkages are automatically inferred by including all *modules* that have external references. However, this mechanism is not sufficient to pick up those objects that

are only referenced at runtime.

(modulo *integer₁ integer₂*) returns the modulo of its arguments. The sign of the result is the sign of the divisor. R⁴RS 22.

(negative? *number*) *predicate* that returns #t when *number* is negative. R⁴RS 21.

(newline *optional-output-port*) outputs a newline character on the *optional-output-port*. R⁴RS 31.

(not *expression*) *predicate* that returns #t when *expression* is #f or (). R⁴RS 13.

(null? *expression*) *predicate* that returns #t when *expression* is (). R⁴RS 16.

number Scheme→C has two internal representations for numbers: *fixed* and *float*. When an arithmetic operation is to be performed with a *float* argument, all arguments will be converted to *float* as needed, and then the operation will be performed. Automatic conversion back to *fixed* is never done. R⁴RS 18.

(number→string *number format descriptor*) returns a *string* that is the printed representation of *number* as specified by *format descriptor*. For compatibility with R³RS.

(number→string *number*) returns a string with the printed representation of the number. R⁴RS 23.

(number→string *number radix*) returns a string with the printed representation of the number in the given radix. Radix must be 2, 8, 10, or 16. R⁴RS 23.

(number? *expression*) *predicate* that returns #t when *expression* is a *number*. R⁴RS 20.

(odd? *integer*) *predicate* that returns #t when *integer* is odd. R⁴RS 21.

(open-file *string₁ string₂*) returns a *port* for file *string₁* that is opened using the operating system's *fopen* option *string₂*.

(open-input-file *string*) returns an *input port* capable of delivering characters from the file *string*. R⁴RS 30.

(open-input-string *string*) returns an *input port* capable of delivering characters from the *string*.

(open-output-file *string*) returns an *output port* capable of delivering characters to the file *string*. R⁴RS 30.

(open-output-string) returns an *output port* capable of delivering characters to a *string*. See `get-output-string`.

(optimize-eval *option...*) controls the optimization done on interpreted programs. When no *option* is supplied, minimal optimization is done. When `call` is specified, calls to top-level procedures that are not interpreted are optimized. When `rewrite` is specified, calls to top-level procedures that take variable number of arguments are rewritten. This option may cause some breakpoints to be missed. Both `call` and `rewrite` may be specified.

optional-input-port if present, it must be an *input-port*. If not present, then it is the value returned by `current-input-port`.

optional-output-port if present, it must be an *output-port*. If not present, then it is the value returned by `current-output-port`.

(or *expression ...*) *syntax* for a conditional expression. R⁴RS 9.

pair record structure with two fields: `car` and `cdr`. R⁴RS 15.

(pair? *expression*) *predicate* that returns #t when *expression* is a *pair*. R⁴RS 16.

(peek-char *optional-input-port*) returns a copy of the next character available on *optional-input-port*. R⁴RS 30.

pointer syntax for declaring a non-Scheme procedure, procedure argument, or global variable as being some type of C pointer. When a value must be supplied, an expression of the type *string*, *procedure*, or *number* is supplied. This will result in either the address of the first character of the *string*, the address of the code associated with the *procedure*, or the value of the number being used. A *pointer* value is returned as a non-negative *number*.

port Scheme object that is capable of delivering or accepting characters on demand. R⁴RS 29.

(port->stdio-file *port*) returns the standard I/O FILE pointer for *port*, or #f if the *port* does not use standard I/O.

(positive? *number*) *predicate* that returns #t when *number* is positive. R⁴RS 21.

(pp *expression optional-output-port*) pretty-prints *expression* on *optional-output-port*.

(pp *expression string*) pretty-prints *expression* to the file *string*.

predicate function that returns #t when the condition is true, and #f when the condition is false. R⁴RS 13.

(procedure? *expression*) *predicate* that returns #t when *expression* is a *procedure*. R⁴RS 27.

(proceed) return from the innermost read-eval-print loop with an unspecified value.

(proceed) resume the computation that previously timed out in an embedded Scheme->C system, or was stopped at a breakpoint.

(proceed *expression*) return from the innermost read-eval-print loop with *expression* as the value. At the outermost level, *expression* must be an *integer* as it will be used as the argument for a call to the C library procedure `exit`.

(proceed *expression*) return *expression* as the value of a procedure that stopped at a breakpoint.

(proceed?) force a breakpoint while resuming the computation that previously timed out in an embedded Scheme->C system.

(putprop *symbol expression₁ expression₂*) stores *expression₂* using key *expression₁* on *symbol*'s property list. See `getprop`.

(quasiquote *back-quote-template*) *syntax* for a *vector* or *list* constructor. R⁴RS 11.

(quote *expression*) *syntax* whose result is *expression*. R⁴RS 7.

(quotient *integer₁ integer₂*) returns the quotient of its arguments. The sign is the sign of the product of its arguments. R⁴RS 22.

(rational? *number*) *predicate* that returns #t when its argument is a rational *number*. This is true for any number in Scheme->C. R⁴RS 20.

(read *optional-input-port*) returns the next readable object from *optional-input-port*. Revised³ 30.

(read-char *optional-input-port*) returns the next character from *optional-input-port*, updating the *port* to point to the next *character*. Revised³ 30.

(read-eval-print *expression* ...) starts a new read-eval-print loop. The optional *expressions* allow one to specify the prompt or the header: PROMPT *string* HEADER *string*. Typing control-D at the prompt will terminate the procedure. See reset, exit, eval, proceed.

(real? *number*) predicate that returns #t when its argument is a real *number*. This is true in Scheme→C for any *number*. R⁴RS 20.

record a heterogeneous mutable structure whose elements are indexed by *integers*. The valid indexes of a record are the exact non-negative integers less than the length of the record. A *record* differs from a *vector* in that a *record* may have method *procedures* that control how its output, compared, and evaluated.

(remainder *integer*₁ *integer*₂) returns the remainder of its arguments. The sign is the sign of *integer*₁. R⁴RS 22.

(remove *expression list*) returns a new *list* that is a copy of *list* with all items equal? to *expression* removed from it.

(remove! *expression list*) returns *list* having deleted all items equal? to *expression* from it.

(remove-file *string*) removes the file named *string*.

(remq *expression list*) returns a new *list* that is a copy of *list* with all items eq? to *expression* removed from it.

(remq! *expression list*) returns *list* having deleted all items eq? to *expression* from it.

(remv *expression list*) returns a new *list* that is a copy of *list* with all items eqv? to *expression* removed from it.

(remv! *expression list*) returns *list* having deleted all items eqv? to *expression* from it.

(rename-file *string*₁ *string*₂) changes the name of the file named *string*₁ to *string*₂.

(reset) returns to the current read-eval-print loop.

(reset-bpt) indicates that the caller wishes to cancel the resumption of computation at the point where a breakpoint occurred in an embedded Scheme→C system.

(reset-error) indicates that the caller is finished examining the last retained error state in an embedded Scheme→C system.

(reverse *list*) returns a new *list* with the elements of *list* in reverse order. R⁴RS 17.

(round *number*) returns *number* rounded to the closest integer. R⁴RS 22.

S2CUINT C type defined by Scheme→C to be an unsigned integer that is the same size as a pointer.

sc-pointer a Scheme object that is represented by a tagged pointer to one or more words of memory.

sc... all modules that compose the Scheme→C runtime system have module names beginning with the letters sc. All procedures and external variables in these modules have names that begin with sc... .

s2cc shell command to invoke the Scheme→C Scheme compiler. See the man page.

SCGCINFO environment variable that when set to 1 will log garbage collection information on stderr. This variable is overridden by the -scgc command line flag.

SCHEAP environment variable that controls the initial heap size. It is set to the desired size in megabytes. If not set, then the default in the main program will be used. If a default size is not supplied, then the implementation default is used. This variable is overridden by the -sch command line flag.

SCLIMIT environment variable that controls the amount of heap retained after a generational garbage collection that will force a full collection. It is expressed as a percent of the heap. The default value is 40. This variable is overridden by the -scl command line flag.

SCMAXHEAP environment variable that controls the maximum heap size. It is set to the desired size in megabytes. If not set and the -scmh command line flag is not supplied, the maximum heap size is five times the initial heap size. This variable is overridden by the -scmh command line flag.

(scheme-byte-ref *sc-pointer integer*) returns the byte at the *integer* byte of *sc-pointer* as a *number*.

(scheme-byte-set! *sc-pointer integer number*) sets the byte at the *integer* byte of *sc-pointer* to *number*. The procedure returns *number* as its value.

(scheme-int-ref *sc-pointer integer*) return the int at the *integer* byte of *sc-pointer* as a *number*.

(scheme-int-set! *sc-pointer integer number*) sets the int at the *integer* byte of *sc-pointer* to *number*. The procedure returns *number* as its value.

(scheme-s2cuint-ref *sc-pointer integer*) returns the S2CUINT at the *integer* byte of *sc-pointer*.

(scheme-s2cuint-set! *sc-pointer integer expression*) sets the S2CUINT at the *integer* byte of *sc-pointer* to *expression*. The procedure returns *expression* as its value.

(scheme-tscp-ref *sc-pointer integer*) returns the TSCP at the *integer* byte of *sc-pointer*.

(scheme-tscp-set! *sc-pointer integer expression*) sets the TSCP at the *integer* byte of *sc-pointer* to *expression*. The procedure returns *expression* as its value.

`s2ci` shell command to invoke the Scheme->C Scheme interpreter. See the `man` page.

`s2ci` *format descriptor* for compatibility with R³RS.

(set! *symbol expression*) *syntax* to set the location bound to *symbol* to the value of *expression*. R⁴RS 9.

(set-car! *pair expression*) sets the car field of *pair* to *expression*. R⁴RS 16.

(set-cdr! *pair expression*) sets the cdr field of *pair* to *expression*. R⁴RS 16.

(set-gcinfo! *integer*) sets the flag controlling the printing of garbage collection statistics to *integer*. See `-scgc`.

(set-generation-limit! *integer*) sets the full collection limit to *integer*. See `-scl`.

(set-maximum-heap! *integer*) sets the maximum heap size to *integer* megabytes. See `-scmh`.

(set-stack-size! *expression*) sets the size of the stack used by Scheme->C to *expression* bytes. This value is ignored if the system does not do explicit stack overflow checking.

(set-time-slice! *expression*) sets the time slice used by the Scheme->C to *expression* ticks. This value is decremented each time a Scheme procedure is called, and the time slice expires when it becomes zero. This value is ignored if the system does not do explicit time slicing.

(set-top-level-value! *symbol expression*) sets the top-level location bound to *symbol* to *value*.

(set-write-circle! *boolean optional-output-port*) controls circular object detection on output to *optional-output-port*. If *boolean* is `#t`, then circular objects are printed as "...". If *boolean* is `#f`, circular object detection is disabled.

(set-write-length! *integer optional-output-port*) sets the list and vector length limits of *optional-output-port* to *integer*. Vectors and lists longer than *integer* have their remaining elements printed as "...".

(set-write-length! `#f optional-output-port`) allows arbitrary length list and vector printing on *optional-output-port*.

(set-write-level! *integer optional-output-port*) sets the number of levels that nested vectors and lists are printed on *optional-output-port* to *integer*. Vectors and lists nesting deeper than this level are printed as "#".

(set-write-level! `#f optional-output-port`) allows arbitrarily deep nested list and vector printing on *optional-output-port*.

(set-write-pretty! *boolean optional-output-port*) controls "pretty-printing" on *optional-output-port*. If *boolean* is `#t`, then output is printed in a more readable form in `write-width` wide lines. A value of `#f` enables normal output.

(set-write-width! *integer optional-output-port*) sets the width of *optional-output-port* to *integer*.

shortint syntax for declaring a non-Scheme procedure, procedure argument, or global variable as the C type `short int`. When a `short int` value must be supplied, an expression of type *number*

must be supplied. When a `short int` value is returned, a value of type *number* is returned.

`shortunsigned` *syntax* for declaring a non-Scheme procedure, procedure argument, or global variable as the C type `short unsigned`. When an unsigned short value must be supplied, an expression of type *number* must be supplied. When a `short unsigned` value is returned, a value of type *number* is returned.

`(sin number)` returns the sine of its argument. R⁴RS 23.

`(signal number expression)` provides a signal handler for the operating system dependent signal *number*. The *expression* is the signal handler and is either a *procedure* or a *number*. When a procedure is supplied, it is called with the signal number when the signal is present. Numeric handler values are interpreted by the underlying operating system. The previous value of the signal handler is returned.

`(sqrt number)` returns the square root of its argument. R⁴RS 23.

`(stack-size)` returns the size in bytes of Scheme->C's stack.

`stderr-port` *port* to output characters to `stderr`.

`stdin-port` *port* to input characters from `stdin`.

`stdout-port` *port* to output characters to `stdout`.

string sequence of *characters*. The valid indexes of a *string* are exact non-negative integers less than the length of the string. R⁴RS 25.

`(string char ...)` returns a newly allocated *string* whose elements contain the given arguments. R⁴RS 25.

`(string->list string)` returns a newly constructed *list* that contains the elements of *string*. R⁴RS 25.

`(string->number string)` returns a number expressed by *string*. If *string* is not a syntactically valid notation for a number then it returns `#f`. R⁴RS 24.

`(string->number string number)` returns a number expressed by *string* with *number* the default radix. Radix must be 2, 8, 10, or 16. If *string* is not a syntactically valid notation for a number then it returns `#f`. R⁴RS 24.

`(string->symbol string)` returns the interned *symbol* whose name is *string*. R⁴RS 18.

`(string->uninterned-symbol string)` returns an uninterned *symbol* whose name is *string*.

`(string-append string string ...)` returns a new *string* whose *characters* are the concatenation of the of the given *strings*. Upper and lower case letters are treated as though they were the same character. R⁴RS 26.

`(string-ci<=? string1 string2)` *predicate* that returns `#t` when *string₁* is less than or equal to *string₂*. Upper and lower case letters are treated as though they were the same character. R⁴RS 26.

`(string-ci<? string1 string2)` *predicate* that returns `#t` when *string₁* is less than *string₂*. Upper and lower case letters are treated as though they were the same character. R⁴RS 26.

`(string-ci=? string1 string2)` *predicate* that returns `#t` when *string₁* is equal to *string₂*. Upper and lower case letters are treated as though they were the same character. R⁴RS 26.

`(string-ci>=? string1 string2)` *predicate* that returns `#t` when *string₁* is greater than or equal to *string₂*. Upper and lower case letters are treated as though they were the same character. R⁴RS 26.

`(string-ci>? string1 string2)` *predicate* that returns `#t` when *string₁* is greater than *string₂*. Upper and lower case letters are treated as though they were the same character. R⁴RS 26.

`(string-copy string)` returns a new *string* whose *characters* are those of the given *string*. R⁴RS 26.

`(string-fill! string char)` stores *char* in every element of *string*. R⁴RS 26.

`(string-length string)` returns the length of *string*. R⁴RS 25.

`(string-ref string integer)` returns *character* that is the *integer* element of *string*. The first element is 0. R⁴RS 25.

`(string-set! string integer character)` sets the *integer* element of *string* to *character*. The first element is 0. R⁴RS 26.

`(string<=? string1 string2)` *predicate* that returns `#t` when *string₁* is less than or equal to *string₂*. R⁴RS 26.

(string<? *string*₁ *string*₂) *predicate* that returns #t when *string*₁ is less than *string*₂. R⁴RS 26.

(string=? *string*₁ *string*₂) *predicate* that returns #t when *string*₁ is equal to *string*₂. R⁴RS 26.

(string>=? *string*₁ *string*₂) *predicate* that returns #t when *string*₁ is greater than or equal to *string*₂. R⁴RS 26.

(string>? *string*₁ *string*₂) *predicate* that returns #t when *string*₁ is greater than *string*₂. R⁴RS 26.

(string? *expression*) *predicate* that returns #t when *expression* is a *string*. R⁴RS 25.

(substring *string* *integer*₁ *integer*₂) returns a *string* consisting of *integer*₂-*integer*₁ elements of *string* starting at element *integer*₁. R⁴RS 26.

(symbol? *expression*) *predicate* that returns #t when *expression* is a *symbol*. R⁴RS 18.

(symbol->string *symbol*) returns the name of *symbol* as a *string*. R⁴RS 18.

syntax indicates a form that is evaluated in a manner that is specific to the form. R⁴RS 5.

(system *string*) issue the shell command contained in *string* and return the result. See the man page for the `system` procedure for details.

(tan *number*) returns the tangent of its argument. R⁴RS 23.

(time-of-day) returns a system dependent *string* representing the current time and date.

(time-slice) returns the current time slice value.

(top-level) returns control to the “top-level” read-eval-print loop.

(top-level-value *symbol*) returns the value in the location that is the “top-level” binding of *symbol*.

(trace) returns a list of the procedures being traced.

(trace *symbol* *symbol* ...) enables tracing on the *procedures* that are the values of the *symbols*.

trace-output-port *port* used for trace output. The default value is the same as `stdout-port`.

(transcript-off) turns off the transcript. R⁴RS 31.

(transcript-on *string*) starts a transcript on the file *string*. R⁴RS 31.

(truncate *number*) returns the truncated value of *number*. R⁴RS 22.

tscp syntax for declaring a non-Scheme procedure, procedure argument, or global variable as the C type TSCP. The type TSCP is a tagged pointer to a Scheme object. When a *tscp* value must be supplied, any expression may be supplied. When a *tscp* value is returned, any type of value may be returned.

(unbpt) *syntax* to remove all breakpoints.

(unbpt *symbol* *symbol* ...) *syntax* to remove breakpoints from the named *procedures*.

(uninterned-symbol? *symbol*) *predicate* that returns #t if *symbol* is not *interned*.

(unless *expression*₁ *expression*₂ ...) *syntax* for a conditional form that is equivalent to (if (not *expression*₁) (begin *expression*₂ ...)).

(unquote *expression*) *syntax* to evaluate the *expression* and replaces it in the *back-quote-template*. R⁴RS 12.

(unquote-splicing *expression*) *syntax* to evaluate the *expression* and splices it into the *back-quote-template*. R⁴RS 12.

unsigned syntax for declaring a non-Scheme procedure, procedure argument, or global variable as the C type unsigned. When a *unsigned* value must be supplied, an expression of type *number* must be supplied. When a *unsigned* value is returned, a value of type *number* is returned.

(untrace) *syntax* to remove tracing from all *procedures*.

(untrace *symbol* *symbol* ...) *syntax* to remove tracing from the named *procedures*.

variable R⁴RS 6.

vector a heterogeneous mutable structure whose elements are indexed by *integers*. The valid indexes of a vector are the exact non-negative integers less than the length of the vector. R⁴RS 26.

(vector *expression* ...) returns a newly allocated *vector* whose elements contain the given arguments. R⁴RS 27.

(vector-fill! *vector expression*) stores *expression* in every element of *vector*. R⁴RS 27.

(vector-length *vector*) returns the number of elements in *vector*. R⁴RS 27.

(vector->list *vector*) returns a newly created *list* of the objects contained in the elements of the *vector*. R⁴RS 27.

(vector-ref *vector integer*) returns the contents of element *integer* of *vector*. The first element is 0. R⁴RS 27.

(vector-set! *vector integer expression*) sets element *integer* of *vector* to *expression*. The first element is 0. R⁴RS 27.

(vector? *expression*) *predicate* that returns #t when *expression* is a *vector*. R⁴RS 26.

void *syntax* for declaring a non-Scheme procedure as returning the C type `void`. The value of such a procedure may not be used.

(wait-system-file *expression*) waits for input on the file with the system file number *expression*. When input is available, the procedure returns. If *expression* is equal to #f, then the procedure will not return until all tasks have been completed.

(weak-cons *expression*₁ *expression*₂) returns a newly allocated *pair* that has *expression*₁ as its *car*, and *expression*₂ as its *cdr*. If the garbage collector discovers that pointers to an object only exist in the *car*'s of *pairs* created by `weak-cons`, then it may recover the object and set the *car*'s in those *pairs* to #f.

(when *expression*₁ *expression*₂ ...) *syntax* for a conditional form that is equivalent to (if *expression*₁ (begin *expression*₂ ...)).

(when-unreferenced *expression procedure*) applies the clean-up procedure *procedure* (with the object represented by *expression* as its argument) at some point in the future when the object represented by *expression* is no longer referenced by the program. The procedure returns either the cleanup procedure supplied by an earlier call to `when-unreferenced`, or #f when no cleanup procedure was defined.

(when-unreferenced *expression* #f) returns either the cleanup procedure for the object represented by *expression* or #f when no cleanup procedure was defined. In either case, the Scheme

system will take no action when the object represented by *expression* is no longer referenced by the program.

(with-input-from-file *string procedure*) opens the file *string*, makes its *port* the default *input-port*, then calls *procedure* with no arguments. R⁴RS 30.

(with-output-to-file *string procedure*) opens the file *string*, makes its *port* the default *output-port*, then calls *procedure* with no arguments. R⁴RS 30.

(write *expression optional-output-port*) outputs *expression* to *optional-output-port* in a machine-readable form. R⁴RS 31.

(write-char *character optional-output-port*) outputs *character* to *optional-output-port*. R⁴RS 31.

(write-circle *optional-output-port*) returns a *boolean* indicating whether circular objects are detected when output to *optional-output-port*.

(write-count *optional-output-port*) returns the number of characters on the current line in *optional-output-port*.

(write-length *optional-output-port*) returns either an *integer* indicating the maximum length vector or list printed on *optional-output-port*, or #f indicating that arbitrary length objects are printed on *optional-output-port*.

(write-level *optional-output-port*) returns either an *integer* indicating the maximum nesting depth of objects that are printed on *optional-output-port*, or #f indicating that arbitrary depth objects are printed on *optional-output-port*.

(write-pretty *optional-output-port*) returns a *boolean* indicating whether pretty-printing is done on *optional-output-port*.

(write-width *optional-output-port*) returns the width of *optional-output-port* in *characters*.

(zero? *number*) *predicate* that returns #t when *number* is zero. R⁴RS 21.

~% *format descriptor* to output a newline character.

~~ *format descriptor* to output a ~.

~A *format descriptor* to output the next *expression* using `display`.

~a format descriptor identical to *~A*.

~C format descriptor to output the next *expression* (that must be a *character*) using `write-char`.

~c format descriptor identical to *~C*.

~S format descriptor to output the next *expression* using `write`.

~s format descriptor identical to *~S*.