Scheme->C Index to the Revised³ Report on the Algorithmic Language Scheme 28 September 1990

Implementation Notes

`Scheme->C` is an implementation of the language Scheme as described in the *Revised^3 Report on the Algorithmic Language Scheme* (*SIGPLAN Notices*, V21, #12, December 1986).

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.
- 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:

- With the previously noted exceptions, the implementation conforms to the required portions of *Revised$^{3.95}$, 11 March 1989*.
- Additional procedures:

```
after-collect
bit-and        bit-lsh
bit-not        bit-or
bit-rsh        bit-xor
bpt
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-shortint-ref
c-shortint-set!
c-shortunsigned-ref
c-shortunsigned-set!
c-string->string
c-tscp-ref     c-tscp-set!
c-unsigned-ref
c-unsigned-set!
collect        collect-all
collect-rusage 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
implementation-information
my-rusage
open-file
open-input-string
open-output-string
peek-char
port->stdio-file
pp             proceed
putprop
read-eval-print
remove         remove!
remq           remq!
remv           remv!
reset
save-heap
scheme-byte-ref
scheme-byte-set!
scheme-int-ref
scheme-int-set!
scheme-tscp-ref
scheme-tscp-set!
set-top-level-value!
set-write-width!
signal
string->uninterned-symbol
top-level
top-level-value
trace          unbpt
uninterned-symbol?
untrace
when-unreferenced
write-count    write-width
```

- Additional syntax:

```
define-c-external
define-constant
define-external
define-in-line
define-macro   include
module
unless         when
```

- Additional variables:

```
*args*
*error-handler*
*obarray*      *result*
stderr-port    stdin-port
```

`stdout-port`

Index

(`-` *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. Revised^3 20.

`-C` command line flag to `scc` 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 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.

`-Ob` command line flag to `scc` 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 `scc` 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 `scc` 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 `scc` 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 `sci`. 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 `sci`. When supplied, the interpreter assumes that it is being run by GNU emacs.

`-i` command line flag to `scc` 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.

`-m` *module* command line flag to `scc` 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.

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

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

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

`-pg` command line flag to `scc` 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 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 a 4MB heap will be used. This flag will override `SCHEAP`.

`-schf` *filename* command line flag to any Scheme program to initialize the heap by loading it from the file *filename*. The minimum heap size used will be that of the Scheme program that saved the image. This flag will override `SCHEAPFILE`. Note that a heap image may only be loaded by the same program that saved it.

`-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 33. 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.

`.` denotes a dotted-pair: (*obj* `.` *obj*). Revised^3 14.

`.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. Revised^3 20.

`;` indicates the start of a comment. The comment continues until the end of the line. Revised^3 5.

(`<` *number number number* ...) *predicate* that returns `#t` when the arguments are monotonically increasing. Revised^3 19.

(`<=` *number number number* ...) *predicate* that returns `#t` when the arguments are monotonically nondecreasing. Revised^3 19.

(`=` *number number number* ...) *predicate* that returns `#t` when the arguments are equal. Revised^3 19.

`=>` used in a `cond` conditional clause. Revised^3 8.

(`>` *number number number* ...) *predicate* that returns `#t` when the arguments are monotonically decreasing. Revised^3 19.

(`>=` *number number number* ...) *predicate* that returns `#t` when the arguments are monotonically nonincreasing. Revised^3 19.

(`abs` *number*) returns the magnitude of its argument. Revised^3 20.

(`acos` *number*) returns the arccosine of its argument. Revised^3 20.

`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*. Revised^3 16.

(`and` *expression* ...) *syntax* for a conditional expression. Revised^3 9.

(`append` *list* ...) returns a list consisting of the elements of the first *list* followed by the elements of the other *lists*. Revised^3 16.

(`apply` *procedure arg-list*) calls the *procedure* with the elements of *arg-list* as the actual arguments. Revised^3 26.

(`apply` *procedure obj ... arg-list*) calls the *procedure* with the list (`append` (`list` *obj* ...) *arg-list*) as the actual arguments. Revised^3 26.

(`asin` *number*) returns the arcsine of its argument. Revised^3 20.

(`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. Revised^3 16.

(`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. Revised^3 16.

(`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. Revised^3 16.

(`atan` *number*) returns the arctangent of its argument. Revised^3 20.

(`atan` *number number*) returns the arctangent of its arguments. Revised^3 20.

back-quote-template list or vector structure that may contain `,`*expression* and `,@`*expression* forms. Revised^3 11.

(`begin` *expression* ...) *syntax* where *expression*'s are evaluated left to right and the value of the last *expression* is returned. Revised^3 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*. Revised^3 9.

(`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 be executed in sequence. Revised^3 9.

(`boolean?` *expression*) *predicate* that returns `#t` if *expression* is `#t` or `#f`. Revised^3 12.

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

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

(`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 tagged Scheme to C pointer at the *integer* byte of *c-pointer*.

(`c-tscp-set!` *c-pointer integer expression*) sets the tagged `Scheme->C` pointer 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`, `char`, `int`, `shortint`, `longint`, `unsigned`, `shortunsigned`, `longunsigned`, `float`, `double`, `tscp`, or `void`.

(`car` *pair*) returns the contents of the `car` field of the *pair*. Revised^3 15.

(`caar` *pair*) returns (`car` (`car` *pair*)). Revised^3 15.

(`ca...r` *pair*) compositions of `car` and `cdr`. Revised^3 15.

(`call-with-current-continuation` *procedure*) calls *procedure* with the current continuation as its argument. Revised^3 27.

(`call-with-input-file` *string procedure*) calls *procedure* with the *port* that is the result of opening the file *string* for input. Revised^3 28.

(`call-with-output-file` *string procedure*) calls *procedure* with the *port* that is the result of opening the file *string* for output. Revised^3 28.

(`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* ...). Revised^3 8.

(`cdr` *pair*) returns the contents of the `cdr` field of the *pair*. Revised^3 15.

(`cd...r` *pair*) compositions of `car` and `cdr`. Revised^3 15.

(`cddddr` *pair*) returns (`cdr` (`cdr` (`cdr` (`cdr` *pair*)))). Revised^3 15.

(`ceiling` *number*) returns the smallest integer that is not smaller than its arguments. Revised^3 20.

`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*. Revised^3 24.

(`char-alphabetic?` *character*) *predicate* that returns `#t` when *character* is alphabetic. Revised^3 23.

(`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. Revised^3 23.

(`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. Revised^3 23.

(`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. Revised^3 23.

(`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. Revised^3 23.

(`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. Revised^3 23.

(`char-downcase` *character*) returns the lower case value of *character*. Revised^3 24.

(`char-lower-case?` *letter*) *predicate* that returns `#t` when *letter* is lower-case. Revised^3 24.

(`char-numeric?` *character*) *predicate* that returns `#t` when *character* is numeric. Revised^3 23.

(`char-ready?` *optional-input-port*) *predicate* that returns `#t` when a character is ready on the *optional-input-port*. Revised^3 29.

(`char-upcase` *character*) returns the upper case value of the *character*. Revised^3 24.

(`char-upper-case?` *letter*) *predicate* that returns `#t` when *letter* is upper-case. Revised^3 24.

(`char-whitespace?` *character*) *predicate* that returns `#t` when *character* is a whitespace character. Revised^3 23.

(`char<=?` *character character*) *predicate* that returns `#t` when the first *character* is less than or equal to the second *character*. Revised^3 23.

(`char<?` *character character*) *predicate* that returns `#t` when the first *character* is less than the second *character*. Revised^3 23.

(`char=?` *character character*) *predicate* that returns `#t` when the first *character* is equal to the second *character*. Revised^3 23.

(`char>=?` *character character*) *predicate* that returns `#t` when the first *character* is greater than or equal to the second *character*. Revised^3 23.

(`char>?` *character character*) *predicate* that returns `#t` when the first *character* is greater than the second *character*. Revised^3 23.

(`char?` *expression*) *predicate* that returns `#t` when *expression* is a *character*. Revised^3 23.

character Scheme object that represents printed characters. See `#\`*character*, `#\`*character-name*, Revised^3 23.

(`close-input-port` *input-port*) closes the file associated with *input-port*. Revised^3 29.

(`close-output-port` *output-port*) closes the file associated with *output-port*. Revised^3 29.

(`close-port` *port*) closes the file associated with *port*. Revised^3 29.

(`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-rusage`) returns a *vector* containing information about resources consumed by the garbage collector. The information is that contained in the *rusage* structure. See *Ultrix-32 Programmer's Manual*, 2-62.

complex number complex numbers are not supported in `Scheme->C`. Revised^3 18.

(`complex?` *expression*) *predicate* that returns `#t` when *expression* is a *complex number*. All `Scheme->C` *numbers* are complex. Revised^3 19.

(`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* ...). Revised^3 8.

(`cons` $expression_1$ $expression_2$) returns a newly allocated *pair* that has $expression_1$ as its `car`, and $expression_2$ as its *cdr*. Revised^3 15.

(`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. Revised[3] 20.

(`current-input-port`) returns the current default input *port*. Revised[3] 28.

(`current-output-port`) returns the current default output *port*. Revised[3] 28.

(`define` *symbol expression*) *syntax* that defines the value of *expression* as the value of either a top-level symbol or a local variable. Revised[3] 11.

(`define` (*symbol formals*) *body*) *syntax* that defines a *procedure* that is either the value of a top-level symbol or a local variable. Revised[3] 11.

(`define` (*symbol* `.` *formal*) *body*) *syntax* that defines a *procedure* that is either the value of a top-level symbol or a local variable. Revised[3] 11.

(`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-type*$_1$...) *c-type*$_2$ *string*) *syntax* for a compiler declaration that defines *symbol* as a non-Scheme procedure with arguments of the type specified in the list *c-type*$_1$. The procedure's name is *string* and it returns a value of type *c-type*$_2$.

(`define-c-external` (*symbol* *c-type*$_1$... `.` *c-type*$_2$) *c-type*$_3$ *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-type*$_1$. Any additional arguments must be of the type *c-type*$_2$. The procedure's name is *string* and it returns a value of type *c-type*$_3$.

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

(`define-external` *symbol*$_1$ *symbol*$_2$) *syntax* for a compiler declaration that *symbol*$_1$ is defined in *module symbol*$_2$.

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

(`define-external` *symbol* *string*$_1$ *string*$_2$) *syntax* for a compiler declaration that *symbol* is in the *module string*$_1$ and has the external name *string*$_1$_*string*$_2$.

(`define-external` (*symbol*$_1$ *formals*) *symbol*$_2$) *syntax* for a compiler declaration that *symbol*$_1$ is a Scheme *procedure* defined in *module symbol*$_2$.

(`define-external` (*symbol*$_1$ `.` *formal*) *symbol*$_2$) *syntax* for a compiler declaration that *symbol*$_1$ is a Scheme *procedure* defined in *module symbol*$_2$.

(`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*) *string*$_1$ *string*$_2$) *syntax* for a compiler declaration that *symbol* is a *procedure* in the *module string*$_1$ that has the external name *string*$_1$_*string*$_2$.

(`define-external` (*symbol* `.` *formal*) *string*$_1$ *string*$_2$) *syntax* for a compiler declaration that *symbol* is a *procedure* in the *module string*$_1$ that has the external name *string*$_1$_*string*$_2$.

(`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. Revised^3 10.

(`display` *expression optional-output-port*) writes a human-readable representation of *expression* to *optional-output-port*. Revised^3 29.

(`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*). Revised^3 10.

`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. Revised^3 5.

(`eof-object?` *expression*) *predicate* that returns `#t` if *expression* is equal to the end of file object. Revised^3 29.

(`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?` $expression_1$ $expression_2$) *predicate* that is the finest test for equivalence between $expression_1$ and $expression_2$. Revised^3 14.

(`equal?` $expression_1$ $expression_2$) *predicate* that is the coarsest test for equivalence between $expression_1$ and $expression_2$. Revised^3 14.

(`eqv?` $expression_1$ $expression_2$) *predicate* that is the medium test for equivalence between $expression_1$ and $expression_2$. Revised^3 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. Revised^3 20.

exact integers are exact, all other numbers are not. Revised^3 18.

(`exact->inexact` *number*) returns the *inexact* representation of *number*. Revised^3 21.

(`exact?` *number*) *predicate* that returns `#t` if *number* is *exact*. Revised^3 19.

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

(`exp` *number*) returns exponential function of *number*. Revised^3 20.

(`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. Revised^3 6.

(`expt` $number_1$ $number_2$) returns $number_1$ raised to the power $number_2$. Revised^3 21.

`fix` *format descriptor*. Revised^3 22.

fixed `Scheme->C` internal representation of an *integer*. The maximum *fixed* value is 536,870,911 and the minimum is -536,870,912. It is represented in a 32-bit word with two bits used by the tag.

(`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*. Revised^3 20.

(`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. Revised^3 26.

(`force` *promise*) returns the forced value of a promise. Revised^3 26.

formals a *symbol* or a *list* of *symbols* that are the arguments. Revised^3 7.

(`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 (`sci` *integer*). Revised^3 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. Revised^3 20.

(`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.

(`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. Revised^3 8.

(`if` *expression*$_1$ *expression*$_2$ *expression*$_3$) *syntax* for a conditional expression. Revised^3 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. Revised^3 18.

(`inexact->exact` *number*) returns the *exact* representation of *number*. Revised^3 21.

(`inexact?` *number*) *predicate* that returns `#t` when *number* is *inexact*. Revised^3 19.

input-port Scheme object that can deliver characters on command. Revised^3 28.

(`input-port?` *expression*) *predicate* when returns `#t` when *expression* is an *input-port*. Revised^3 28.

`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. The `Scheme->C` system uses this type internally and requires that it be 32 bits long.

`int` *format descriptor*. Revised^3 22.

integer integers are represented by *fixed* values. Revised^3 18.

(`integer->char` *integer*) returns the *character* whose ASCII code is equal to *integer*. Revised^3 24.

(`integer?` *expression*) *predicate* that returns `#t` when *expression* is an *integer*. Revised^3 19.

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

(`lambda` *formals body*) the ultimate imperative, the ultimate declarative. Revised^3 7.

(`last-pair` *list*) returns the last *pair* of *list*. Revised^3 16.

(`lcm` *number* ...) returns the least common multiple of its arguments. Revised^3 20.

(`length` *list*) returns the length of *list*. Revised^3 16.

(`let` *bindings body*) *syntax* for a binding construct that computes initial values before any bindings are done. Revised^3 9.

(`let` *symbol bindings body*) *syntax* for a general looping construct. Revised^3 10.

(`let*` *bindings body*) *syntax* for a binding construct that computes initial values and performs bindings sequentially. Revised^3 9.

(`letrec` *bindings body*) *syntax* for a binding construct that binds the variables before the initial values are computed. Revised^3 9.

letter an alphabetic *character*. Revised^3 24.

list the empty list, or a *pair* whose `cdr` is a *list*. Revised^3 15.

(`list` *expression* ...) returns a *list* of its arguments. Revised^3 16.

(`list->string` *list*) returns the string formed from the *characters* in *list*. Revised^3 25.

(`list->vector` *list*) returns a *vector* whose elements are the members of *list*. Revised^3 26.

(`list-ref` *list integer*) returns the *integer* element of *list*. Elements are numbered starting at 0. Revised^3 16.

(`list-tail` *list integer*) returns the sublist of *list* obtained by omitting the first *integer* elements. Revised^3 16.

(`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. Revised^3 30.

(`loade` *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. Revised^3 30.

(`loadq` *string*) loads the expressions in the file *string* into the Scheme interpreter. The results of the expressions are not printed. Revised^3 30.

(`log` *number*) returns the natural logarithm of *number*. Revised^3 20.

`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. Revised^3 24.

(`make-string` *integer char*) returns a string of length *integer* with all elements initialized to *char*. Revised^3 24.

(`make-vector` *integer*) returns a vector of length *integer* with unknown elements. Revised^3 25.

(`make-vector` *integer expression*) returns a vector of length *integer* with all elements set to *expression*. Revised^3 25.

(`map` *procedure list list* ...) returns a *list* constructed by applying *procedure* to each element of the *lists*. The order of application is not defined. Revised^3 26.

(`max` *number number* ...) returns the maximum of its arguments. Revised^3 19.

(`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. Revised^3 16.

(`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. Revised^3 16.

(`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. Revised^3 16.

(`min` *number number* ...) returns the minimum of its arguments. Revised^3 19.

(`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 infered 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` *integer*$_1$ *integer*$_2$) returns the modulo of its arguments. The sign of the result is the sign of the divisor. Revised^3 20.

(`my-rusage`) returns a *vector* containing information about resources consumed by the program. The information is that contained in the *rusage* structure. See *Ultrix-32 Programmer's Manual*, 2-62.

(`negative?` *number*) *predicate* that returns `#t` when *number* is negative. Revised^3 19.

(`newline` *optional-output-port*) outputs a newline character on the *optional-output-port*. Revised^3 29.

(`not` *expression*) *predicate* that returns `#t` when *expression* is `#f` or `()`. Revised^3 12.

(`null?` *expression*) *predicate* that returns `#t` when *expression* is `()`. Revised^3 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. Revised^3 17.

(`number->string` *number format descriptor*) returns a *string* that is the printed representation of *number* as specified by *format descriptor*. Revised^3 21.

(`number->string` *number*) returns a string with the printed representation of the number.

(`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.

(`number?` *expression*) *predicate* that returns `#t` when *expression* is a *number*. Revised^3 19.

(`odd?` *integer*) *predicate* that returns `#t` when *integer* is odd. Revised^3 19.

(`open-file` *string*$_1$ *string*$_2$) returns a *port* for file *string*$_1$ that is opened using the Ultrix-32 *fopen* option *string*$_2$. See *Ultrix-32 Programmer's Manual*, 3-189.

(`open-input-file` *string*) returns an *input port* capable of delivering characters from the file *string*. Revised^3 28.

(`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*. Revised^3 28.

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

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. Revised^3 9.

pair record structure with two fields: car and cdr. Revised^3 14.

(`pair?` *expression*) *predicate* that returns `#t` when *expression* is a *pair*. Revised^3 15.

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

`pointer` *syntax* for declaring a non-Scheme procedure, procedure argument, or global varible 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 an non-negative *number*.

port Scheme object that is capable of delivering or accepting characters on demand. Revised^3 28.

(`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. Revised^3 19.

(`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. Revised^3 13.

(`procedure?` *expression*) *predicate* that returns `#t` when *expression* is a *procedure*. Revised^3 26.

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

(`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 *ULTRIX-32* procedure *exit*.

(`putprop` *symbol* $expression_1$ $expression_2$) stores $expression_2$ using key $expression_1$ on *symbol's* property list. See `getprop`.

(`quasiquote` *back-quote-template*) *syntax* for a *vector* or *list* constructor. Revised^3 11.

(`quote` *expression*) *syntax* whose result is *expression*. Revised^3 7.

(`quotient` $integer_1$ $integer_2$) returns the quotient of its arguments. The sign is the sign of the product of its arguments. Revised^3 20.

(`rational?` *number*) predicate that returns `#t` when its argument is a rational *number*. This is true when *number* is an *fixed* value. Revised^3 19.

(`read` *optional-input-port*) returns the next readable object from *optional-input-port*. Revived^3 29.

(`read-char` *optional-input-port*) returns the next character from *optional-input-port*, updating the *port* to point to the next *character*. Revived^3 29.

(`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*. Revised^3 19.

(`remainder` $integer_1$ $integer_2$) returns the remainder of its arguments. The sign is the sign of $integer_1$. Revised^3 20.

(`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.

(`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.

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

(`reverse` *list*) returns a new *list* with the elements of *list* in reverse order. Revised^3 16.

(`round` *number*) returns *number* rounded to the closest integer. Revised^3 20.

(`save-heap` *string* . *procedure*) saves a copy a Scheme program's heap in the file named *string*. When the heap is reloaded into a newly created process, execution will start at *procedure* that will be called with a list of the command line arguments. If *procedure* is not supplied, then execution will begin at the normal startup procedure. Note that heap image files may only be used by processes that are running the same code that was being run by the process that wrote the heap image file. N.B. Items such as ports are not automatically reinitialized.

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 begining with the letters `sc`. All procedures and external variables in these modules have names that begin with `sc..._`.

`scc` 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.

`SCHEAP` environment variable that controls the 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 a 4mb heap is used.

`SCHEAPFILE` environment variable that controls initialization of the heap from a saved heap image. If it is set to the name of a file, then the initial heap for the program will be loaded from that file.

`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 33.

(`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-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.

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

`sci` *format descriptor*. Revised^3 22.

(`set!` *symbol expression*) *syntax* to set the location bound to *symbol* to the value of *expression*. Revised^3 8.

(`set-car!` *pair expression*) sets the car field of *pair* to *expression*. Revised^3 15.

(`set-cdr!` *pair expression*) sets the cdr field of *pair* to *expression*. Revised^3 15.

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

(`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 a `short unsigned` 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. Revised^3 20.

(`signal` *number expression*) provides a signal handler for the *ULTRIX-32* 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. Revised^3 20.

`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*. Revised^3 24.

(`string->list` *string*) returns a newly constructed *list* that contains the elements of *string*. Revised^3 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`.

(`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`.

(`string->symbol` *string*) returns the interned *symbol* whose name is *string*. Revised^3 17.

(`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. Revised^3 25.

(`string-ci<=?` *string*$_1$ *string*$_2$) *predicate* that returns `#t` when *string*$_1$ is less than or equal to *string*$_2$. Upper and lower case letters are treated as though they were the same character. Revised^3 25.

(`string-ci<?` *string*$_1$ *string*$_2$) *predicate* that returns `#t` when *string*$_1$ is less than *string*$_2$. Upper and lower case letters are treated as though they were the same character. Revised^3 25.

(`string-ci=?` *string*$_1$ *string*$_2$) *predicate* that returns `#t` when *string*$_1$ is equal to *string*$_2$. Upper and lower case letters are treated as though they were the same character. Revised^3 25.

(`string-ci>=?` *string*$_1$ *string*$_2$) *predicate* that returns `#t` when *string*$_1$ is greater than or equal to *string*$_2$. Upper and lower case letters are treated as though they were the same character. Revised^3 25.

(`string-ci>?` *string*$_1$ *string*$_2$) *predicate* that returns `#t` when *string*$_1$ is greater than *string*$_2$. Upper and lower case letters are treated as though they were the same character. Revised^3 25.

(`string-copy` *string*) returns a new *string* whose *characters* are those of the given *string*. Revised^3 25.

(`string-fill!` *string char*) stores *char* in every element of *string*. Revised^3 25.

(`string-length` *string*) returns the length of *string*. Revised^3 24.

(`string-ref` *string integer*) returns *character* that is the *integer* element of *string*. The first element is 0. Revised^3 24.

(`string-set!` *string integer character*) sets the *integer* element of *string* to *character*. The first element is 0. Revised^3 24.

(`string<=?` *string*$_1$ *string*$_2$) *predicate* that returns `#t` when *string*$_1$ is less than or equal to *string*$_2$. Revised^3

25.

(`string<?` *string*$_1$ *string*$_2$) *predicate* that returns `#t` when *string*$_1$ is less than *string*$_2$. Revised[3] 25.

(`string=?` *string*$_1$ *string*$_2$) *predicate* that returns `#t` when *string*$_1$ is equal to *string*$_2$. Revised[3] 25.

(`string>=?` *string*$_1$ *string*$_2$) *predicate* that returns `#t` when *string*$_1$ is greater than or equal to *string*$_2$. Revised[3] 25.

(`string>?` *string*$_1$ *string*$_2$) *predicate* that returns `#t` when *string*$_1$ is greater than *string*$_2$. Revised[3] 25.

(`string?` *expression*) *predicate* that returns `#t` when *expression* is a *string*. Revised[3] 24.

(`substring` *string integer*$_1$ *integer*$_2$) returns a *string* consisting of *integer*$_2$-*integer*$_1$ elements of *string* starting at element *integer*$_1$. Revised[3] 25.

(`symbol->string` *symbol*) returns the name of *symbol* as a *string*. Revised[3] 17.

(`symbol?` *expression*) *predicate* that returns `#t` when *expression* is a *symbol*. Revised[3] 17.

syntax indicates a form that is evaluated in a manner that is specific to the form. Revised[3] 1.

(`tan` *number*) returns the tangent of its argument. Revised[3] 20.

(`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*.

(`transcript-off`) turns off the transcript. Revised[3] 30.

(`transcript-on` *string*) starts a transcript on the file *string*. Revised[3] 30.

(`truncate` *number*) returns the truncated value of *number*. Revised[3] 20.

`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*. Revised[3] 11.

(`unquote-splicing` *expression*) *syntax* to evaluate the expression and splices it into the *back-quote-template*. Revised[3] 11.

`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*.

vector a heterogenous mutable structure whose elements are indexed by *integers*. Revised^3 25.

(`vector` *expression* ...) returns a newly allocated *vector* whose elements contain the given arguments. Revised^3 25.

(`vector-fill!` *vector expression*) stores *expression* in every element of *vector*. Revised^3 26.

(`vector->list` *vector*) returns a newly created *list* of the objects contained in the elements of the *vector*. Revised^3 26.

(`vector-length` *vector*) returns the number of elements in *vector*. Revised^3 25.

(`vector-ref` *vector integer*) returns the contents of element *integer* of *vector*. The first element is 0. Revised^3 25.

(`vector-set!` *vector integer expression*) sets element *integer* of *vector* to *expression*. The first element is 0. Revised^3 25.

(`vector?` *expression*) *predicate* that returns `#t` when *expression* is a *vector*. Revised^3 25.

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

(`when` *expression*$_1$ *expression*$_2$ ...) *syntax* for a conditional form that is equivalent to (`if` *expression*$_1$ (`begin` *expression*$_2$ ...)).

(`when-unreferenced` *expression procedure*) applys 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. Revised^3 28.

(`with-output-to-file` *string procedure*) opens the file *string*, makes its *port* the default *output-port*, then calls *procedure* with no arguments. Revised^3 28.

(`write` *expression optional-output-port*) outputs *expression* to *optional-output-port* in a machine-readable form. Revised^3 29.

(`write-char` *character optional-output-port*) outputs *character* to *optional-output-port*. Revised^3 29.

(`write-count` *optional-output-port*) returns the number of characters on the current line in *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. Revised^3 19.

`~%` *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`.