Debugging And Testing

These routines provide assertions, warnings, errors, and other debugging facilities one's used to having in other languages. See UnitTesting for discussion about unit testing facilities.

Asserts, warnings and debugging messages

Some of these functions assume the existence of a global `line` that gives the current line number of the file being processed, if that makes any sense.

See also `fast_assert` in OptimisationCodingTips.

```
-- Give warning with, optionally, the name of program and file
--   s: warning string
function warn(s)
  if prog.name then write(_STDERR, prog.name .. ": ") end
  if file then write(_STDERR, file .. ": ") end
  writeLine(_STDERR, s)
end

-- Die with error
--   s: error string
function die(s)
  warn(s)
  error()
end

-- Die with line number
--   s: error string
function dieLine(s)
  die(s .. " at line " .. line)
end

-- Die with error if value is nil
--   v: value
--   s: error string
function affirm(v, s)
  if not v then die(s) end
end
```

```
-- Die with error and line number if value is nil
--   v: value
--   s: error string
function affirmLine(v, s)
  if not v then dieLine(s) end
end

-- Print a debugging message
--   s: debugging message
function debug(s)
  if _DEBUG then writeLine(_STDERR, s) end
end
```

Debugging utilties

Also useful for general interactive use: extensions of `tostring` and hence `print` to show tables better.

```
-- Extend tostring to work better on tables
-- make it output in {a,b,c...;x1=y1,x2=y2...} format; use nexti
-- only output the LH part if there is a table.n and members 1..n
--   x: object to convert to string
-- returns
--   s: string representation
function tostring(x)
  local s
  if type(x) == "table" then
    s = "{"
    local i, v = next(x)
    while i do
      s = s .. tostring(i) .. "=" .. tostring(v)
      i, v = next(x, i)
      if i then s = s .. "," end
    end
    return s .. "}"
  else return %tostring(x)
  end
end

-- Extend print to work better on tables
--   arg: objects to print
function print(...)
  for i = 1, getn(arg) do arg[i] = tostring(arg[i]) end
```

```
  call(%print, arg)
end
```

Here are `Get/DiffGlobalNames` which can be used to track down accidental global assignments:

```
-- GetGlobalNames - returns hash table of current
--     global names
--
function GetGlobalNames()
  local names = {}
  for i,x in globals() do
    names[i] = 1
  end
  return names
end

-- DiffGlobalNames - shows diff of current global names
--     vs previously recorded
--
function DiffGlobalNames(t)
  local gtable = globals()
  local deleted = {}

  local added = {}
  for i,x in t do
    if not gtable[i] then
      tinsert(deleted, i)
    end
  end
  for i,x in gtable do
    if not t[i] then
      tinsert(added, i)
    end
  end
  sort(deleted)
  sort(added)
  print("Changes to global names:")
  print("    Deleted")
  for i = 1, getn(deleted) do
    print("        "..deleted[i])
  end
  print("    Added")
  for i = 1, getn(added) do
    print("        "..added[i])
```

```
    end
  end
```

FindPage · RecentChanges · preferences
edit · history
Last edited June 18, 2003 1:19 am PDT (diff)