KR wrote:

> My first issue is that the following code:

>

> local ffi = require("ffi")

> local p = ffi.gc(ffi.C.malloc(10), ffi.C.free)

>

> results in the following error (on WinXP, visual studio 2008, git HEAD built

> with msvcbuild.bat): "missing declaration for symbol 'malloc'".

Well, I thought the error message was pretty unambiguous. :-)

You have to declare C functions before you use them:

ffi.cdef[[

void \*malloc(size\_t size);

void free(void \*ptr);

]]

> Moreover, in a previous post of yours I noticed you posted the following code:

> return ffi.gc(ffi.new("void \*[10]"), delarray)

>

> I thought that all values returned from ffi.new were always garbage collected,

> does ffi.new being inside of a ffi.gc call change this behaviour?

The array itself is a garbage-collected object and nothing can

change that. The point was to show how to free the 10 pointers

\_inside\_ the array, too (the GC doesn't know anything about them).

> -- Constructor for vector

> function Vec(size)

> -- Should I use ffi.new here instead?

> local n = ffi.sizeof("double")\*size

> local ptr = ffi.gc(ffi.C.malloc(n), ffi.C.free)

> return vec\_ctor(ptr, n)

> end

>

> Will the m\_ptr inside of the vec\_t keep the data of the vector

> alive (until the value returned from Vec is alive) or will

> ffi.C.free be called at any time after the execution of the Vec

> function is terminated?

The latter. Pointers are not followed by the GC. Only references

from local variables, upvalues, Lua tables etc. are considered.

This also means you must not store the only reference to an object

allocated with ffi.new() as a pointer in a struct field.

In general, you need to attach the GC finalizer to the garbage

collected cdata object that's returned to the user. In your use

case this is the vector struct and not the pointer inside:

local vec\_mt = {

\_\_len = function(x) return x.m\_size end,

\_\_index = function(x, i) return x.m\_ptr[i] end,

\_\_newindex = function(x, i, v) x.m\_ptr[i] = v end,

-- Free array before struct is deallocated by the GC.

\_\_gc = function(x) ffi.C.free(x.m\_ptr) end,

}

local vec\_ctor = ffi.metatype("vec\_t", vec\_mt)

function Vec(size)

-- Allocate array + struct.

return vec\_ctor(ffi.C.malloc(size\*8), size)

end

--Mike