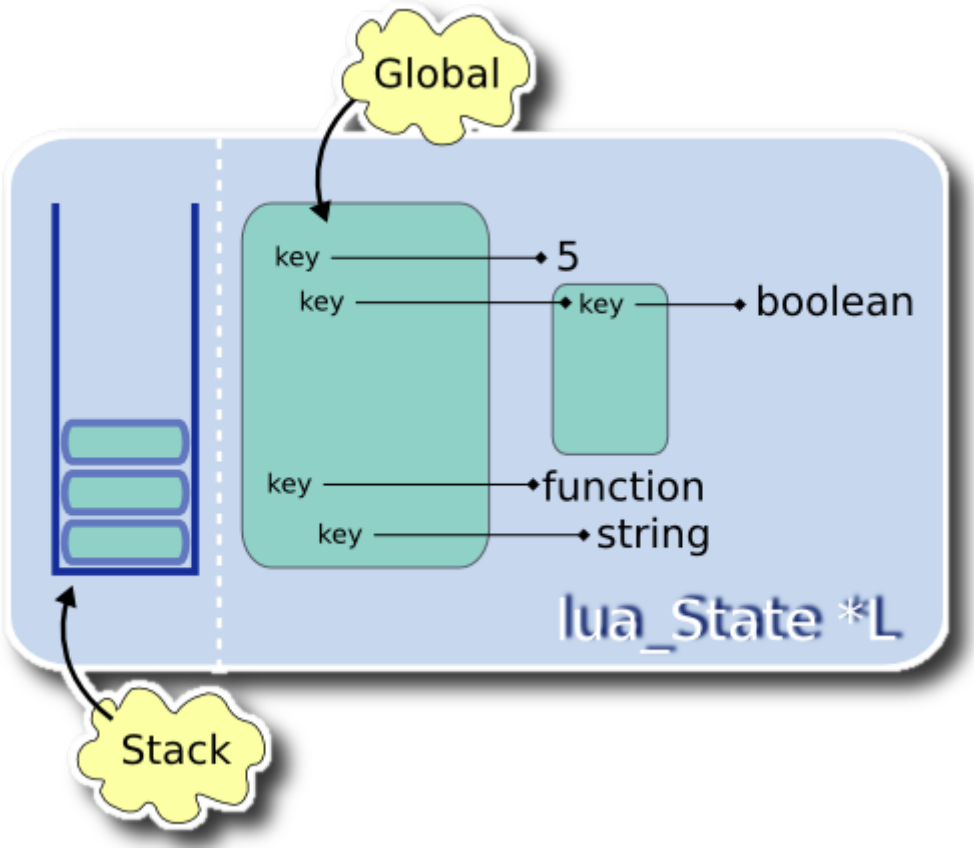
Luanatic with features

PpluX 's blog

lua API, introducción

Haciendo honor al nombre del blog, y aprovechando que recientemente me han comentado que el API de lua es un poco rara, vamos a hincarle el diente directamente al problema. Este post es sólo para programadores, no trata del léxico/sintaxis de lua, sólo de una parte muy particular del API, concretamente, la que más problema da al programador que se está iniciando en esto de lua.

Veamos en primer lugar que es eso de un **lua_State***, es fácil crearlo (luaL_newstate), y destruirlo (lua_close) y representa un estado completo de lua. A efectos prácticos es como si con cada lua_State fuera una máquina virtual independiente, por lo que podemos tener tantos como queramos (uno por thread, uno por efecto, uno por agente, etc, etc...).



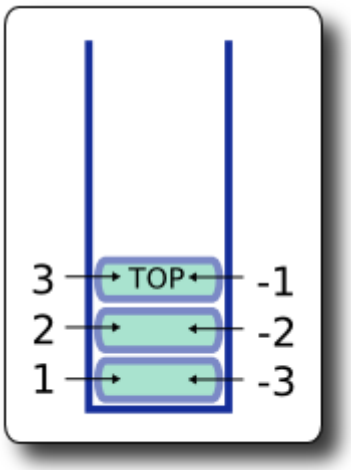
En la imagen vemos que un estado de lua ofrece básicamente un **stack**(pila) para trabajar con el estado. Esto es lo más complicado de entender del API, pero una vez se ve en el contexto, es una forma super eficiente de trabajar. No voy a explicar hoy qué razón se oculta tras la pila, asumamos que es así y que hay que aprender a usarla.

También está representado en la imagen los tipos básicos de lua: strings, números, booleanos, funciones, tablas... y poco más. La estructura clave aquí es la tabla que es el

único contenedor que tiene lua y se trata de un map de pares clave-valor. La clave puede ser cualquier tipo de lua y el valor, por supuesto, también.

Las funciones en lua también son tipos de primer orden, esto quiere decir que las funciones son un valor más que se puede copiar, asignar, devolver como resultado de otra función, etc. Como hemos dicho antes, las tablas incluso pueden usar funciones como claves, nada lo impide.

La linea de puntos de la imagen que separa el stack de la tabla global es para resaltar que no accedemos directamente a la tabla de valores globales. Para poder manejar la tabla de valores globales, u otra tabla, usaremos operaciones que apilarán o consumirán valores del stack. En resumidas cuentas **siempre trabajamos con el stack**



El stack se accede por índice, en lua los índices numéricos empiezan siempre en 1, en contraposición con lo típico en C/C++ de empezar todo en 0. También hay razones tras este acto de maldad, pero como somos programadores serios y profesionales, esto no es más que un detalle, y nos da igual.

Los índices pueden ser positivos, o negativos. Si son positivos contamos desde la base del stack y si son negativos desde el *top* del stack. Para saber el top actual, usamos lua_gettop(L). La mayor parte de las funciones, por no decir todas ellas, utilizan los valores cercanos al top... y lo divertido de esto, si has cursado alguna asignatura de compiladores, es que se parece mucho a la forma de trabajar en ensamblador para llamar a funciones 🙂

Bueno, para ilustrar todo lo anterior, y explicar con detalle un ejemplo del manual, vamos a ver paso a paso la ejecución del código equivalente a esta expresión de lua:

```
a = f("how", t.x, 14)
```

según el manual se traduce en las siguientes instrucciones del API de lua:

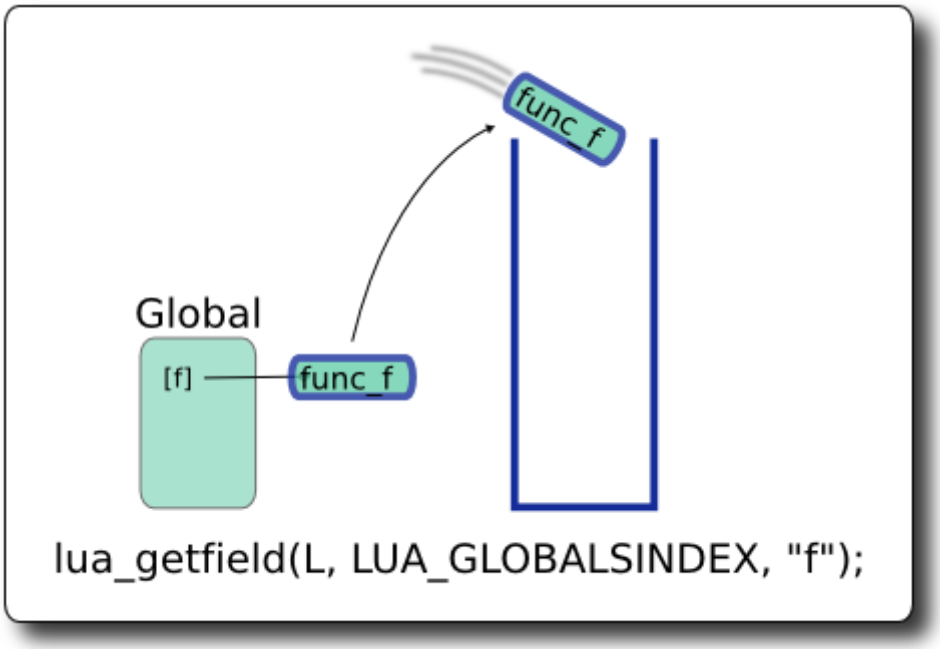
```
lua_getfield(L, LUA_GLOBALSINDEX, "f"); /* function to be called */
lua_pushstring(L, "how"); /* 1st argument */
lua_getfield(L, LUA_GLOBALSINDEX, "t"); /* table to be indexed */
lua_getfield(L, -1, "x"); /* push result of t.x (2nd arg) */
lua_remove(L, -2); /* remove 't' from the stack */
lua_pushinteger(L, 14); /* 3rd argument */
lua_call(L, 3, 1); /* call 'f' with 3 arguments and 1 result */
lua_setfield(L, LUA_GLOBALSINDEX, "a"); /* set global 'a' */
```

Las instrucciones del API del manual van acompañadas de una etiqueta de la forma [**-o**, **+p**, x] :

- **-o**: número de elementos que consume del stack (pops from the stack)
- **+p**: número de elementos que apila en el stack
- x: tipos de errores que pueden saltar... esto es para otro día (así que como si no estuviera)

Es fácil deducirlo pero las funciones de lua pueden devolver varios elementos, y por supuesto, consumir otros tantos. Las etiquetas vienen bien para saber de un vistazo de qué forma van a operar con el stack.

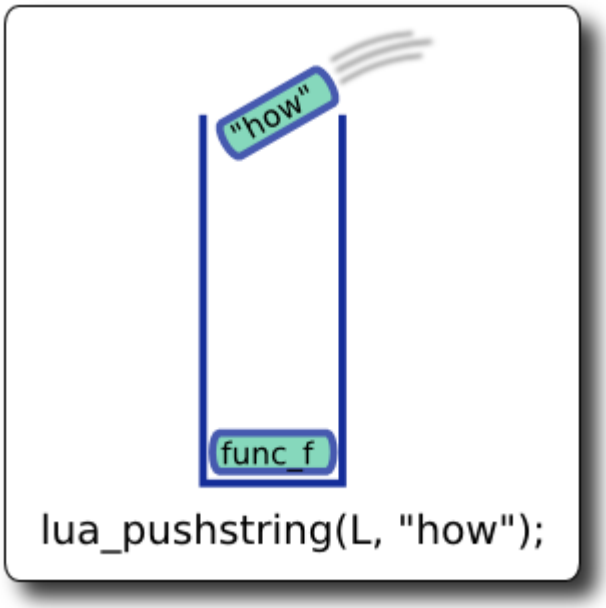
Volviendo al ejemplo anterior, supongamos que ahora mismo el stack está vacío, y veamos paso a paso cada una de las instrucciones.



void lua_getfield (lua_State *L, int index, const char *k) [-0, +1, e]

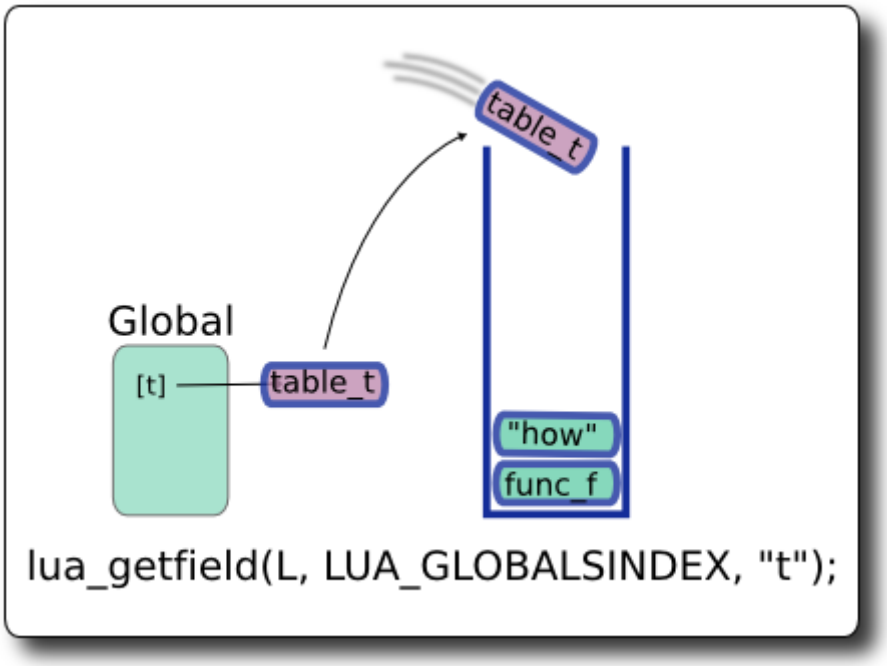
Por la etiqueta sabemos que esta función no va a consumir nada del stack y siempre va a hacer push de un elemento. Esta función concretamente busca el elemento *key* de la tabla que está en el índice *index* y lo devuelve en el stack. Si el elemento no existe hará un push de "nil" que es otro tipo de datos, usado para indicar, precisamente, la ausencia de tipo de datos 😀

LUA_GLOBALSINDEX es un pseudo-índice, en el stack no hay ninguna posición que permita acceder a la tabla de valores globales, así que hay una serie de pseudo-índices para acceder a ciertas tablas especiales (como la de los valores globales en concreto). Puedes imaginar ese índice de la forma que más te apetezca, a efectos prácticos es equivalente a poner un 1,2,... -1,-128, etc... en unos pasos veremos un getfield sobre una tabla en el stack para compensar.

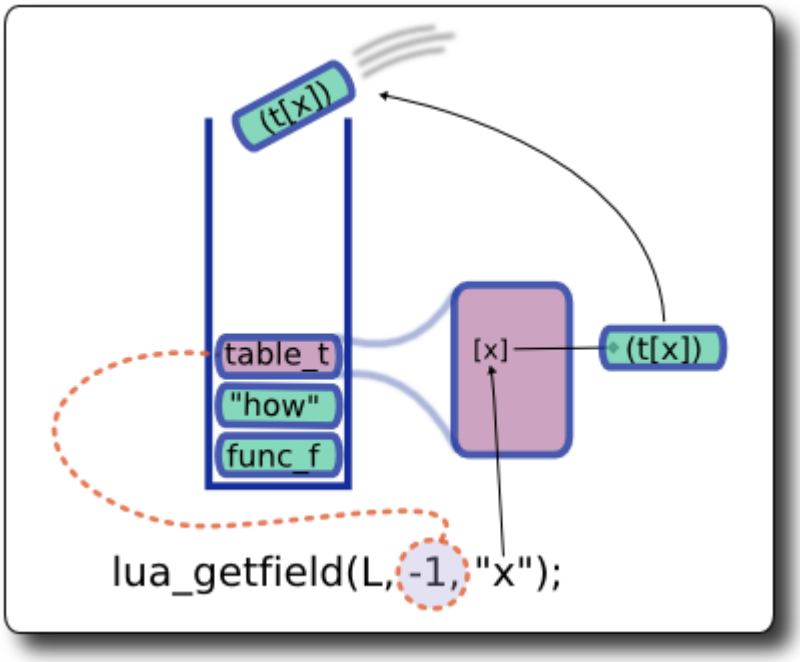


void lua_pushstring (lua_State *L, const char *s) [-0, +1, m]

Lua tiene unas cuantas funciones para meter y sacar elementos del stack, esta es una de tantas, y sirve para apilar un string. [Nota mental: *hablar de los strings de lua en otro post...*]

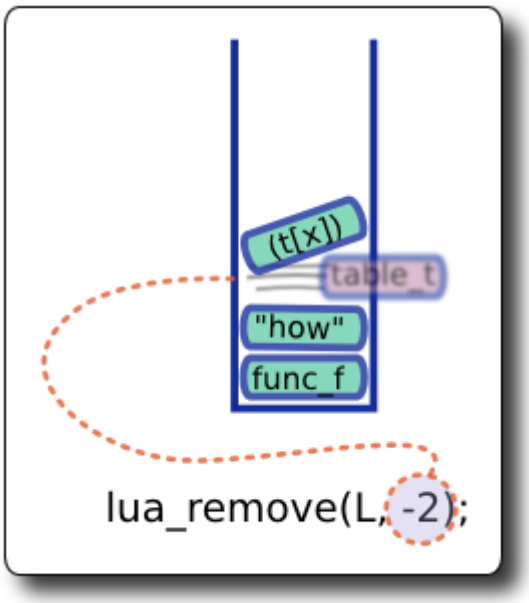


Otro acceso a la tabla global, en este caso tras el *key* "t" se esconde una tabla, por lo que se apila una tabla. Lo he dibujado con otro color como alarde de creatividad, pero no es más que otro valor [booleano, string, funcion, tabla, numero....] metido en la tabla.



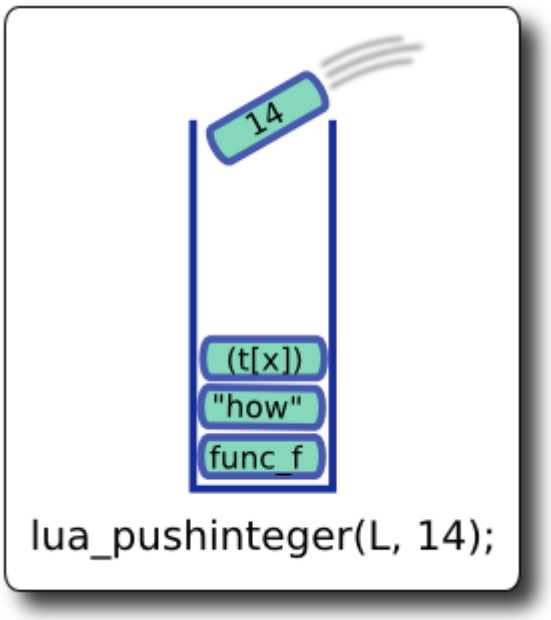
Esta es más interesante, aunque la función es conocida. Se trata de un getfield de un elemento en la propia tabla, por lo que usamos un índice numérico para indicar de qué tabla queremos buscar la clave. Como el índice es **-1**, estamos haciendo referencia al top actual del stack, donde *voilà* está justo la tabla que acabábamos de apilar.

> ***¿Podíamos haber usado 3 cómo índice para acceder a la tabla?***
> sí, … pero mejor no te acostumbres. El "3" depende de todas las acciones que hayamos hecho antes, mientras que el -1 depende sólo de las últimas acciones sobre la pila. Así que… en general, es mejor usar índices negativos para este tipo de acciones puntuales.
>
> En este caso queremos acceder al elemento "t.x" para hacer ejecutar "a = f("how" , t.x, 14)" , si por ejemplo fuese "a = f(1,2,3," how" , t.x, 14) ya no podríamos usar el índice 3 (usaríamos 6, al haber 3 elementos más en la pila), pero sí que podríamos seguir usando el -1.



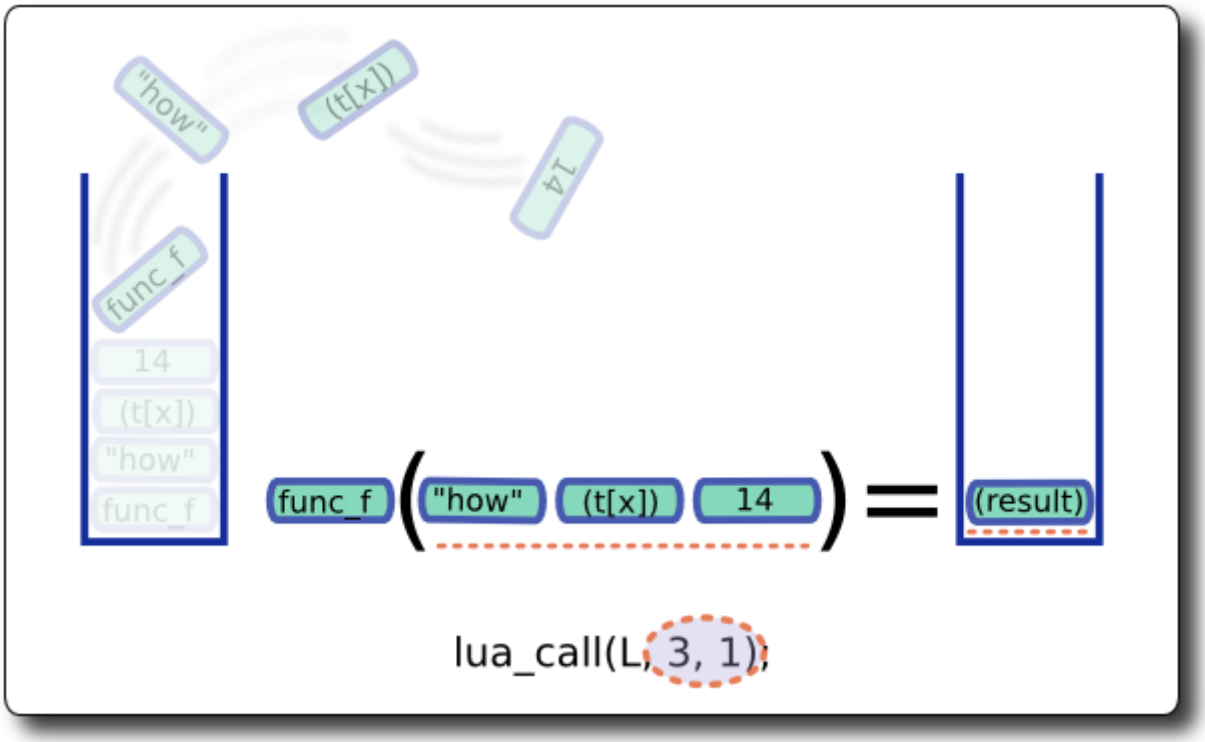
> void lua_remove (lua_State *L, int index); [-1, +0, -]

Esta función, si miras la etiqueta, elimina un elemento del stack y no añade nada. Hay poco que explicar aquí, nos cargamos lo que esté en la posición apuntada por *index*. Al margen de la función, hemos conseguido obtener el elemento "t.x" para la llamada "a = f("how" ,**t.x**,14)" (bieeeen!)



void lua_pushinteger (lua_State *L, lua_Integer n) [-0, +1, m]

... No hace falta dar detalles, ¿no? otro "lua_pushxxxx"

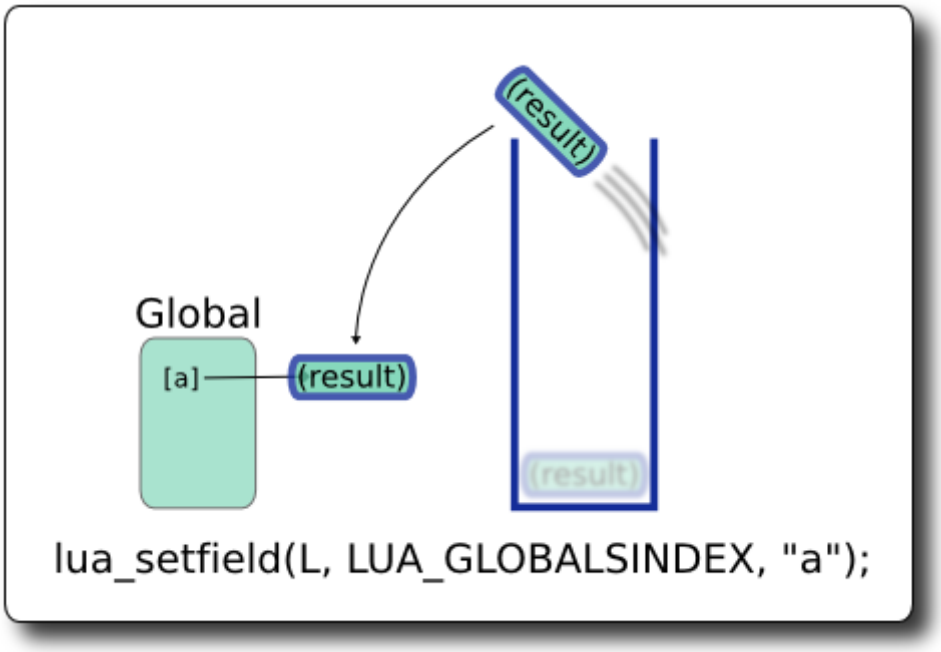


void lua_call (lua_State *L, int nargs, int nresults); [-(nargs + 1), +nresults, e]

Ahora vamos a realizar una "llamada a función" (un *call*) de *nargs* argumentos de entrada y esperando obtener *nresults* elementos de salida. Si miráis la etiqueta de la función pone que se consumen del stack nargs+1 elementos: n argumentos + 1 función.

El órden es el que podéis ver en el stack, primero la función apilada, después cada argumento en orden y llamamos a lua_call. Con esto ejecutamos la función tal y como se ve en la figura.

No sabemos el tipo del resultado devuelto, pero por la forma de llamar a call sabemos que siempre tendremos un elemento en la pila. Aquí lua realiza un proceso de ajuste: si la función devolvió 600 elementos, al poner nosotros lua_call(..., 1); se queda con el primero y descarta los otros 599, si la función no devolvía nada (0 elementos) y nosotros queríamos 1, rellenará el espacio con nil' s , finalmente si queremos que la función devuelva todos los elementos que quiera, en vez de poner un numero en *nresults*, usamos la constante LUA_MULTRET (y con ello se evita el proceso de ajuste).



void lua_setfield (lua_State *L, int index, const char *k); [-1, +0, e]

Para acabar, asignamos el resultado de la operación a la variable global "a" , usando lua_setfield (análogo a lua_getfield). La etiqueta nos dice que va a consumir un elemento y que no va a apilar nada. Con esto tenemos ejecutada toda la secuencia de " a = f("how" , t.x, 14)" .

Nota:el call lo hicimos con lua_call(...,1) precisamente porque la intención era guardar el resultado en **una** variable (gracias al ajuste nos da igual lo que haya devuelto la función).

Y hasta aquí el primer tutorial sobre el API de lua... para las dudas y aclaraciones, ahí tenéis los comentarios 😀

Skip to comment form »

1. **Rubén Penalva** said on abril 16, 2008 at 10:27 am

   Muy interesante! Seguramente, dentro de relativamente poco, voy a tener que lidiar para embeberlo y aunque ya tengo bastante experiencia extendiendo Python, soy totalmente novato con Lua. Esto me va a ayudar bastante. Gracias!

   Por cierto, ¿que programa usas para hacer los diagramas?

2. **PpluX** said on abril 16, 2008 at 10:40 am

   Si necesitas algún tipo de ayuda, o comentario, sobre lo de lua no tienes más que escribir un correo y estaré encantado de echar una mano 😉

   Para los diagramas uso inkscape, y aunque soy un pésimo "artista" es realmente potente en manos de quien sabe usarlo. Y es soft libre!

3. **sole** said on abril 18, 2008 at 10:39 am

   Esto es el momento "AAAAAAAHAAAAAAAAAAA!!!"

   Ya lo entiendo!! 🙂

   No se como no se me ocurrio pensar en los pops y pushes de ASM (bueno, si, porque creo que he escrito 3 lineas de ASM en mi vida y de eso hace eones :D)

   Bueno pues ahora voy a seguir estudiando el manual, ahora que ya he subido el primer escalon que conduce al lua-zen.

   Gracias!

4. **PpluX** said on abril 18, 2008 at 10:52 am

   Felicidades! Si esto fuera "Finding Nemo" acabas de pasar el volcán de burbujas 🙂

   Superada la primera prueba, el resto es más sencillo. Respecto al API, mi recomendación para todos los que quieran iniciarse es echarle un ojo a la versión 5.0, es más fácil de leer que la del 5.1, con diferencia. Pero sin duda lo mejor es leerlo con todo detalle en el libro, que aunque sea para la versión 5.0 los conceptos básicos se mantienen.

   Y por supuesto, para cualquier duda/problema, un mail 🙂

5. **Escena.org invtro v2 - soledad penadés** said on mayo 16, 2008 at 11:30 am

   [...] course big thanks to Pplux for his awesome explanation about how the stack in Lua works. That was all I needed to take [...]

« disléxicos del mundo unisors!!!
Enjuto también usa ubuntu! »

Entry Details

You' re currently reading "lua API, introducción," an entry on Luanatic with features

Published:
abril 16, 2008 around 10am

Category:
C/C++, Lua, Programación

Comments:

5 comments so far

Light Reading

Blogroll

- biestado
- Blog autoestereoscópico
- cAjóN DeSaStRe
- Hypertension’ s code
- I see dead pixels
- In Web We Trust
- Mushrooms & Cookies
- Nighty build
- Pluton Tech
- Soledad Penadés

Enlaces

- DreamHost

Latest Posts

- 05.11 Welcome to Codepixel’ s planet
- 02.18 The Safe bool idiom.
- 02.07 rsync, root and sudo
- 11.26 from subversion to git
- 11.24 The return to the dark side
- 11.02 switching from Spanish to English
- 09.10 Vuelta al cole!
- 05.30 Real como la vida misma (II)
- 05.13 gpu gems 2: online !
- 04.29 ¿Qué sistema de ficheros usas?