

1.1 LUA

Linguaggio includibile

Lua¹ nasce dalla fusione di un linguaggio per la descrizione di dati DEL (*Data Entry Language*) e di SOL (Simple Object Language), un linguaggio per descrivere e trattare oggetti. Lua è stato progettato e realizzato presso il Computer Graphics Technology Group dell'Università Cattolica di Rio de Janeiro da Roberto Ierusalimschy, Waldemar Celes e Luiz Henrique nel 1993/94.

Il risultato è un linguaggio di programmazione adatto ad essere incluso in applicazioni, ma anche utilizzabile per scopi più generali. Lua ha sei tipi di date basilari: nil, numeri, stringhe, funzioni, tabelle e dati definiti dall'utilizzatore (*userdata*).

Le tabelle sono una struttura essenziale di LUA; sono di tipo associativo, vale a dire gli elementi sono individuati da una chiave, tuttavia, possono essere utilizzate anche contemporaneamente come usuali matrici indicizzate, di fatto `tabella[1]` e `tabella["1"]`, sono due oggetti diversi. Le tabelle possono contenere metodi (puntatori a funzioni), ciò dà a LUA alcune caratteristiche di un linguaggio ad oggetti:

```
-- ... indica una lista di valori. LUA li fornisce alla funzione
-- sotto forma di tabella di nome arg, il cui elemento di nome "n"
-- (arg.n o arg["n"]) contiene il numero degli elementi della lista
function mediarmonica(...)
    media = 0
    for i = 1, arg.n do
        if arg[i] ~= 0 then media = media + 1/arg[i] end
    end
    return media/arg["n"]
end
Tab = {}
Tab["mediar"] = mediarmonica
```

Le funzioni possono restituire più valori:

```
function medie(...)
    media = 0
    for i = 1, arg.n do media = media + arg[i] end
    return media/arg.n, call(Tab.mediar, arg) -- per poter passare la
lista
end
numeri = {7,14,21,28}
m, ma = call(medie,numeri) -- ritorna 2 valori
write(_OUTPUT, "\nCalcolo di medie del vettore\n")
for indx, val in numeri do write(_OUTPUT, format("%3d ", val)) end
write(_OUTPUT, format("\nmedia: %5.2f  media armonica: %5.3f
\n", m, ma))
```

L'esecuzione del programma produce:

```
Calcolo di medie del vettore
 7  14  21  28
media: 17.50  media armonica: 0.074
```

LUA può essere incluso in applicazioni per dotarle di programmabilità, grazie alle molte funzioni d'interfacciamento di cui dispone, funzioni che rendono possibile l'estensione del linguaggio attraverso i tipi di dato *userdata* di fatto oggetti dell'applicazione ospite.

L'ambito delle variabili è globale, salvo che non sia esplicitamente dichiarata locale o sia in un blocco `do ... end` o in una dichiarazione di funzione. Ha delle stranezze sintattiche quali: `x = x or v` equivalente a `if x == nil then x = v end`, oppure `x = a and b or c` equivalente a: `if a then x = b else x = c end`.

Rispetto a linguaggi simili, ha poche librerie, ma la possibilità di richiamare procedure esterne non ne fa un grosso limite. Un limite, specialmente per un linguaggio includibile, è la mancanza

¹ Lua è in portoghese *luna*.

di alcune funzionalità su stringhe quali la separazione in token o l'eliminazione degli spazi in testa o in coda ad una stringa. Come molti interpreti permette l'esecuzione di istruzioni "costruite" tramite le istruzioni `dostring()` e `dofile()`.

E' stato utilizzato per sviluppare giochi (Grim Fandango).

L'esempio che segue è un interprete (limitato) del linguaggio PILOT (v.**Errore. L'origine riferimento non è stata trovata.**):

```
-- Lua 4.0 Copyright (C) 1994-2000 TeCGraf, PUC-Rio
interp = function(cmd,cond,arg)
  if cond ~= '' then
    if cond == "Y" and mtch == nil then return end
    if cond == "N" and mtch == 1 then return end
    if strfind("YN",cond) == nil then
      dostring("mtch = " .. cond) -- concateno stringhe
      if mtch == nil then return end
    end
  end
  if cmd == "T" then print(arg)
  elseif cmd == "CH" then execute("cls")
  elseif cmd == "A" then
    risposta = read(_INPUT,"*l")          -- da tastiera
    if arg ~= '' then
      if strsub(arg,1,1) == '#' then      -- numeri PILOT
        if tonumber(risposta) == nil then risposta = "0" end
        dostring(strsub(arg,2) .. " = tonumber(0" .. risposta
        ..)")")
      elseif strsub(arg,1,1) == '$' then  -- stringhe PILOT
        dostring(strsub(arg,2) .. " = '" .. risposta .."'")
      end
    end
  elseif cmd == "M" then mtch = (strfind(arg,risposta))
  elseif cmd == "J" or cmd == "U" then go(gsub(arg,' ',''),cmd)
  elseif cmd == "C" then dostring(arg)   -- Compute
  elseif cmd == "E" then ret()
  else
    end
  return
end

go = function(arg,tipo) -- gosub ("U") o jump ("J")
  posiz = label[arg]
  if posiz == nil then
    print(arg,"label sconosciuta")
    exit(11)
  end
  if tipo == "U" then tinsert(stack,seek(flh)) end -- mem. Ind. di
ritorno
  seek(flh,"set",posiz)
end

ret = function() -- ritorno da subroutine
  if getn(stack) == 0 then exit(0) end -- presumo fine programma
  seek(flh,"set",tremove(stack))
end

-- inizio interprete
pgm = "pilot.hlp"
if arg["n"] > 0 then pgm = arg[1] end
memcmd = "R" -- commento
label = {} -- inizializzo tabella (asociativa) delle label
stack = {} -- inizializzo stack indirizzi di ritorno
```

```

flh = openfile (pgm,"r")
while not nil do
  row = read(flh,"*1")
  if row == nil then break end
  if strsub(row,1,1) == "*" then
    label[gsub(row,' ','')] = seek(flh) -- label con posizione nel
file
  end
end
seek(flh,"set") -- all'inizio del programma
while not nil do
  row = read(flh,"*1")
  if row == nil then break end
  k = strfind(row,":")
  if k ~= nil then
    cmd = strsub(row,1,k-1)
    while strsub(cmd,-1) == ' ' do cmd = strsub(cmd,1,-2) end
    cond = ''
    if strsub(row,2,2) == "(" then
      cond = gsub(strsub(row,2,k-1),"<>","~=") -- cambio <> in
diverso LUA ~=
      cmd = strupper(strsub(row,1,1))
    elseif strfind("YyNn",strsub(cmd,-1)) ~= nil then
      cond = strupper(strsub(cmd,-1))
      cmd = strupper(strsub(cmd,1,-2)) -- elimino cond
    end
    arg = strsub(row,k+1)
    while strsub(arg,-1) == ' ' do arg = strsub(arg,1,-2) end
    if cmd ~= '' then memcmd = cmd end
    interp(memcmd,cond,arg)
  end
end
end

```

1.1.1 Lua 5

Lua 5.1.4 è stato rilasciato il 22 agosto, 2008.

La caratteristica principali è che la maggior parte delle funzioni di libreria ora sono definiti all'interno di tabelle, di fatto il loro utilizzo sintattico è simile a quello dei metodi degli oggetti; ciò vale in particolare per `string`, `math` e `io` (input output), `coroutine` (multithreading collaborativo); tuttavia lo script (`compat.lua`) può essere usato per mantenere la compatibilità con le versioni precedenti.

Un'altra, significativa, modifica è nel richiamo delle funzioni, che avviene nella forma `funzione(argumenti)` invece di `call(funzione,argumenti)`.

In questo esempio di funzione Lua (eseguita all'interno del text processor **gema-ge1**), una stringa di caratteri è rielaborata e restituita.

```

function Sant(inp)
  if string.sub(inp, 1, 8) == "Auguri a" then
    if string.find("AEIOU",string.sub(inp,10,10),1,true) == nil then
      return "San"..string.sub(inp, 9)
    else
      return "Sant'"..string.sub(inp, 10)
    end
  else
    return inp
  end
end
end

```

1.1.2 Strumenti con LUA

Lua 5 è stato inglobato nel text processor **gema** col nome di **ge1**.

