1.1 Scriptol

Scripting 2001-2007

Scriptol *(Scriptwriter Oriented Language*) è stato creato nel 2001 da Denis.G. Sureau; gli ultimi aggiornamenti risalgono al 2007.

Scriptol è una linguaggio di scripting object oriented di cui esiste un interprete e due generatori di codice per C++ e PHP; apparentemente non è più in evoluzione, la documentazione è parzialmente carente, è affetto da alcuni errori e le tre versioni non sono implementate allo stesso livello.

La sintassi del linguaggio è di per se lineare, tuttavia accetta delle varianti sintattiche che ne offuscano la semplicità.
Ogni variabile deve essere dichiarata ed avere un tipo, l'ambito di validità delle variabili è all'interno delle funzioni o dei blocchi in cui sono dichiarate, le variabili dichiarate all'esterno delle funzioni sono, come in C, valide ovunque.
I tipi di dato sono i consueti numerici (`number`, `int`, `natural` e `real`), booleani, testo (`text`) e matrici (`array`) ai quali si aggiungono `dir` e `files`; inoltre dizionari accessibili per chiave testuale e strutture XML, entrambi i tipi hanno metodi per il loro trattamento, compresi operatori insiemistica, purtroppo non molto documentati.

I tipi basilari hanno dei costruttori, usati soprattutto per conversioni di tipo, ad esempio da testo a numero. Gli `array`s, tramite i metodi `push` e `pop` sono facilmente utilizzabili come memorie stack, i `dict` (dizionari) accettano anche chiavi duplicate, i due metodi *`dict`*`.key()` e *`dict`*`.value()` permettono, all'interno di una `scan`, di accedere a chiave e valore.

I delimitatori di blocco delle *`istruzioni`* di controllo sono */`istruzione`*, ma è possibile una notazione "corta" unilineare in cui l'istruzione è seguita da `?` oppure `let`. Una variante dell'`if` opera come l'istruzione `case`, ma esiste anche una variante del `do`, la `do case` con un'interessante estensione data dalla parola riservata `always` che permette di eseguire del codice comune a tutti i casi.

`for` *`variabile`* `in` *`insieme`* e `scan` *`insieme`* sono entrambi utilizzabili per percorrere un insieme, nel primo caso l'accesso all'elemento è tramite *`variabile`*, nel secondo è *`insieme`*`[]`. Esiste anche una poco significativa `enum` per raggruppare assegnazioni.

Gli operatori aritmetici e di confronto sono quelli usuali con `<>` per diverso e `mod` che fornisce il resto di una divisione fra interi, anche gli operatori logici and e or sono fra quelli largamente adottati `&` e `|`, inoltre `^` (exclusive or), `~` (not), `<<` e `>>` (shift left e shift right); `&`, `|`e `^` sono anche operatori insiemistici.
Infine da segnalare in per verificare l'inclusione di un elemento in una sequenza.
Tranne tranne che per l'operatore unario, che ha la precedenza sugli altri, non esistono precedenze, quindi queste devono essere espresse tramite parentesi.

Non esiste `goto`, ma `break` e `continue` per uscire da un blocco di istruzioni o per ripetere dall'inizio il ciclo.

I parametri delle funzioni possono essere funzioni, inoltre i parametri accettano anche valori di default.

Fra l'insieme, esauriente, delle funzioni di input output, anche la possibilità di leggere e scrivere testi XML; `echo` e `print`, la seconda da un minimo di formattazione: spazio fra gli operandi e ritorno a capo alla fine, gestiscono l'output verso la console.

La dichiarazione di funzioni è semplice:

```
tiporitorno [tiporitorno ...] nomefunzione(tipo parametro ...)
    ...
    return valore1 [valore2 ...]
```

Come si può notare dalla sintassi, l'istruzione `return` chiude il blocco della funzione e può restituire più valori. Le variabili sono passate per valore, tranne nel caso di insiemi o oggetti o

se nella funzione si fa prcedere tipo e nome dalla parola chiave `alias`; è possibile anche indicare dei valori di difetto:

L'esempio che segue è la determinazione dei primi valori dei coefficienti di $(a+b)^n$.

```
//Scriptol Interpreter / Alpha version A-22 (April 2006)
www.scriptol.net
//
// a cumbersome Tartaglia (Niccolò Cusano) triangle
int GodelNumber(text txt)
   int GN = 1
   text t
   for t in txt
     if t = "a" ? GN * 3
     if t = "b" let GN * 5
   /for
   return GN
array m1 = {"a","b"}
array m2 = {"a","b"}
array m0 = {}
text item1,item2
int G,i,j,nT,f
for i in 1..10
  m0 = {}
  scan m1
    scan m2
       m0.push(m1[]+m2[])      // make multiplication
    /scan
  /scan
  m2 = m0
  for item1 in m0
    if item1 <> ""
      nT = 1   // equal addendum counter
      G = GodelNumber(item1)
      f = m0.find(item1)
      array ar = m0[f..]
      for item2 in ar
        if GodelNumber(item2) = G
          nT+1
          m0[m0.find(item2)] = ""
        /if
      /for
      echo " ";echo nT
    /if
  /for
  print
/for
```

```
>si s.sol

Scriptol Interpreter / Alpha version A-22 (April 2006)
www.scriptol.net

 1 2 1
 1 3 3 1
 1 4 6 4 1
 1 5 10 10 5 1
 1 6 15 20 15 6 1
 1 7 21 35 35 21 7 1
 1 8 28 56 70 56 28 8 1
 1 9 36 84 126 126 84 36 9 1
```

```
 1 10 45 120 210 252 210 120 45 10 1
 1 11 55 165 330 462 462 330 165 55 11 1
>Exit code: 0
```

Figura 1

Il sorgente che segue dimostra l'utilizzo della struttura XML, il programma estrae dai dati la struttura gerarchica contenuta.

```
dict emp
text searchSlave(text nm)
      text x = ""
      scan emp
            if emp.value() = nm
                  x = emp.key()
                  emp[x] = "*****"  ` remove item
                  break
            /if
      /scan
return x          // return must be last instruction
xml dom
/xml
dom doc
doc.load("dipend.xml")
array boss
int level = 0
// empty tag
xml ins
      emp
      /emp
/xml
doc.emp.end()           // after last tag <emp>
doc.addNext(ins)
doc.setAttribute("boss","teta")
doc.setData("epsilon")
doc.reset()
doc.down()
while doc.found()
    text name = doc.getData()
    text head = doc.getValue("boss")
    if head = "" let boss.push(name)      ` boss of bosses
    emp[name] = doc.getValue("boss")
let doc.inc()         // while terminator
print boss
while boss.size() > 0
      text bs = boss.pop()
      boss.push(bs)     ` take item
      text x = searchSlave(bs)
      if x = ""
            boss.pop()
            level = level-1
      else
            level = level+1
            for int i in 1..level echo "  "     `  "  ".dup(level)  don't
works
            print x
            boss.push(x)
      /if
/while
```

Figura -2

Qui di seguito i dati di input ed il risultato:

```
<?xml version='1.0'?>
<db>
<emp>aleph </emp>
<emp boss="aleph">bet </emp>
<emp boss="bet">gamma </emp>
<emp boss="bet">teta </emp>
<emp boss="aleph">iota </emp>
</db>
```

```
Scriptol Interpreter Alpha v. A-30 2006/11, 6.3
based - www.scriptol.net

aleph
  bet
    gamma
    teta
      epsilon
  iota
```

Figura -3