# Reinventare l'inventario

Roberto Giacomelli
giaconet.mailbox@gmail.com

14 dicembre 2010

**Sommario**

Nell'articolo viene spiegato come elaborare dati relativi ad articoli di magazzino in un formato opportuno, per mezzo del linguaggio di scripting Lua e di LaTeX, per la produzione di report d'inventario.

## Indice

# 1 In cosa consiste un inventario?

Moltissimi esercizi commerciali alla fine dell'anno redigono l'inventario del magazzino. Si tratta di un elenco di oggetti dei quali se ne riportano le quantità rilevate ed il costo di acquisto dal fornitore. Normalmente il report dell'inventario viene strutturato per reparti o per fornitori, oppure ancora per classi di articolo, riportando i relativi totali. In generale un lavoro piuttosto noioso se non è supportato da software opportuni.

## 1.1 Una soluzione con il software libero

Nella maggior parte dei casi non si dispone di programmi di reporting, così ci si arrangia con un foglio di calcolo come OpenOffice.org Calc, utilizzando funzioni di subtotale per ottenere le somme parziali. Per evitare errori e noiosi interventi manuali, in puro stile hacker, scriveremo un programma che costruisce un file e lo compila con LaTeX per produrre il documento relativo all'inventario nel formato pdf.

## 1.2 Inventario con Lua più LaTeX

Il programma elaborerà i dati in un formato di testo puro e LaTeX assumerà il ruolo di motore di reporting. I programmi necessari alla soluzione presentata sono disponibili per un gran numero di sistemi operativi, solo che dovrete pensarci da soli ad installarli.

Ho scelto di scrivere il programma in Lua (versione 5.1), perché è un linguaggio di scripting che non necessita di compilazione, semplice ma potente, nato proprio con l'obbiettivo di svolgere elaborazioni di *data description*. Per produrre il PDF ho scelto LaTeX perché elabora file di puro testo per fornire un output tipograficamente perfetto :-)

Occorre quindi che sia Lua 5.1 sia una distribuzione TeX (per esempio TeX Live) siano installati sul vostro sistema.

# 2 Formato dei dati d'inventario

Le caratteristiche degli articoli del nostro magazzino sono peculiari. Per fissare le idee stabiliremo la seguente serie di campi abbastanza comuni tanto da consentire facilmente estensioni:

- **codice**: identificativo univoco articolo;
- **descrizione**: breve testo descrittivo;
- **fornitore**: codice univoco del fornitore;
- **reparto**: codice univoco del reparto;
- **costo**: costo di acquisto di un pezzo;
- **quantità**: pezzi rilevati a magazzino.

Caliamo direttamente questa struttura in un formato *auto-descrittivo* definito da un lista di elementi **item** simili al seguente, dove i dati sono racchiusi tra parentesi graffe e dove l'identazione ed i ritorni a capo sono opzionali (nella tastiera italiana le graffe si ottengono con le combinazioni `Alt Gr + Shift + [` e `Alt Gr + Shift + ]`):

```
item{code="C123",
     descr="NOME MODELLO - Classe",
     supplier="MYSUP",
     department="MYDEP",
     cost=123.45,
     qty=1,
     }
```

Come è evidente nell'esempio, i valori testuali sono racchiusi tra doppi apici mentre i valori numerici sono rappresentati con il punto decimale (non possiamo usare la virgola, simbolo già previsto dalla sintassi per la separazione dei campi (a proposito l'ultima virgola è opzionale)). Possiamo anche inserire liberamente caratteri spazio, tranne che all'interno dei valori di tipo testo perché ovviamente diventano parte del valore stesso, così i due codici C123 e  C123 o C 123 individuano due diversi articoli!

Nel formato key=value le parole chiavi sono sempre le stesse, così un blocco di dati potrebbe apparire come segue:

```
item{code="W1234",descr="W123 -Stilo", supplier="LOL",department="CRT",cost=   12,qty= 8}
item{code="Z4321",descr="Z432 -Penna", supplier="LOL",department="CRT",cost= 5.20,qty= 1}
item{code="Z5521",descr="Z552 -Gomma", supplier="LOL",department="CRT",cost= 2.80,qty=10}
item{code="Q0021",descr="Q002 -Matita",supplier="LOL",department="CRT",cost= 1.20,qty=80}
item{code="Q9921",descr="Q992 -Penna", supplier="OLO",department="CRT",cost=15.20,qty= 8}
...
```

Questo miscuglio di formato e sintassi è già interessante di per se e, considerando che le parentesi tonde che racchiudono gli argomenti di una funzione in Lua possono essere omesse se l'argomento è un unica tabella (od anche una stringa letterale), diventa anche codice Lua perfettamente eseguibile...

# 3 Le tabelle di Lua

In Lua esiste solo una struttura dati complessa con cui si implementano gli array, le liste e le strutture ad albero, chiamata tabella. Il linguaggio prevede che per creare un oggetto tabella si debba usare un costruttore che nella forma più semplice è **{}**. Ciascun elemento di una tabella viene restituito specificando la chiave tra parentesi quadre od in *dot notation* come nel seguente esempio:

```
-- nb: il doppio trattino indica un commento
-- per prima cosa creiamo una nuova tabella e ne assegnamo
-- il riferimento ad una variabile locale
local mytab = {}

mytab["primo"] = 1
mytab["secondo"] = 2

-- dot notation:
print(mytab.primo)   --> stampa 1

mytab.terzo = mytab.primo + mytab.secondo

print(mytab.terzo)   --> stampa 3

local sec = "secondo"
print(mytab[sec])    --> stampa 2
```

Introdotte anche solo così velocemente le tabelle, si può intravedere il carattere del linguaggio Lua, semplice essenziale ma anche piuttosto elegante e potente. Il formato dati ideato precedentemente rappresenta in Lua una serie di chiamate alla funzione **item** alla quale è passato come argomento il record dei dati dell'articolo d'inventario sotto forma di tabella.

## 3.1 Esercizio

Per prendere confidenza con le tabelle di Lua scriviamo una versione della funzione **item** perché restituisca il totale di magazzino in termini di costo, quantità e numero di articoli.

Intanto alcune note pratiche: apriamo il nostro editor di testo preferito (per Windows per esempio il Blocco Note), ed incolliamo nella relativa finestra i dati di esempio precedenti o quelli generati da un vostro foglio di calcolo. Salviamo il file con il nome di inv.txt. Creiamo poi un secondo file chiamato count.lua in cui avremo inserito il codice seguente:

```
-- tabella codici articoli
local codeart = {}

-- variabili di conteggio
local totalcost = 0
local totalqty = 0
local numart = 0

function item(record)
    -- memorizzo il codice articolo in tabella
    -- in questo modo conteggio i codici
    -- non le chiamate ad item
    if not codeart[record.code] then
       codeart[record.code] = true
       numart = numart + 1
    end

    totalcost = totalcost + record.cost*record.qty
    totalqty = totalqty + record.qty
end

dofile("inv.txt")
```

```
print("Totali articoli: "..numart)
print("Costo totale: "..totalcost)
print("Quantita totale pezzi: "..totalqty)
```

A questo punto per stampare il conteggio d'inventario lanciamo da console il comando:

```
lua count.lua
```

# 4 Costruire l'albero ovvero mettere tabella dentro tabella

Ammettiamo che il magazzino sia suddiviso in reparti e che a sua volta i reparti siano suddivisi per fornitore, a cui apparterranno i relativi articoli. Questa struttura è un *albero*, e se rammentate come sono organizzati i vostri file nel disco rigido ne avrete un secondo esempio assai famigliare. Bene, scriveremo la nostra funzione **item** in modo che costruisca l'albero corrispondente all'inventario di magazzino utilizzando una tabella che conterrà le tabelle dei reparti ciascuno contenente le tabelle dei fornitori, che ancora conterranno le tabelle degli articoli, traducendo esattamente la struttura con cui abbiamo deciso di organizzare il magazzino.

Presa confidenza con la struttura a livelli il codice non è ne difficile ne lungo:

```
-- nuovo oggetto tabella d'inventario
local store = {}

-- definizione della funzione item:
-- main data processing function
function item(record)
    -- livello principale: reparti
    -- creo per comodita una variabile che contiene l'ID
    -- del reparto
    local dep = record.department

    -- creo la tabella relativa al reparto se ancora non esiste
    if not store[dep] then
        store[dep] = {}
    end

    -- livello fornitore
    -- variabile di comodo che contiene il codice del fornitore
    local sup = record.supplier
    -- var di comodo per il riferimento alla tabella di reparto
    local tabdep = store[dep]

    -- se non esiste la tabella fornitore la creo
    if not tabdep[sup] then
        tabdep[sup] = {}
    end

    -- livello articolo
    -- vars di comodo alla tabella fornitore ed al codice articolo
    local tabsup = tabdep[sup]
    local codeart = record.code

    -- se il codice esiste aggiorniamo per quantita
    -- altrimenti creo nuova tabella articolo dove memorizzo
    -- la descrizione, il costo e la quantita
    if tabsup[codeart] then
        tabsup[codeart].qty = tabsup[codeart].qty + record.qty
    else
        tabsup[codeart] = {}
        local tabart = tabsup[codeart]
        tabart.descr = record.descr
        tabart.cost = record.cost
        tabart.qty = record.qty
    end
end
```

### 4.1 Raccolta dati

I vantaggi della struttura ad albero creata dalla funzione **item** sono due: oltre a classificare per livelli gli articoli è possibile inserire lo stesso articolo con successive chiamate alla funzione. Durante la raccolta dati infatti capita che pezzi diversi dello stesso articolo vengano registrati in tempi diversi.

La raccolta dati può essere effettuata connettendo un lettore di codici a barre ad un notebook e gestendo i dati con un foglio elettronico o meglio con un database. In questo modo l'inventario procede molto velocemente e senza fatica, a condizione che sugli articoli sia presente il codice a barre, tramite un etichetta per esempio, e che siano già stati inseriti i dati corrispondenti ai codici.

## 5 Visitare l'albero

La seconda parte del problema è generare il report. Occorre infatti *visitare* l'albero per trarne i totali dei vari livelli e produrre con essi il file sorgente LaTeX. Per ragioni di tempo, terremo semplice il codice lasciando irrisolti alcuni problemi come un migliore utilizzo della potenza compositiva di LaTeX e l'ordinamento in ordine alfabetico delle categorie e classi del nostro magazzino. Useremo per semplicità l'ambiente verbatim con impaginazione su due colonne, trascrivendo per ciascun articolo la descrizione, la quantità, il costo, ed il valore totale.

Chiamiamo **renderTree** la funzione Lua che visita l'albero iterativamente e scrive le righe in una tabella che, per motivi di performance, verrà trascritta nel file sorgente tutta in una volta. Useremo l'iteratore **pairs** per leggere i valori nelle tabelle dei vari livelli, iteratore che restituisce due argomenti la chiave ed il suo valore nella tabella indicata.

```
-- tabella per memorizzare le righe di output
local invLines = {}

-- larghezze in caratteri delle singole colonne
local descrCol = 25
local qtyCol = 8
local costCol = 10

-- formatta un intero
local function formatInt(n)
   local nn = string.format("%d",n)
   return string.rep(" ", qtyCol-#nn) .. nn
end

-- formatta un importo
local function formatDec( x )
   local xx = string.gsub(string.format("%.2f", x),"%." , ",")
   return string.rep(" ", costCol-#xx) .. xx
end

-- funzione di comodo per aggiungere elementi riga
local function addRow( s )
   invLines[#invLines+1] = s
end

-- funzione di formattazione riga articolo
-- l'operatore di concatenazione stringhe si scrive ..
local function addItem(d, q, c)
   -- d descrizione
   -- q quantita
   -- c costo di un unico pezzo

   if #d > descrCol then
      d = string.sub(d,1,descrCol)
   else
      d = d .. string.rep(" ", descrCol-#d)
   end

   addRow(d.." "..
         formatInt(q).." "..
         formatDec(c).." "..
         formatDec(q*c)
         )
end
```

```
-- funzione di formattazione riga totale
local function addTot( q, c )
    local d = "Tot." .. string.rep(" ", descrCol-4)
    addRow(d .. " " ..
           formatInt(q) .. " " ..
           string.rep(" ", costCol+1)..
           formatDec(c)
           )
end

-- funzione principale creazione "vista"
local function renderTree()
   -- contatori generali
   local totstoreCost = 0
   local totstoreQty = 0

   local rule = string.rep("-",descrCol+qtyCol+2*costCol+3)

   -- ciclo principale per ciascun reparto
   for keydep, dep in pairs(store) do
      -- scriviamo l'intestazione di reparto
      addRow(rule)
      addRow("Reparto: " .. keydep)
      addRow(rule)
      addRow("")

      -- contatori di reparto
      local totdepCost = 0
      local totdepQty = 0

      -- ciclo per ogni fornitore del reparto
      for keysup, sup in pairs( dep ) do
          -- scriviamo una riga d'intestazione per il fornitore
          addRow("Fornitore :" .. keysup)
          addRow(rule)

          -- contatori del fornitore
          local totsupCost = 0
          local totsupQty = 0

          -- ciclo sugli articoli del fornitore
          for keycode, code in pairs( sup ) do
              addItem(code.descr, code.qty, code.cost)
              totsupCost = totsupCost + code.qty*code.cost
              totsupQty = totsupQty + code.qty
          end
          -- chiudo il fornitore scrivendone i totali
          addRow(rule)
          addTot(totsupQty, totsupCost)
          addRow(rule)
          addRow("")

          -- aggiorno i totali di reparto
          totdepCost = totdepCost + totsupCost
          totdepQty = totdepQty + totsupQty
      end
      -- chiudo il reparto scrivendone i totali
      addRow(rule)
      addRow("Fine reparto")
      addTot(totdepQty, totdepCost)
      addRow(rule)
      addRow("")

      -- aggiorno i totali d'inventario
      totstoreCost = totstoreCost + totdepCost
      totstoreQty = totstoreQty + totdepQty
```

```
    end

    -- scrivo i totali di magazzino
    addRow(rule)
    addTot(totstoreQty, totstoreCost)
    addRow(rule)
end
```

# 6 Creare il file sorgente

Per questo basterà creare delle stringhe opportune che contengano le necessarie istruzioni LaTeX di preambolo e di chiusura racchiudendo il testo tra doppie parentesi quadre:

```
local docPreamble = [[
%
% this file was created by makeinv.lua script
% Copyright (c) 2010 Roberto Giacomelli
%

\documentclass[a4paper,11pt]{report}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[italian]{babel}

\usepackage[margin=1.6cm,bottom=2.5cm]{geometry}
\usepackage{inconsolata}

%
%
%
\begin{document}
\twocolumn
\scriptsize
\sffamily

\begin{verbatim}]]

local docEnd = [[
\end{verbatim}
\end{document}
]]
```

In questo modo il sorgente LaTeX è *incastonato* nel sorgente dello script. Naturalmente esiste la possibilità di prelevare il codice necessario da file esterni in modo da rendere il tutto più facilmente gestibile. Poi sarà possibile costruire il file chiamato **inventary.tex** con il seguente codice:

```
local filename = "inventary"

-- create the LaTeX source file
local function makesource( fn )
        -- apertura file
        local f = assert(io.open( fn  .. ".tex", "w"))

        -- data assembly
        local s = table.concat(invLines,"/n")

        -- scrittura dati
        f:write(s)

        -- chiusura del file
        f:close()
end
```

# 7 Compilare il file con LaTeX

Certo, occorre una libreria in grado di interagire con il sistema operativo ed in Lua questa libreria si chiama **os**. Ecco il codice:

```
local function makepdf( fn )
       print("Start pdfLaTeX run...")
       os.execute("pdflatex " .. fn)
       os.remove(fn..".log")
       os.remove(fn..".aux")
       print("End")
end
```

# 8 Codice finale

Il codice finale si occupa di lanciare le funzioni per l'esecuzione del lavoro, eccolo:

```
-- execute all the item functions in the source datafile
dofile("inv.txt")

-- setup the start of the source LaTeX file
addRow(docPreamble)

-- make the pdf file of inventary
renderTree()

-- setup the end of the source LaTeX file
addRow(docEnd)

makesource( filename )
makepdf( filename )

-- end of script file
```

# 9 Lo script per intero

Assemblando le varie funzioni scritte il codice completo dello script è il seguente:

```
-- makeinv.lua
-- a script to make an invetary
-- Copyright (c) 2010 Roberto Giacomelli
-- see the page robitex.wordpress.com/legalese/ for licence details
-- enjoy

-- nuovo oggetto tabella d'inventario
local store = {}

-- definizione della funzione item:
-- main data processing function
function item(record)
    -- livello principale: reparti
    -- creo per comodita una variabile che contiene l'ID
    -- del reparto
    local dep = record.department

    -- creo la tabella relativa al reparto se ancora non esiste
    if not store[dep] then
        store[dep] = {}
    end

    -- livello fornitore
    -- variabile di comodo che contiene il codice del fornitore
    local sup = record.supplier
    -- var di comodo per il riferimento alla tabella di reparto
    local tabdep = store[dep]
```

```
    -- se non esiste la tabella fornitore la creo
    if not tabdep[sup] then
        tabdep[sup] = {}
    end

    -- livello articolo
    -- vars di comodo alla tabella fornitore ed al codice articolo
    local tabsup = tabdep[sup]
    local codeart = record.code

    -- se il codice esiste aggiorniamo per quantita
    -- altrimenti creo nuova tabella articolo dove memorizzo
    -- la descrizione, il costo e la quantita
    if tabsup[codeart] then
        tabsup[codeart].qty = tabsup[codeart].qty + record.qty
    else
        tabsup[codeart] = {}
        local tabart = tabsup[codeart]
        tabart.descr = record.descr
        tabart.cost = record.cost
        tabart.qty = record.qty
    end
end

-- tabella per memorizzare le righe di output
local invLines = {}

-- larghezze in caratteri delle singole colonne
local descrCol = 25
local qtyCol = 8
local costCol = 10

-- formatta un intero
local function formatInt(n)
   local nn = string.format("%d",n)
   return string.rep(" ", qtyCol-#nn) .. nn
end

-- formatta un importo
local function formatDec( x )
   local xx = string.gsub(string.format("%.2f", x),"%." , ",")
   return string.rep(" ", costCol-#xx) .. xx
end

-- funzione di comodo per aggiungere elementi riga
local function addRow( s )
   invLines[#invLines+1] = s
end

-- funzione di formattazione riga articolo
-- .. operatore di concatenazione stringhe
local function addItem(d, q, c)
   -- d descrizione
   -- q quantita
   -- c costo di un unico pezzo

   if #d > descrCol then
      d = string.sub(d,1,descrCol)
   else
      d = d .. string.rep(" ", descrCol-#d)
   end

   addRow(d.." "..
         formatInt(q).." "..
         formatDec(c).." "..
         formatDec(q*c)
         )
```

```
end

-- funzione di formattazione riga totale
local function addTot( q, c )
    local d = "Tot." .. string.rep(" ", descrCol-4)
    addRow(d .. " " ..
           formatInt(q) .. " " ..
           string.rep(" ", costCol+1) ..
           formatDec(c)
           )
end

-- funzione principale creazione "vista"
local function renderTree()
   -- contatori generali
   local totstoreCost = 0
   local totstoreQty = 0

   local rule = string.rep("-",descrCol+qtyCol+2*costCol+3)

   -- ciclo principale per ciascun reparto
   for keydep, dep in pairs(store) do
      -- scriviamo l'intestazione di reparto
      addRow(rule)
      addRow("Reparto: " .. keydep)
      addRow(rule)
      addRow("")

      -- contatori di reparto
      local totdepCost = 0
      local totdepQty = 0

      -- ciclo per ogni fornitore del reparto
      for keysup, sup in pairs( dep ) do
          -- scriviamo una riga d'intestazione per il fornitore
          addRow("Fornitore: " .. keysup)
          addRow(rule)

          -- contatori del fornitore
          local totsupCost = 0
          local totsupQty = 0

          -- ciclo sugli articoli del fornitore
          for keycode, code in pairs( sup ) do
              addItem(code.descr, code.qty, code.cost)
              totsupCost = totsupCost + code.qty*code.cost
              totsupQty = totsupQty + code.qty
          end
          -- chiudo il fornitore scrivendone i totali
          addRow(rule)
          addTot(totsupQty, totsupCost)
          addRow(rule)
          addRow("")

          -- aggiorno i totali di reparto
          totdepCost = totdepCost + totsupCost
          totdepQty = totdepQty + totsupQty
      end
      -- chiudo il reparto scrivendone i totali
      addRow(rule)
      addRow("Fine reparto")
      addTot(totdepQty, totdepCost)
      addRow(rule)
      addRow("")

      -- aggiorno i totali d'inventario
      totstoreCost = totstoreCost + totdepCost
```

```
      totstoreQty = totstoreQty + totdepQty
   end

   -- scrivo i totali di magazzino
   addRow(rule)
   addTot(totstoreQty, totstoreCost)
   addRow(rule)
end

local docPreamble = [[
%
% this file was created by makeinv.lua script
% Copyright (c) 2010 Roberto Giacomelli
%

\documentclass[a4paper,11pt]{report}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[italian]{babel}

\usepackage[margin=1.6cm,bottom=2.5cm]{geometry}
\usepackage{inconsolata}

%
%
%
\begin{document}
\twocolumn
\scriptsize
\sffamily

\begin{verbatim}
]]

local docEnd = [[
\end{verbatim}
\end{document}
]]

local filename = "inventary"

-- create the LaTeX source file
local function makesource( fn )
        -- apertura file
        local f = assert(io.open( fn  .. ".tex", "w"))

        -- data assembly
        local s = table.concat(invLines,"\n")

        -- scrittura dati
        f:write(s)

        -- chiusura del file
        f:close()
end

local function makepdf( fn )
       print("Start pdfLaTeX run...")
       os.execute("pdflatex " .. fn)
       os.remove(fn..".log")
       os.remove(fn..".aux")
       print("End")
end

-- execute all the item functions in the source datafile
dofile("inv.txt")
```

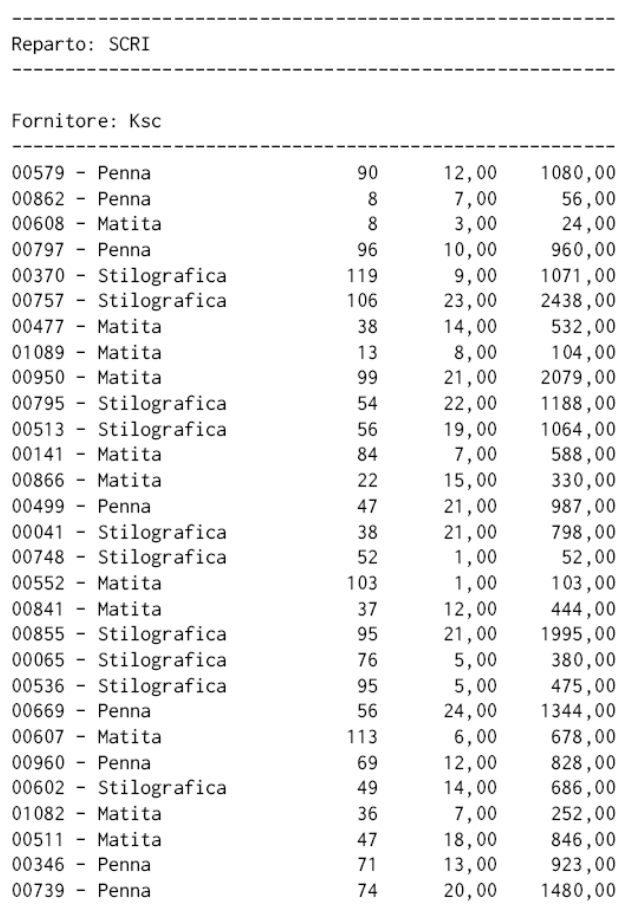
```
-- setup the start of the source LaTeX file
addRow(docPreamble)

-- make the pdf file of inventary
renderTree()

-- setup the end of the source LaTeX file
addRow(docEnd)

makesource( filename )
makepdf( filename )

-- end of script file
```



```
-----------------------------------------------------
Reparto: SCRI
-----------------------------------------------------

Fornitore: Ksc
-----------------------------------------------------
00579 - Penna                  90      12,00    1080,00
00862 - Penna                   8       7,00      56,00
00608 - Matita                  8       3,00      24,00
00797 - Penna                  96      10,00     960,00
00370 - Stilografica          119       9,00    1071,00
00757 - Stilografica          106      23,00    2438,00
00477 - Matita                 38      14,00     532,00
01089 - Matita                 13       8,00     104,00
00950 - Matita                 99      21,00    2079,00
00795 - Stilografica           54      22,00    1188,00
00513 - Stilografica           56      19,00    1064,00
00141 - Matita                 84       7,00     588,00
00866 - Matita                 22      15,00     330,00
00499 - Penna                  47      21,00     987,00
00041 - Stilografica           38      21,00     798,00
00748 - Stilografica           52       1,00      52,00
00552 - Matita                103       1,00     103,00
00841 - Matita                 37      12,00     444,00
00855 - Stilografica           95      21,00    1995,00
00065 - Stilografica           76       5,00     380,00
00536 - Stilografica           95       5,00     475,00
00669 - Penna                  56      24,00    1344,00
00607 - Matita                113       6,00     678,00
00960 - Penna                  69      12,00     828,00
00602 - Stilografica           49      14,00     686,00
01082 - Matita                 36       7,00     252,00
00511 - Matita                 47      18,00     846,00
00346 - Penna                  71      13,00     923,00
00739 - Penna                  74      20,00    1480,00
```

Figura 1: The final PDF document of example inventary

### 9.1 Applicazione

Scaricate il seguente file di prova chiamato appunto inv.pdf contenente 5000 articoli casuali. Purtroppo Wordpress per ragioni di sicurezza non consente di fare l'upload di file con estensione .txt per cui dovrete fare un passaggio in più copiando ed incollando i dati dal file in formato PDF (generato con SciTE).

Salvate lo script precedente in un file chiamato **makeinv.lua** (eventualmente assegnategli preventivamente i permessi di esecuzione), ed eseguitelo con il comando:

```
lua makeinv.lua
```

Otterrete questo file PDF in meno di un decimo di secondo!!!

## 10 Conclusioni

Lo script è veloce ed indipendente dal sistema operativo: legge un file di testo puro contenente i dati, li assembla nella struttura che corrisponde all'organizzazione reale del nostro magazzino e restituisce un file pronto per essere

compilato da LaTeX.

Generando il sorgente LaTeX lo script può essere definito come un componente *metaLaTeX*. Tutti i dati sono testuali quindi trasparenti e facili da personalizzare. LaTeX poi produce un risultato tipografico eccellente.

L'idea si applica alla costruzione di report di dati complessi come un database cinematografico od un bilancio aziendale, oppure ad inventari ancora più complessi per esempio per una catena di negozi.

# 11 Bibliografia

Testo consigliato in assoluto per la programmazione in Lua è il PiL, scritto in maniera brillante dall'Autore stesso del linguaggio Roberto Ierusalimschy, pertanto non deve mancare nella nostra libreria.

# 12 Auguri

Tempo d'inventario, tempo di Natale e fine d'anno dunque...
Auguri a tutti!!