Γλώσσες Σεναρίων

Βασισμένο στο Κεφάλαιο 13
Πραγματολογία των Γλωσσών Προγραμματισμού
Michael L. Scott

Παρουσίαση: Ν. Παπασπύρου

Κοινά χαρακτηριστικά

- Χρήση μαζική (batch) αλλά και διαδραστική (interactive)
- Οικονομία έκφρασης
- Απουσία δηλώσεων, απλοί κανόνες εμβέλειας
- Ευέλικτο, δυναμικό σύστημα τύπων
- Εύκολη πρόσβαση σε άλλα προγράμματα
- Πολύπλοκο ταίριασμα προτύπων (pattern matching) και επεξεργασία συμβολοσειρών
- Τύποι δεδομένων υψηλού επιπέδου

Περιοχές εφαρμογών (i)

- Γλώσσες εντολών φλοιού (shell command languages)
  - {c,tc,k,ba,z}?sh
- Επεξεργασία κειμένου και παραγωγή εκθέσεων
  - sed, awk, Perl
- Μαθηματικά και στατιστική
  - “3M” (Maple, Mathematica, Matlab), S, R

Περιοχές εφαρμογών (ii)

- Γλώσσες “συγκόλλησης” (glue) και σενάρια γενικού σκοπού
  - Tcl, Python, Ruby
- Γλώσσες επέκτασης (extension languages)
  - JavaScript, Visual Basic, AppleScript, Emacs Lisp

Περιοχές εφαρμογών (iii)

- Σενάρια στον παγκόσμιο ιστό
  - CGI — Perl, ...
  - Server-side embedded — PHP, Visual Basic, ...
  - Client-side — JavaScript, ...
  - Applets — Java, ...
  - διάφορες άλλες τεχνολογίες — XSLT, XML, ...

Γλώσσες φλοιού

```
#!/bin/bash

for fig in *.eps
do
  target=${fig%.eps}.pdf
  if [ $fig -nt $target ]
  then
    ps2pdf $fig
  fi
done
```

Μετατροπή αρχείων PS σε PDF. Υποθέτουμε ότι έχουμε ήδη κάποια αρχεία PDF και θέλουμε να δημιουργήσουμε όσα λείπουν.

Sed

```
# label (target for branch):
:top
/<[hH][123]>.*<\/[hH][123]>/ {      ;# ταίριαξε όλη την επικεφαλίδα
    h                               ;# αντίγραφε το χώρο προτύπων
    s/\(<\/[hH][123]>\).*$/\1/      ;# σβήσε το κείμενο μετά το κλείσιμο
    s/^.*\(<[hH][123]>\)/\1/        ;# σβήσε το κείμενο πριν το άνοιγμα
    p                               ;# τύπωσε ό,τι απέμεινε
    g                               ;# επανάφερε το χώρο προτύπων
    s/<\/[hH][123]>//               ;# σβήσε το κλείσιμο
    b top
}                                   ;# και πήγαινε στην αρχή του σεναρίου
/<[hH][123]>/ {                     ;# ταίριαξε το άνοιγμα (μόνο)
    N                               ;# επέκτεινε την αναζήτηση
    b top                           ;#    στην επόμενη γραμμή
}                                   ;# και πήγαινε στην αρχή του σεναρίου
d                                   ;# αν δεν ταιριάζει, σβήσε
```

Σενάριο για την εξαγωγή επικεφαλίδων HTML. Υποθέτει ότι οι ετικέτες των επικεφαλίδων είναι ταιριασμένες και όχι ένθετες.

Awk

```
/<[hH][123]>/ {
    # εκτέλεσε αυτό το μπλοκ αν η γραμμή περιέχει ετικέτα ανοίγματος
    do {
        open_tag = match($0, /<[hH][123]>/)
        $0 = substr($0, open_tag)       # σβήσε το κείμενο πριν από αυτή
                                        # $0 είναι η τρέχουσα γραμμή εισόδου
        while (!/<\/[hH][123]>/) {      # τύπωσε τις εσωτερικές γραμμές
            print                       #    ολόκληρες
            if (getline != 1) exit
        }
        close_tag = match($0, /<\/[hH][123]>/) + 4

        print substr($0, 0, close_tag)  # τύπωσε μέχρι το κλείσιμο
        $0 = substr($0, close_tag + 1)  # σβήσε μέχρι το κλείσιμο
    } while (/<[hH][123]>/)             # επανάλαβε όσο υπάρχει άνοιγμα
}
```

Σενάριο awk για την εξαγωγή επικεφαλίδων HTML. Τυπώνει τις εσωτερικές γραμμές βηματικά. Υποθέτει καλοσχηματισμένη HTML.

Awk

```
BEGIN {                     # λέξεις "θόρυβος"
    nw["a"] = 1;    nw["an"] = 1;   nw["and"] = 1;  nw["but"] = 1
    nw["by"] = 1;   nw["for"] = 1;  nw["from"] = 1; nw["in"] = 1
    nw["into"] = 1; nw["nor"] = 1;  nw["of"] = 1;   nw["on"] = 1
    nw["or"] = 1;   nw["over"] = 1; nw["the"] = 1;  nw["to"] = 1
    nw["via"] = 1;  nw["with"] = 1
}
{
  for (i=1; i <= NF; i++) {
    if ((!nw[$i] || i == 0 || $(i-1) ~ /[:-]$/) && ($i !~ /.+[A-Z]/)) {
      # μετατροπή σε κεφαλαίο
      $i = toupper(substr($i, 1, 1)) substr($i, 2)
   }
   printf $i " ";      # μην προσθέσεις αλλαγή γραμμής
  }
  printf "\n";
}
```

Σενάριο awk για τη μορφοποίηση τίτλων. Μετατρέπει σε κεφαλαίο το πρώτο γράμμα των «σημαντικών» λέξεων.

Perl

```
while (<>) {                                # επανάλαβε για κάθε γραμμή εισόδου
    next if !/<[hH][123]>/;                   # άλμα στην επόμενη επανάληψη
    while (!/<\/[hH][123]>/) { $_ .= <>; }  # πρόσθεσε επόμενη γραμμή στο $_
    s/.*?(<[hH][123]>.*?<\/[hH][123]>)//s;
        # ελάχιστο ταίριασμα; η έκφραση με παρενθέσεις μπαίνει στο $1
    print $1, "\n";
    redo unless eof;    # συνέχισε χωρίς να διαβάσεις επόμενη γραμμή
}
```

Σενάριο Perl για την εξαγωγή επικεφαλίδων HTML. Τοποθετεί τις επικεφαλίδες σε προσωρινή μνήμη, αντί να τις τυπώνει βηματικά.

Perl

```
$#ARGV == 0 || die "usage: $0 pattern\n";
open(PS, "ps -w -x -o'pid,command' |");  # διαταγή εμφάνισης διεργασιών
<PS>;                                    # αγνόησε τη γραμμή επικεφαλίδας
while (<PS>) {
    @words = split;             # ανάλυσε τη γραμμή σε λέξεις
    if (/$ARGV[0]/i && $words[0] ne $$) {
        chomp;                  # σβήσε την αλλαγή γραμμής στο τέλος
        print;
        do {
            print "? ";
            $answer = <STDIN>;
        } until $answer =~ /^[yn]/i;
        if ($answer =~ /^y/i) {
            kill 9, $words[0]; # το σήμα 9 στο Unix προκαλεί άμεσο τερματισμό
            sleep 1;           # περίμενε να ολοκληρωθεί η 'kill'
            die "unsuccessful; sorry\n" if kill 0, $words[0];
        }                      # το kill 0 ελέγχει την ύπαρξη μιας διεργασίας
    }
}
```

Σενάριο Perl για τον άμεσο τερματισμό διεργασιών.

Tcl

```
if {$argc != 1} {puts stderr "usage: $argv0 pattern"; exit 1}
set PS [open "|/bin/ps -w -w -x -opid,command" r]

gets $PS                        ;# αγνόησε τη γραμμή επικεφαλίδας
while {! [eof $PS]} {
    set line [gets $PS]         ;# στο τέλος αρχείου επιστρέφεται κενή γραμμή
    regexp {[0-9]+} $line proc
    if {[regexp [lindex $argv 0] $line] && [expr $proc != [pid]]} {
        puts -nonewline "$line? "
        flush stdout            ;# τύπωσε τώρα το μήνυμα στην οθόνη
        set answer [gets stdin]
        while {! [regexp -nocase {^[yn]} $answer]} {
            puts -nonewline "? "
            flush stdout
            set answer [gets stdin]
        }
        if {[regexp -nocase {^y} $answer]} {
            set stat [catch {exec kill -9 $proc}]
            exec sleep 1
            if {$stat || [exec ps -p $proc | wc -l] > 1} {
                puts stderr "unsuccessful; sorry"; exit 1
            }
        }
    }
}
```

Σενάριο Tcl για τον άμεσο τερματισμό διεργασιών.

Python

Σενάριο Python για τον άμεσο τερματισμό διεργασιών.

```
import sys, os, re, time
if len(sys.argv) != 2:
    sys.stderr.write('usage: ' + sys.argv[0] + ' pattern\n')
    sys.exit(1)

PS = os.popen("/bin/ps -w -w -x -o'pid,command'")
line = PS.readline()                # αγνόησε τη γραμμή επικεφαλίδας
line = PS.readline().rstrip()       # αφαίρεσε το τέλος γραμμής
while line != "":
    proc = int(re.search('\S+', line).group())
    if re.search(sys.argv[1], line) and proc != os.getpid():
        print line + '? ',
        answer = sys.stdin.readline()
        while not re.search('^[yn]', answer, re.I):
            print '? ',            # το κόμμα εμποδίζει την αλλαγή γραμμής
            answer = sys.stdin.readline()
        if re.search('^y', answer, re.I):
            os.kill(proc, 9)
            time.sleep(1)
            try:                    # θα προκληθεί εξαίρεση αν η
                os.kill(proc, 0)    # διεργασία δεν υπάρχει πια
                sys.stderr.write("unsuccessful; sorry\n"); sys.exit(1)
            except: pass      # μην κάνεις τίποτα
        sys.stdout.write('')  # εμπόδισε το αρχικό κενό στην επόμενη εκτύπωση
    line = PS.readline().rstrip()
```

Ruby

Σενάριο Ruby για τον άμεσο τερματισμό διεργασιών.

```
ARGV.length() == 1 or begin
  $stderr.print("usage: #{$0} pattern\n"); exit(1)
end^L

pat = Regexp.new(ARGV[0])
IO.popen("ps -w -w -x -o'pid,command'") {|PS|
  PS.gets                   # αγνόησε τη γραμμή επικεφαλίδας
  PS.each {|line|
    proc = line.split[0].to_i
    if line =~ "pat" and proc != Process.pid then
      print line.chomp
      begin
        print "? "
        answer = $stdin.gets
      end until answer =~ /^[yn]/i
      if answer =~ /^y/i then
        Process.kill(9, proc)
        sleep(1)
        begin       # περίμενε εξαίρεση (δεν υπάρχει η διεργασία)
          Process.kill(0, proc)
          $stderr.print("unsuccessful; sorry\n"); exit(1)
        rescue      # χειριστής -- μην κάνεις τίποτα
        end
      end
    end
  }
}
```

Emacs Lisp

```
(setq-default line-number-prefix "")
(setq-default line-number-suffix ") ")
(defun number-region (start end &optional initial)
  "Add line numbers to all lines in region.
With optional prefix argument, start numbering at num.
Line number is bracketed by strings line-number-prefix
and line-number-suffix (default \"\" and \") \")."
  (interactive "*r\np")     ; τα ορίσματα όταν καλείται από το πληκτρολόγιο
  (let* ((i (or initial 1))
         (num-lines (+ -1 initial (count-lines start end)))
         (fmt (format "%%%dd" (length (number-to-string num-lines))))
                          ; δίνει "%1d", "%2d", κλπ. (όποιο χρειάζεται)
         (finish (set-marker (make-marker) end)))
    (save-excursion
      (goto-char start)
      (beginning-of-line)
      (while (< (point) finish)
        (insert line-number-prefix (format fmt i) line-number-suffix)
        (setq i (1+ i))
        (forward-line 1))
      (set-marker finish nil))))
```

Web Scripting

- HTML
- Δυναμικό περιεχόμενο
  - σενάρια εξυπηρετητή (server-side scripting)
    - CGI
    - ενσωματωμένα, π.χ. PHP
  - σενάρια πελάτη (client-side scripting)
    - π.χ. Javascript
    - μικροεφαρμογές (applets)
- XML, XSL, XSLT, XHTML, DTD, XSD, XPath, XSL-FO, *ουφ...*