Ειδικά Θέματα Βιοπληροφορικής

Διάλεξη 7η :

Perl (5) : Regular expressions, Άσκηση 5η.

Regular expressions

Τα regular expressions (regexp, "κανονικές εκφράσεις" ή "κανονικές παραστάσεις") είναι ίσως το πλέον διάσημο χαρακτηριστικό της γλώσσας (και, συνηθέστατα, ένα από τα πλέον δυσνόητα τμήματα των προγραμμάτων σε perl). Πρόκειται για μία τεχνική χειρισμού και επεξεργασίας αλληλουχιών χαρακτήρων που βασίζεται στην αναζήτηση και εύρεση μοτίβων σε αυτές τις αλληλουχίες. Ο βασικός τελεστής που σηματοδοτεί τη χρήση των regexps είναι το =~
Με τη μορφή παραδειγμάτων :

Regular expressions

```
#!/usr/bin/perl -w

while ( $line = <STDIN> )
  {
  if ( $line =~ /ATTGACG/ )
    {
      print "A match was found in line $line";
    }
  }
```

Regular expressions

```
# ./test.pl
CGATGCATCGATGC
CGTGTAGCAGCTGTCGAT
CGATCGAATTGACGGTCA
A match was found in line CGATCGAATTGACGGTCA
CAGTCAGCATCGATC
```

Regular expressions

```
#!/usr/bin/perl -w

(@ARGV==1) or die "Usage: test.pl <search string>\n";

while ( $line = <STDIN> )
  {
  if ( $line =~ /$ARGV[0]/ )
    {
      chomp( $line );
      print "Line $line matches\n";
    }
  }
```

Regular expressions

```
# ./test.pl
Usage: test.pl <search string>
# ./test.pl ATGCTA
CACATTGCATTGCA
CATGCACATGCATCTATC
CACTGTAC
CACTGTCATTCTAATGCTACATC
Line CACTGTCATTCTAATGCTACATC matches
CACATGACTATCAGT
CACTGTACGACTATCTC
```

Regexps : απλοί χαρακτήρες

Απλοί χαρακτήρες (όπως γράμματα και αριθμοί) αναζητούνται ως έχουν (τα "T" και "F" στα παραδείγματα που ακολουθούν αντιστοιχούν στα "True" και "False") :

```
T: ( "test" =~ /es/ )
F: ( "whatisnot" =~ /whatiz/ )
F: ( "Whatisnot" =~ /whatis/ )
```

Regexps : τελεία

Η τελεία ταιριάζει με έναν (και μόνο ένα) οποιοδήποτε χαρακτήρα εκτός από το \n :

```
T: ( "test" =~ /t..t/ )
F: ( "test" =~ /t...t/ )
T: ( "test" =~ /t.s./ )
T: ( "test" =~ /..s./ )
```

Regexps : \w

Το \w (για "word") ταιριάζει με έναν (και μόνο ένα) αλφαριθμητικό χαρακτήρα (δηλ. a-z A-Z 0-9) :

```
T: ( "A test" =~ /A.\w\wst/ )
F: ( "A test" =~ /A\w.\wst/ )
T: ( "A test" =~ /..\w\wst/ )
```

Regexps : \W

Το \W (με κεφαλαίο W) είναι η άρνηση του \w : ταιριάζει με έναν (και μόνο ένα) μη αλφαριθμητικό χαρακτήρα (δηλ. οποιοδήποτε εκτός από τα a-z A-Z 0-9) :

```
T: ( "A test" =~ /\w\W\w\w\w\w/ )
F: ( "A test" =~ /\W\w\w\w\w\w/ )
F: ( "A test" =~ /.\W..\W./ )
T: ( "A test" =~ /.\W..\w./ )
```

Regexps : \s

Το \s ταιριάζει με έναν (και μόνο ένα) 'λευκό' χαρακτήρα (δηλ. κενό, tab, return, newline, form feed, \t\r\n\f) :

```
T: ( "A test" =~ /.\s..../ )
T: ( "A test" =~ /.\s/ )
F: ( "A test" =~ /\W\stes/ )
T: ( "A test" =~ /\w\stes/ )
```

Regexps : \S

Το \S είναι η άρνηση του \s : ταιριάζει με έναν (και μόνο ένα) μη 'λευκό' χαρακτήρα :

```
T: ( "A test" =~ /\S\s..../ )
T: ( "A test" =~ /\S\s/ )
F: ( "A test" =~ /\W\Stes/ )
T: ( "A test" =~ /\w\s\S\S\S/ )
```

Regexps : \d και \D

Το \d (για "digit") ταιριάζει με έναν (και μόνο ένα) αριθμητικό χαρακτήρα (δηλ. 0-9). Το \D είναι η άρνηση του \d και ταιριάζει με έναν (και μόνο ένα) μη αριθμητικό χαρακτήρα :

```
T: ( "121st test" =~ /\w\d\d\St ..../ )
T: ( "121st test" =~ /\w\d\d\St..../ )
F: ( "121st test" =~ /\w\d\d\S\s..../ )
T: ( "121st test" =~ /\w\d\d\S..../ )
F: ( "121st test" =~ /\w\D\d\S..../ )
T: ( "121st test" =~ /\w\d\d\D\S..../ )
```

Regexps : ομάδες χαρακτήρων

Όπως και στην grep (του unix), σύνολα χαρακτήρων μπορούν να δηλωθούν μέσω της χρήσης των '[' και ']' όπως για παράδειγμα [1-4a-d] ή [\w\d._]
Τα \w \s και \d μπορούν να χρησιμοποιηθούν στο εσωτερικό των αγκυλών με το συνηθισμένο νόημα τους.
Εάν ο πρώτος χαρακτήρας μετά την αριστερή αγκύλη είναι το ^ τότε η επιλογή αντιστρέφεται (δηλ. το [^0-9] σημαίνει "οποιοσδήποτε χαρακτήρας που δεν είναι αριθμητικό ψηφίο").

Regexps : \

Εάν αναζητούμε 'ειδικούς' χαρακτήρες όπως την τελεία ή το \ ή το $, θα πρέπει με κάποιο τρόπο να μπορούμε να αναιρέσουμε την ειδική τους χρήση. Ο τρόπος είναι η χρήση του \ πριν από τους εν λόγο χαρακτήρες, π.χ. χρησιμοποιούμε (στο regexp) το \\ για να βρούμε τον χαρακτήρα '\', το \$ για το χαρακτήρα '$', ...

Regexps : \Q και \E

Εάν θέλουμε όλοι οι χαρακτήρες ενός μοτίβου να χρησιμοποιηθούν ως έχουν (χωρίς, δηλαδή, να τους αποδίδεται οποιοδήποτε 'ειδικό' νόημα), μπορούμε να εσωκλείσουμε το μοτίβο σε ένα ζεύγος χαρακτήρων ελέγχου, τα \Q και \E. Αυτό που κάνουν τα \Q-\E είναι να προσθέτουν ένα \ μπροστά από κάθε χαρακτήρα του μοτίβου που περιβάλουν.

```
T: ( "123st.!@#$%^" =~ /\Q.!@#$%^\E/ )
```

Regexps : i

Εάν θέλουμε η αναζήτηση ενός μοτίβου να είναι επιτυχής άσχετα με το εάν οι χαρακτήρες που το αποτελούν είναι κεφαλαίοι ή μικροί, χρησιμοποιούμε το 'i' μετά το τέλος του μοτίβου :

```
F: "This is a test" =~ /THIS/
T: "This is a test" =~ /THIS/i
T: "This is a test" =~ /THIS\sIS/i
F: "This is a test" =~ /THIS\SIS/i
```

Regexps : χαρακτήρες ελέγχου

- ? : ταιριάζει με μία ή καμία παρουσίες του μοτίβου που προηγείται.
- * : ταιριάζει με όσες (ή και καμία) παρουσίες του μοτίβου που προηγείται.
- + : ταιριάζει με μία ή περισσότερες παρουσίες του μοτίβου που προηγείται (αλλά όχι με καμία).
- | : λογικό ' ή '. Ταιριάζει με το μοτίβο που προηγείται ή με το μοτίβο που έπεται.
- ^ : ταιριάζει με την αρχή της αλληλουχίας χαρακτήρων που ερευνούμε.
- $: ταιριάζει με το τέλος της αλληλουχίας χαρακτήρων που ερευνούμε.
- () : ομαδοποιεί διαδοχικά μοτίβα.

Regexps : χαρακτήρες ελέγχου

```
?: ( "This is test number 123" =~ /123.+/ )
?: ( "This is test number 123" =~ /T*iz*/ )
?: ( "This is test number 123" =~ /T*iz/ )
?: ( "This is test number 123" =~ /^his is/ )
?: ( "This is test number 123" =~ /^\w*is/ )
?: ( "This is test number 123" =~ /^.*/ )
?: ( "This is test number 123" =~ /\d\d\D$/ )
?: ( "This is test number 123" =~ /\d\d3$/ )
?: ( "This is test number 123" =~
                /^\w*\s*\w*\s*\w*\s*\w*\s*$/ )
?: ( "This is test number 123" =~
                /^\w*\s*\w*\s*\w*\s*\w*\s*\w*\s*$/ )
?: ( "This is test number 123" =~ /(this|that)/ )
?: ( "This is test number 123" =~ /(This|that)/ )
?: ( "This is test number 123" =~ ((T|t)his|that) )
```

Regexps : χαρακτήρες ελέγχου

```
F: ( "This is test number 123" =~ /123.+/ )
T: ( "This is test number 123" =~ /T*iz*/ )
F: ( "This is test number 123" =~ /T*iz/ )
F: ( "This is test number 123" =~ /^his is/ )
T: ( "This is test number 123" =~ /^\w*is/ )
T: ( "This is test number 123" =~ /^.*/ )
F: ( "This is test number 123" =~ /\d\d\D$/ )
T: ( "This is test number 123" =~ /\d\d3$/ )
F: ( "This is test number 123" =~
                /^\w*\s*\w*\s*\w*\s*\w*\s*$/ )
T: ( "This is test number 123" =~
                /^\w*\s*\w*\s*\w*\s*\w*\s*\w*\s*$/ )
F: ( "This is test number 123" =~ /(this|that)/ )
T: ( "This is test number 123" =~ /(This|that)/ )
T: ( "This is test number 123" =~ ((T|t)his|that) )
```

Regexps : εξαγωγή μεταβλητών

Ένα από τα πλέον χρήσιμα χαρακτηριστικά των regexps είναι η δυνατότητα που παρέχουν στον χρήστη τους να εξαγάγει υπακολουθίες χαρακτήρων που ταιριάζουν με ένα μοτίβο. Πιο αναλυτικά, εάν ένα regexp αληθεύει, και εάν το regexp περιέχει μοτίβα ομαδοποιημένα σε παρενθέσεις, τότε οι διαδοχικές υπακολουθίες που αντιστοιχούν σε διαδοχικά ζεύγη παρενθέσεων αποθηκεύονται από την perl στις βαθμωτές μεταβλητές $1, $2, $3, ...

```
/(\w+)\s+(\w+)\s+/
 -----    -----
   $1       $2
```

Regexps : εξαγωγή μεταβλητών

Θέλουμε να γράψουμε ένα πρόγραμμα το οποίο θα διαβάζει την καθιερωμένη είσοδο, και θα βρίσκει (και θα τυπώνει στην καθιερωμένη έξοδο) όλα τα πλήρη ονόματα ειδών του γένους 'Homo'.

Homo is the genus that includes modern humans and their close relatives. The genus is estimated to be between 1.5 and 2.5 million years old. All species except Homo sapiens (modern humans) are extinct. Homo neanderthalensis, traditionally considered the last surviving relative, died out 30,000 years ago while recent evidence suggests that Homo floresiensis lived as recently as 12,000 years ago.

Regexps : εξαγωγή μεταβλητών

```
#!/usr/bin/perl -w

while ( $line = <STDIN> )
  {
    if ( $line =~ /(Homo)\s+(\w+)/ )
      {
        print "Homo $2\n";
      }
  }
```

```
Homo is
Homo sapiens
Homo floresiensis
```

Regexps : εξαγωγή μεταβλητών

```
#!/usr/bin/perl -w

while ( $line = <STDIN> )
  {
    if ( $line =~ /(Homo)\s+(\w\w\w+)/ )
      {
        print "Homo $2\n";
      }
  }
```

```
Homo sapiens
Homo floresiensis
```

Regexps : εξαγωγή μεταβλητών

```
#!/usr/bin/perl -w

while ( $line = <STDIN> )
  {
    if ( $line =~ /(Homo)\s+(\w\w\w+)/ )
      {
        print "Homo $2\n";
      }
  }
```

```
Homo sapiens
Homo floresiensis
```

Αλλά, ο "Homo neanderthalensis" που είναι ;

Regexps : εξαγωγή μεταβλητών

Homo is the genus that includes modern humans and their close relatives. The genus is estimated to be between 1.5 and 2.5 million years old. All species except Homo sapiens (modern humans) are extinct. Homo neanderthalensis, traditionally considered the last surviving relative, died out 30,000 years ago while recent evidence suggests that Homo floresiensis lived as recently as 12,000 years ago.

Regexps : εξαγωγή μεταβλητών

Homo is the genus that includes modern humans and their close relatives. The genus is estimated to be between 1.5 and 2.5 million years old. All species except Homo sapiens (modern humans) are extinct. Homo neanderthalensis, traditionally considered the last surviving relative, died out 30,000 years ago while recent evidence suggests that Homo floresiensis lived as recently as 12,000 years ago.

Στην πρώτη εμφάνιση του μοτίβου, το if του προγράμματος μας αληθεύει, και έτσι το υπόλοιπο της αλληλουχίας δεν ερευνάται. Έτσι προκύπτει η αναγκαιότητα για τα $` και $' :

Regexps : εξαγωγή μεταβλητών

Εάν ένα regexp αληθεύει, τότε οι βαθμωτές μεταβλητές με τα προφανή ονόματα $` $' και $& αποκτούν ως περιεχόμενα τα :

- $& : το τμήμα της αλληλουχίας που ταίριαξε με το μοτίβο που ψάχνουμε.
- $` : το τμήμα της αλληλουχίας που προηγείται του μοτίβου.
- $' : το τμήμα της αλληλουχίας που έπεται του μοτίβου.

Regexps : εξαγωγή μεταβλητών

```
#!/usr/bin/perl -w

while ( $line = <STDIN> )
  {
    while ( $line =~ /(Homo)\s+(\w\w\w+)/ )
      {
        print "Homo $2\n";
        $line = $';
      }
  }
```

```
Homo sapiens
Homo neanderthalensis
Homo floresiensis
```

Regexps : εξαγωγή μεταβλητών

Αλλά, εάν το όνομα του είδους καταλαμβάνει δύο γραμμές ;

Homo is the genus that includes modern humans and their close relatives. The genus is estimated to be between 1.5 and 2.5 million years old. All species except Homo sapiens (modern humans) are extinct. Homo neanderthalensis, traditionally considered the last surviving relative, died out 30,000 years ago while recent evidence suggests that Homo floresiensis lived as recently as 12,000 years ago.

```
# ./test.pl < input
Homo sapiens
Homo floresiensis
```

Regexps : εξαγωγή μεταβλητών

Μία λύση είναι να διαβάζουμε γραμμή-γραμμή, και στη συνέχεια να ενώσουμε τις γραμμές σε ένα string (αφού αφαιρέσουμε τα \n) :

```
#!/usr/bin/perl -w

while ( $line = <STDIN> )
   {
     chomp( $line );
     $input = $input . " " . $line;
   }

while ( $input =~ /(Homo)\s+(\w\w\w+)/ )
   {
     print "Homo $2\n";
     $input = $';
   }
```

Regexps : εξαγωγή μεταβλητών

Μία άλλη, πιο απλή λύση είναι να διαβάσουμε απ' ευθείας ολόκληρη την είσοδο (μέχρι το τέλος εισόδου) σε ένα string. Αυτό μπορεί να επιτευχθεί με τον επαναπροσδιορισμό του $/ όπως φαίνεται στον κώδικα που ακολουθεί:

```
#!/usr/bin/perl -w

$/ = undef;
$input = <STDIN>;

while ( $input =~ /(Homo)\s+(\w\w\w+)/ )
  {
    print "Homo $2\n";
    $input = $';
  }
```

Regexps : τα *? και +?

Έστω η αλληλουχία χαρακτήρων :

Well done! You did it again! Congrats!

Εάν ερευνήσουμε αυτήν την αλληλουχία χρησιμοποιώντας ως μοτίβο το /!(.*)!/ τότε υπάρχουν τρεις πιθανές λύσεις για το τι θα μπορούσε να ταιριάξει η perl με αυτό το μοτίβο:

- Το " You did it again! Congrats"
- Το " You did it again"
- Το " Congrats"

Η προκαθορισμένη συμπεριφορά της perl είναι να βρίσκει πάντα την πρώτη από αριστερά και όσο το δυνατό μεγαλύτερη υπακολουθία (σε αυτήν την περίπτωση, την πρώτη από τις τρεις πιθανές λύσεις).

Regexps : τα *? και +?

Οι χαρακτήρες ελέγχου *? και +? τροποποιούν αυτήν τη συμπεριφορά με το ζητούν από τη γλώσσα να βρει όχι τη μεγαλύτερη, αλλά τη μικρότερη από τις υπακολουθίες που ταιριάζουν με το μοτίβο. Στο παράδειγμα της προηγούμενης διαφάνειας, το μοτίβο /!(.*?)!/ θα έβρισκε για λύση την υπακολουθία " You did it again". Το +? επάγει την ίδια αλλαγή συμπεριφοράς, αλλά για τον χαρακτήρα ελέγχου +

tr: character translate

Το tr 'μεταγλωττίζει' αλληλουχίες χαρακτήρων με το να αντικαθιστά κάποιον ή κάποιους χαρακτήρες με άλλους :

```
#!/usr/bin/perl -w

while ( $line = <STDIN> )
  {
    $line =~ tr/abgdezh8iklmnxoprstyfcwo/αβγδεζηθικλμνξοπρστυφχψω/;
    print "$line";
  }

# ./test.pl
Kalhmera!
Καλημερα!
Eiste kala;
Ειστε καλα;
```

Υποκατάσταση με την s/

Η έκφραση $string =~ s/pattern1/pattern2/ βρίσκει και αντικαθιστά (στο $string) το pattern1 από το pattern2. Εάν το τελευταίο / ακολουθείται από το γράμμα g, τότε η αντικατάσταση γίνεται επαναληπτικά:

```
#!/usr/bin/perl -w

while ( $line = <STDIN> )
  {
    $line =~ s/is/is not/g;
    print "$line";
  }

# ./test.pl
This is a test. This is a second test.
This not is not a test. This not is not a second test.
```

Split και regexps

Τη συνάρτηση split την έχουμε ήδη συναντήσει στη μορφή @array = split(' ', $string). Στην πραγματικότητα, το πρώτο όρισμα της split είναι ένα regexp το οποίο χρησιμοποιείται για να διαχωρίσει το '$string' :

```
#!/usr/bin/perl -w
$/ = undef;
$input = <STDIN>;
$input =~ tr/\n/ /;
@sentences = split( /\. /, $input );
foreach $sentence ( @sentences ) { print "=> $sentence.\n"; }

# ./test.pl
This is a test. But
what isn't. Goodbye.
=> This is a test.
=> But what isn't.
=> Goodbye.
```

index(), substr() και length()

Αυτές είναι τρεις χρήσιμες συναρτήσεις για το χειρισμό strings :

- index(string, search_string, pos)

Βρες τη θέση του search_string στο string ξεκινώντας από τη θέση pos.

- substr(string, pos, length)

Επέστρεψε την υπακολουθία του string που ξεκινάει από τη θέση pos και έχει μήκος (σε χαρακτήρες) length.

- length(string)

Ποιό είναι το μήκος σε χαρακτήρες του string ?

ΑΣΚΗΣΗ

Το αρχείο /work/tmp/tmp/pdbaa.fasta περιέχει όλες τις αλληλουχίες της PDB σε FASTA format :

```
>gi|442541|pdb|102L|  Lysozyme Insertion Mutant With Ala Inserted After ...
MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAAKSELDKAIGRNTNGVITKDEAEKLFNQDVDAAVRGIL
RNAKLKPVYDSLDAVRRAALINMVFQMGETGVAGFTNSLRMLQQKRWDEAAVNLAKSRWYNQTPNRAKRVITTFRTGTWD
AYKNL
>gi|442542|pdb|103L|  Phage T4 Lysozyme Insertion Mutant With Ser, Leu, ...
MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNSLDAAKSELDKAIGRNTNGVITKDEAEKLFNQDVDAAVRG
ILRNAKLKPVYDSLDAVRRAALINMVFQMGETGVAGFTNSLRMLQQKRWDEAAVNLAKSRWYNQTPNRAKRVITTFRTGT
WDAYKNL
>gi|442544|pdb|104L|B Chain B, Lysozyme Insertion Mutant With Ala, Ala ...
MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAKSAAELDKAIGRNTNGVITKDEAEKLFNQDVDAAVRGI
LRNAKLKPVYDSLDAVRRAALINMVFQMGETGVAGFTNSLRMLQQKRWDEAAVNLAKSRWYNQTPNRAKRVITTFRTGTW
DAYKNL
>gi|442545|pdb|107L|  Lysozyme (E.C.3.2.1.17) Mutant With Ser 44 Replaced By ...
MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAKGELDKAIGRNTNGVITKDEAEKLFNQDVDAAVRGILR
NAKLKPVYDSLDAVRRAALINMVFQMGETGVAGFTNSLRMLQQKRWDEAAVNLAKSRWYNQTPNRAKRVITTFRTGTWDA
YKNL
>gi|442546|pdb|108L|  Lysozyme (E.C.3.2.1.17) Mutant With Ser 44 Replaced By ...
MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAKIELDKAIGRNTNGVITKDEAEKLFNQDVDAAVRGILR
NAKLKPVYDSLDAVRRAALINMVFQMGETGVAGFTNSLRMLQQKRWDEAAVNLAKSRWYNQTPNRAKRVITTFRTGTWDA
YKNL
>gi|442547|pdb|109L|  Lysozyme (E.C.3.2.1.17) Mutant With Ser 44 Replaced By ...
MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAKKELDKAIGRNTNGVITKDEAEKLFNQDVDAAVRGILR
NAKLKPVYDSLDAVRRAALINMVFQMGETGVAGFTNSLRMLQQKRWDEAAVNLAKSRWYNQTPNRAKRVITTFRTGTWDA
YKNL
>gi|442548|pdb|110L|  Lysozyme (E.C.3.2.1.17) Mutant With Ser 44 Replaced By ...
MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAKLELDKAIGRNTNGVITKDEAEKLFNQDVDAAVRGILR
.................
```

ΑΣΚΗΣΗ

Γράψτε και στείλτε στο διδάσκοντα μέσω e-mail ένα πρόγραμμα σε perl το οποίο θα διαβάζει ένα τέτοιο αρχείο και θα βρίσκει το μήκος σε κατάλοιπα και τον κωδικό (της PDB) της μεγαλύτερης σε μήκος αλληλουχίας. Ένα παράδειγμα χρήσης του ζητούμενου προγράμματος είναι :

```
# ./seq.pl /work/tmp/tmp/pdbaa.fasta
The longest sequence was 1733 residues long
and corresponds to entry 2B63
```