Κεφάλαιο 12: Η Γλώσσα Προγραμματισμού Perl

Σύνοψη
Η γλώσσα προγραμματισμού Perl είναι μια ιδιαίτερα εύχρηστη γλώσσα με πολλές εφαρμογές στη βιοπληροφορική. Η Perl είναι γλώσσα scripting και κατασκευάστηκε αρχικά για να διευκολύνει τους διαχειριστές συστημάτων UNIX στην καθημερινή τους δουλειά, δηλαδή στην επεξεργασία μεγάλων αρχείων, σε μαζικές αποστολές μηνυμάτων κ.ο.κ., γι' αυτό και βρήκε πολλές εφαρμογές στη βιοπληροφορική, όπου ο χειρισμός μεγάλων αρχείων και η αναζήτηση προτύπων, είναι μέρος της καθημερινής σχεδόν ενασχόλησης. Στο κεφάλαιο αυτό θα παρουσιάσουμε τα βασικά στοιχεία της γλώσσας και θα δείξουμε τη χρησιμότητά της σε πλήθος πρακτικών προβλημάτων που προκύπτουν στην ανάλυση βιολογικών (κυρίως μοριακών) δεδομένων.

Προαπαιτούμενη γνώση
Το κεφάλαιο αυτό δεν έχει ιδιαίτερες απαιτήσεις και προαπαιτούμενα πέρα από τις βασικές γνώσεις της βιολογίας και μια εξοικείωση με τους Η/Υ.

12. Εισαγωγή

Η Perl είναι μια γλώσσα προγραμματισμού ιδιαίτερα γνωστή στο χώρο του UNIX και του Linux. Το όνομα προέρχεται από τα αρχικά των όρων «*Practical Extraction and Report Language*» το οποίο σε ελεύθερη μετάφραση σημαίνει «*Γλώσσα για την Πρακτική Αναφορά και Εξαγωγή (δεδομένων)*». Ορισμένοι βέβαια, χιουμοριστικά, υποστηρίζουν ότι το όνομα Perl προέρχεται από τα αρχικά των λέξεων «*Pathologiacally Eclectic Rubbish Lister*» δηλαδή «*Παθολογικά Επιλεκτική Παρουσίαση Σκουπιδιών*». Και οι δύο εκδοχές όμως φαίνεται ότι γίνονται αποδεκτές από τον δημιουργό της γλώσσας, τον Larry Wall. Η Perl μπορεί να βοηθήσει τον προγραμματιστή στη δημιουργία και εκτέλεση προγραμμάτων τα οποία σε άλλες γλώσσες προγραμματισμού θα ήταν πιο πολύπλοκα και θα χρειαζόταν σημαντικά περισσότερος χρόνος για να υλοποιηθούν. Η Perl θεωρείται παραδοσιακά γλώσσα scripting. Scripts ονομάζονται γενικά, τα μικρά προγραμματάκια που υλοποιούνται για πολύ συγκεκριμένες δουλειές και μπορεί να είναι συχνά μίας χρήσης. Το χαρακτηριστικό τους είναι ότι διερμηνεύονται (interpret) και δεν μεταγλωττίζονται (compile), δηλαδή δεν μετατρέπονται σε γλώσσα μηχανής. Η γλώσσα κατασκευάστηκε αρχικά για να διευκολύνει τους διαχειριστές συστημάτων UNIX στην καθημερινή τους δουλειά που απαιτούσε «scripting», δηλαδή στην επεξεργασία μεγάλων αρχείων, σε μαζικές αποστολές μηνυμάτων κ.ο.κ. Γι' αυτό το λόγο βρήκε και πολλές εφαρμογές στη βιοπληροφορική, όπου ο χειρισμός μεγάλων αρχείων και η αναζήτηση προτύπων είναι μέρος της καθημερινής σχεδόν ενασχόλησης. Η Perl στηρίχθηκε σε ό,τι καλύτερο υπήρχε εκείνη την εποχή διαθέσιμο για τέτοιου είδους δουλειές και το ενσωμάτωσε. Έτσι, στην Perl θα βρούμε στοιχεία του awk, του sed, του ίδιου του bash shell, αλλά και της γλώσσας C. Προφανώς, όποιος γνωρίζει κάποιο από τα παραπάνω εργαλεία, θα βρει την Perl αρκετά εύκολη και μάλλον ενδιαφέρουσα.

Ένα βασικό χαρακτηριστικό της Perl, είναι η απλότητα της σε πολλά επίπεδα που θα εξηγηθούν παρακάτω, η οποία οδηγεί σε μια μάλλον «απότομη» καμπύλη μάθησης. Αυτό σημαίνει, ότι κάποιος που δεν γνωρίζει τη γλώσσα, μπορεί σε σύντομο χρονικό διάστημα να τη χρησιμοποιήσει για να πραγματοποιήσει κάποιες βασικές λειτουργίες (στον ίδιο χρόνο δεν θα μπορούσε να πετύχει το ίδιο σε κάποιες άλλες γλώσσες όπως η C, C++ ή Java). Το μειονέκτημα σε αυτό, είναι ότι με το "χαλαρό" τρόπο της, η Perl μπορεί να οδηγήσει κάποιες φορές τον προγραμματιστή σε λάθος. Το άλλο χαρακτηριστικό της, αποτυπώνεται στο σύνθημα «*TIMTOWTDI*», που προέρχεται από τα αρχικά της φράσης «*There Is More Than One Way To Do It*» (και προφέρεται «*Tim Toady*»). Η γλώσσα σχεδιάστηκε με αυτό σαν στόχο, υπηρετώντας δηλαδή τη βασική ιδέα ότι δεν πρέπει η γλώσσα να επιβάλλει στον προγραμματιστή το πως θα γράψει ένα πρόγραμμα. Αυτό σημαίνει ότι υπάρχουν πολλοί (και μάλλον αρκετά διαφορετικοί) τρόποι να γραφτεί ένα συγκεκριμένο πρόγραμμα, κάνοντας κάθε φορά χρήση συναρτήσεων διαφορετικής προέλευσης και λειτουργίας. Το μειονέκτημα αυτού βέβαια, είναι ότι πολλές φορές τα προγράμματα που έχει φτιάξει κάποιος, δεν είναι εύκολο να διαβαστούν και να γίνουν κατανοητά από κάποιον άλλον, ειδικά αν ο προγραμματιστής δεν επιλέξει να βάλει τα κατάλληλα σχόλια στον κώδικα. Πολλές φορές οι χρήστες της Perl κάνουν και άτυπους (ή και λιγότερο άτυπους), διαγωνισμούς για το ποιος μπορεί να γράψει το πιο μικρό ή το πιο γρήγορο πρόγραμμα για να κάνει μια συγκεκριμένη δουλειά (τα γνωστά oneliners), προγράμματα τα οποία πολλές

φορές απαιτούν...μελέτη για να καταλάβει κανείς τί ακριβώς κάνουν. Στον αντίποδα όλων αυτών, υπάρχει η γνωστή γλώσσα Python, ο μεγαλύτερος αντίπαλος της Perl, το σύνθημα της οποίας αποτυπώνεται στη φράση «*There should be one-and preferably only one-obvious way to do it*» (αλλά καλύτερα, ας αφήσουμε τις συγκρίσεις μεταξύ των γλωσσών, θα οδηγηθούμε σε άσχημα μονοπάτια!).

Τα λειτουργικά συστήματα Unix/Linux και MacOS X έχουν εγκατεστημένη την Perl από «το κουτί», ενώ είναι διαθέσιμη δωρεάν για χρήση και για συστήματα Windows. Αν χρησιμοποιούμε ένα τέτοιο λειτουργικό σύστημα, για να εγκαταστήσουμε την Perl στο σύστημά μας θα πρέπει να επισκεφτούμε τον ιστότοπο http://www.perl.org/get.html και να επιλέξουμε την έκδοση που επιθυμούμε να εγκαταστήσουμε, ενώ στη συνέχεια πρέπει να ακολουθήσουμε τα βήματα που υποδεικνύει το πρόγραμμα εγκατάστασης. Επειδή η Perl δουλεύει με τον παλιό καλό τρόπο, δηλαδή από τη γραμμή εντολών, ιδιαίτερα στα Windows πολλοί επιλέγουν να τη χρησιμοποιούν μέσα από κάποιο είδος κέλυφους (shell) που να θυμίζει και να δίνει τις αντίστοιχες ευκολίες με το περιβάλλον του UNIX. Μια πολύ καλή τέτοια λύση είναι ο εξομοιωτής **Cygwin** (www.cygwin.com), αλλά και διάφορες «native» (φυσικές) όπως λέγονται μεταφορές του περιβάλλοντος του bash στα Windows, όπως τα **UnixUtils** (http://unxutils.sourceforge.net/) και το **MinGW** (http://www.mingw.org/). Μαζί με τη γλώσσα και τον μεταγλωττιστή (compiler), εγκαθίσταται και το σύστημα τεκμηρίωσης της Perl, παραδείγματα και άλλα βοηθήματα.

12.1. Τα βασικά της Perl

Για την δημιουργία ενός αρχείου που περιέχει κώδικα Perl αρκεί να χρησιμοποιήσουμε έναν απλό κειμενογράφο. Συνιστάται βέβαια, να χρησιμοποιούνται κειμενογράφοι που υποστηρίζουν την γλώσσα αυτή ώστε να χρωματίζουν κατάλληλα τις δεσμευμένες λέξεις, τις μεταβλητές και τα διάφορα κομμάτια (blocks) του κώδικα. Αφού συντάξουμε τον κώδικά μας πρέπει να αποθηκεύσουμε το αρχείο, και συνίσταται να το αποθηκεύουμε με κατάληξη .pl ώστε να μπορεί να επεξεργασθεί από τον compiler στην περίπτωση των Windows (αλλά και να αναγνωρίζεται από τους κειμενογράφους για να έχουμε το επιθυμητό αποτέλεσμα στην οπτικοποίηση του κώδικα). Από την στιγμή που έχουμε το αρχείο, μπορούμε να το εκτελέσουμε μέσα από τη γραμμή εντολών, καλώντας την Perl με την εντολή:

```
perl file_name.pl
```

Στην περίπτωση που το αρχείο που θα εκτελέσουμε απαιτεί την εισαγωγή δεδομένων από τον χρήστη για να εκτελεστεί, τα δεδομένα αυτά θα πρέπει να εισαχθούν στην παραπάνω γραμμή μετά την ονομασία του αρχείου χωρισμένα με κενό (αυτά είναι τα ορίσματα σε ένα πρόγραμμα). Εφόσον υπάρχουν λάθη σύνταξης ο μεταγλωττιστής θα μας ειδοποιήσει στη συνέχεια με μήνυμα στην ίδια γραμμή εντολών. Εάν δεν υπάρχουν λάθη και ο κώδικάς μας είναι σωστός θα εκτελεστεί το πρόγραμμα και στην περίπτωση που δεν υπάρχουν ούτε λογικά λάθη θα δούμε το αποτέλεσμά του είτε και πάλι στην ίδια γραμμή εντολών, είτε σε κάποιο άλλο αρχείο που έχουμε ορίσει να δημιουργεί ή να τροποποιεί το πρόγραμμα. Για την εισαγωγή σχολίων στον κώδικα της Perl χρησιμοποιούμε το σύμβολο #. Παρακάτω παρουσιάζεται το πρώτο πρόγραμμα Perl:

```
#!/usr/bin/perl
#Πρώτο Πρόγραμμα
print "Hello world \n";
```

Η συνάρτηση print έχει ως όρισμα μια συμβολοσειρά και το χαρακτήρα \n ο οποίος αντιπροσωπεύει την αλλαγή γραμμής. Η εντολή print, όπως και κάθε εντολή στη γλώσσα Perl αλλά και σε άλλες γλώσσες προγραμματισμού, ολοκληρώνεται με το ελληνικό ερωτηματικό (;). Ειδικά η πρώτη γραμμή αυτού του προγράμματος, είναι προαιρετική και χρησιμεύει στο να δείξει τη θέση του μεταγλωττιστή της Perl. Με αυτόν τον τρόπο, σε συστήματα Linux/UNIX αλλά και σε εξομοιωτές, το παραπάνω πρόγραμμα μπορεί να εκτελεστεί και σαν εκτελέσιμο shell script, δηλαδή απλά με την εντολή:

```
./program.pl
```

Το ./ στην αρχή, το χρησιμοποιούμε όταν το πρόγραμμα βρίσκεται στον τρέχοντα κατάλογο (directory) του δίσκου. Φυσικά, αν το πρόγραμμα μας θέλουμε να χρησιμοποιείται γενικά, μπορούμε να το τοποθετήσουμε

σε κάποιον κατάλογο που βρίσκεται στο μονοπάτι του συστήματος (Path), και έτσι να είναι προσβάσιμο από παντού.

Κάτι άλλο που πρέπει να έχουμε υπόψη μας, είναι ότι η Perl δεν κατασκευάζει κώδικα σε γλώσσα μηχανής (όπως για παράδειγμα η C). Αυτό σημαίνει ότι τα προγράμματα, με ελάχιστες μόνο εξαιρέσεις που αφορούν μονοπάτια αρχείων στο δίσκο κλπ, θα μπορούν να χρησιμοποιηθούν αυτούσια σε όλα τα συστήματα. Το μειονέκτημα σε αυτό, είναι ότι τα προγράμματα είναι σχετικά πιο αργά στην εκτέλεση σε σχέση με τα αντίστοιχα που θα μπορούσαν να είχαν γραφτεί για παράδειγμα στη C (πάλι όμως, υπάρχουν εξαιρέσεις, ειδικά γιατί κάποιες ρουτίνες που είναι εξειδικευμένες στην επεξεργασία αρχείων και κειμένου είναι ιδιαίτερα βελτιστοποιημένες στην Perl). Εν τούτοις, η Perl δεν είναι κλασική γλώσσα με διερμηνέα (interpreter), καθώς το κάθε πρόγραμμα δεν εκτελείται γραμμή-γραμμή, αλλά διαβάζεται συνολικά, ελέγχεται για λάθη και μετασχηματίζεται σε κάποια εσωτερική ενδιάμεση μορφή πριν την εκτέλεση. Γι' αυτούς τους λόγους πολλές φορές θα δούμε ότι αναφέρεται η ύπαρξη του «μεταγλωττιστή» της Perl. Η Python λειτουργεί επίσης με ένα παρόμοιο σύστημα, ενώ στην περίπτωση της Java, η ενδιάμεση μορφή αποθηκεύεται ως bytecode το οποίο μπορεί να χρησιμοποιηθεί μετά σε κάθε σύστημα.

12.2. Μεταβλητές

Στην Perl, τα ονόματα των βαθμωτών μεταβλητών (scalars) ξεκινούν πάντα με το σύμβολο του δολαρίου ($) ακολουθούμενο από οποιοδήποτε όνομα. Κάποια ειδικά ονόματα, είναι δεσμευμένα για ειδικές λειτουργίες της γλώσσας που θα συναντήσουμε παρακάτω ($1, $2, $_, $/ κ.ο.κ.). Μια μεγάλη διαφορά από άλλες γλώσσες προγραμματισμού, βρίσκεται στο γεγονός ότι οι μεταβλητές μπορεί να περιέχουν αριθμούς, μαθηματικές εκφράσεις, σμβολοσειρές ή ακόμα και αναφορά σε άλλες μεταβλητές, ενώ το πιο σημαντικό από όλα είναι ότι δεν χρειάζεται να ορίσουμε από την αρχή τί είδους μεταβλητή θα κατασκευάσουμε. Αυτό αποτελεί μια μεγάλη ευκολία, αλλά χρειάζεται ιδιαίτερη προσοχή καθώς το πώς θα χειριστούμε τη μεταβλητή αυτή, εξαρτάται από μας γιατί μπορούμε σε μια αριθμητική μεταβλητή να εφαρμόσουμε πράξεις για αριθμούς αλλά και για συμβολοσειρές. Για την δημιουργία οποιουδήποτε τύπου μεταβλητής αρκεί να πληκτρολογήσουμε:

```
$var = ___;
```

Με τον παραπάνω τρόπο δημιουργούμε μια μεταβλητή με όνομα $var και τύπο που εξαρτάται από το τι θα βρίσκεται δεξιά του ίσον (κάνουμε δηλαδή, ανάθεση). Δεξιά από την ανάθεση, μπορεί να βρίσκεται μια βαθμωτή τιμή (scalar) ή μια έκφραση. Όταν θέλουμε σε μια μεταβλητή να αναθέσουμε μια συμβολοσειρά μπορούμε να χρησιμοποιήσουμε τα διπλά λατινικά εισαγωγικά ("") για να εισάγουμε την μεταβλητή. Επίσης με τα διπλά εισαγωγικά χρησιμοποιούμε την παρεμβολή μεταβλητών (variable interpolation) και έτσι μπορούμε να περάσουμε το περιεχόμενο άλλων μεταβλητών σε μια συμβολοσειρά, ή να χρησιμοποιήσουμε χαρακτήρες διαφυγής (π.χ. \n). Οι χαρακτήρες αυτοί και ο συμβολισμός με την ανάποδη κάθετο (\) χρησιμοποιούνται για να ορίσουμε ιδιαίτερες λειτουργίες (το \n σημαίνει «αλλαγή γραμμής», το \s σημαίνει «κενό», το \t «στηλοθέτης» κ.ο.κ.). Έτσι, με τους χαρακτήρες διαφυγής μπορούμε να εισάγουμε τέτοιες ειδικές σημασίες των χαρακτήρων, αλλά και να αποφύγουμε την ερμηνεία ενός χαρακτήρα όταν δεν το θέλουμε. Για παράδειγμα, αν θέλουμε να περάσουμε τον χαρακτήρα $ χωρίς η Perl να νομίζει ότι ορίζουμε μεταβλητή, θα πρέπει να χρησιμοποιήσουμε το \$ (και όμοια γίνεται και για άλλους τέτοιους χαρακτήρες).

Αν χρησιμοποιήσουμε τα μονά λατινικά εισαγωγικά ('') οι χαρακτήρες αναγνωρίζονται σαν απλό κείμενο όπως ακριβώς το γράφουμε. Προσοχή χρειάζονται τα ανάποδα εισαγωγικά (``), με τα οποία η Perl ανοίγει εσωτερικά το shell, εκτελείται ο κώδικας που βρίσκεται μέσα στα εισαγωγικά σαν εντολή του συστήματος, και επιστρέφει στο πρόγραμμα το αποτέλεσμα.

Εάν μέσα στο πρόγραμμά μας υπάρχει ο παρακάτω κώδικας:

```
…
$var=3;
....
$var="hello";
….
```

η μεταβλητή $var αλλάζει τύπο και περιεχόμενο. Στο παραπάνω παράδειγμα η μεταβλητή $var περιείχε αρχικά έναν ακέραιο αριθμό και έπειτα μια συμβολοσειρά. Όπως είπαμε, αντίθετα με άλλες γλώσσες η Perl δεν απαιτεί από το χρήστη να δηλώσει τον τύπο των μεταβλητών πριν τις χρησιμοποιήσει. Στη συνέχεια δίνονται κάποια παραδείγματα ορισμού ή τροποποίησης μεταβλητών στην Perl.

```
$number=5;
$name="George";
$exp=3*$number+($number+1);
$a+=5;
$b*=3;
++$a; ή $a++;
$a--;
```

12.2.1. Τελεστές

Τελεστές (Operators) όπως και στις υπόλοιπες γλώσσες, ορίζουμε τα σύμβολα ή τις δεσμευμένες εκφράσεις μια γλώσσας που πραγματοποιούν πράξεις ή συγκρίσεις μεταξύ μεταβλητών. Οι βασικοί τελεστές που χρησιμοποιούνται για αριθμητικές πράξεις, φαίνονται στον Πίνακα 12.1 ενώ οι τελεστές που χρησιμοποιούνται για τις πράξεις και τις συγκρίσεις συμβολοσειρών (string), δίνονται στον Πίνακα 12.2.

Αριθμητικοί Τελεστές	**Περιγραφή**
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο Διαίρεσης
**	Δύναμη

Πίνακας 12.1: *Αριθμητικοί τελεστές της Perl*

Αριθμητικοί Τελεστές	**Αλφαριθμητικοί Τελεστές**	**Σύγκριση**
+	.	*πρόσθεση*
*	x	*πολλαπλασιασμός*
==	eq	*Ίσο*
!=	ne	*Άνισο*
>	gt	*Μεγαλύτερο από*
<	lt	*Μικρότερο από*
>=	ge	*Μεγαλύτερο ή ίσο*
<=	le	*Μικρότερο ή ίσο*

Πίνακας 12.2: *Αριθμητικοί τελεστές πράξεων και σύγκρισης και οι αντίστοιχοι τελεστές συμβολοσειρών στην Perl. Ο πολλαπλασιασμός στην περίπτωση των συμβολοσειρών αφορά τον πολλαπλασιασμό αριθμού με συμβολοσειρά.*

12.2.2. Συναρτήσεις της Perl

Κάποιες συνήθεις συναρτήσεις που χρησιμοποιούνται για την διαχείριση μεταβλητών που περιέχουν συμβολοσειρές δίνονται στον Πίνακα 12.3:

chomp	Κόβει το τελικό \n από την μεταβλητή	
chop	Κόβει τον τελευταίο χαρακτήρα από τη μεταβλητή	
substr	$x = substr($name, 0, 1, "L");	Αντικαθιστά στην ακολουθία $name 1 χαρακτήρα ξεκινώντας από τη θέση την 0 και αποθηκεύει την νέα στην μεταβλητή $x Η μεταβλητή $name δεν αλλάζει.
	$x = substr($name, 0, 1);	Αντιγράφει ακολουθία μήκους 1 με αρχική θέση την 0 από την ακολουθία $name στην μεταβλητή $x .
index	Index($name, "k");	Δίνει την θέση του k στην ακολουθία $name
rindex	Επιστρέφει ό,τι και η index μετρώντας όμως από το τέλος της ακολουθίας.	

Πίνακας 12.3: *Συναρτήσεις διαχείρισης συμβολοσειρών στην Perl*

Για παράδειγμα, οι εντολές:

```
$name="Takis";
$x=substr($name,0,1);
```

Θα ορίσουν αρχικά τη μεταβλητή $name και θα της δώσουν περιεχόμενο, ενώ στη συνέχεια θα δημιουργήσουν μια βαθμωτή μεταβλητή που περιέχει τον πρώτο χαρακτήρα του $name (δηλαδή «Τ»).

12.2.3. Κανονική είσοδος (<STDIN>)

Χρησιμοποιώντας τη λειτουργία <STDIN> σε ένα πρόγραμμα, η Perl διαβάζει τη γραμμή από την κανονική είσοδο (δηλαδή, από το πληκτρολόγιο) μέχρι την πρώτη αλλαγή γραμμής και τη χρησιμοποιεί ως τιμή της ειδικής μεταβλητής <STDIN>. Μια συνηθισμένη διαδικασία εισόδου είναι η παρακάτω:

```
$a=<STDIN>; #Αποθήκευση του κειμένου που πληκτρολογεί ο χρήστης στη μεταβλητή $a
print $a;
```

Το ίδιο αποτέλεσμα θα μπορούσε να προκύψει και με μία μόνο εντολή:

```
chomp ($a=<STDIN>);
```

12.3. Πίνακες και λίστες

Λίστα (list) είναι μια ταξινομημένη σειρά από βαθμωτές τιμές, μεταβλητές ή εκφράσεις. Είναι στην ουσία ένα διάνυσμα. Κάποια παραδείγματα από λίστες, φαίνονται παρακάτω:

```
(1, 2, 3)
("perl", 3, 15)
($x, 3, $x+2, "$y$x")
(1..10)
($a..$b)
```

Η λίστα κατασκευάζεται δυναμικά και χρησιμοποιείται από διάφορες εντολές και συναρτήσεις. Με την χρήση λίστας για παράδειγμα, μπορούμε να κάνουμε πολύ εύκολα αλλαγή των τιμών μεταξύ δύο μεταβλητών (swap):

```
($a,$b)=($b,$a);
```

ή να αναθέσουμε μαζικά τιμές σε βαθμωτές μεταβλητές

```
($a, $b, $c)=(1,2,3);
```

Όταν όμως θέλουμε να αποθηκεύσουμε μια λίστα και να την προσπελάσουμε χρειαζόμαστε μια νέα δομή. Αυτή είναι ο πίνακας (Array). Ένας πίνακας είναι στην ουσία μια μεταβλητή που περιέχει τα περιεχόμενα μιας λίστας. Τα στοιχεία του πίνακα έχουν μια θέση το ένα μετά το άλλο, ανάλογα με την θέση που έχουν δηλωθεί κατά την δημιουργία του πίνακα. Η πιο απλή σύνταξη για την δημιουργία ενός πίνακα είναι η ανάθεση τιμών από μια λίστα:

```
@array = (     );
```

Το όνομα κάθε πίνακα, πρέπει να περιέχει υποχρεωτικά το @ σαν πρώτο χαρακτήρα. Προσοχή χρειάζεται επίσης, στο γεγονός ότι μπορεί να υπάρχει πίνακας @name και ταυτόχρονα βαθμωτή μεταβλητή $name. Επίσης, όμοια με τις βαθμωτές μεταβλητές, κάποια ονόματα είναι δεσμευμένα (@_, @ARGV κ.ο.κ.). Ανάλογα με το τι περιέχεται ανάμεσα στις παρενθέσεις ο πίνακας έχει και τον αντίστοιχο τύπο δηλαδή ο πίνακας μπορεί να περιέχει αριθμούς, συμβολοσειρές ή συνδυασμό τους:

```
@name=("John", "George", "Mike");
@name=(1..10);
@table=1;
@table=(1, 2, @name, 7);
@table=(1,2,3);
```

Η αρίθμηση των στοιχείων, όπως και στις περισσότερες άλλες γλώσσες, ξεκινάει από το 0, αλλά σε αντίθεση με τις άλλες γλώσσες το πλήθος των στοιχείων δεν είναι απαραίτητο να δηλωθεί όταν ορίζεται ο πίνακας. Μπορούμε να ορίσουμε έναν πίνακα και με τον παρακάτω τρόπο χρησιμοποιώντας τον τρόπο δήλωσης των στοιχείων του με τη σειρά:

```
$table [0] = 1;
$table [1] = 2;
$table [2] = 3;
```

Τα στοιχεία του πίνακα, είναι κανονικές βαθμωτές μεταβλητές τόσο στην ονομασία (έχουν πάντα το $ στην αρχή του ονόματος) όσο και στη λειτουργία τους. Αυτό σημαίνει ότι μπορούμε να τα χρησιμοποιήσουμε σε κάθε είδους διεργασία στην οποία θα χρησιμοποιούσαμε μια βαθμωτή μεταβλητή. Μπορούμε δηλαδή να κάνουμε πράξεις με αυτά, να πραγματοποιήσουμε ελέγχους (βλ. παρακάτω) ή να τα βάλουμε σε μια λίστα και να χρησιμοποιήσουμε τη λίστα. Για παράδειγμα θα μπορούσαμε να έχουμε τις παρακάτω περιπτώσεις:

```
$x=$table[0];
$table[1]++;
($table[0], $table[1])= ($table[1], $table[0]);
@table[0,1,2]=@table[1,1,1];
```

Αυτό πρακτικά σημαίνει, ότι μπορούμε και να ορίσουμε έναν πίνακα δίνοντας ως στοιχείο του έναν άλλον πίνακα (με αυτόν τον τρόπο η αρίθμηση των στοιχείων θα μεταφέρεται αυτόματα καθώς τα νέα στοιχεία παρεμβάλλονται στα παλιά):

```
@table=(1, 2, @name, 7)
```

Κάποιες συνήθεις συναρτήσεις που χρησιμοποιούνται για την διαχείριση πινάκων δίνονται στον Πίνακα 12.4. Για παράδειγμα, η push συντάσσεται με τον εξής τρόπο:

```
push @table,$scalar;
```

το οποίο είναι ισοδύναμο, με το να ορίσεις τον πίνακα ως εξής:

```
@table=(@table, $scalar);
```

Αντίστοιχα, η unshift συντάσσεται έτσι:

```
unshift(@table, $scalar);
```

το οποίο είναι ισοδύναμο με την εξής εντολή:

```
@table=($scalar, @table );
```

push	Εισαγωγή στοιχείου στο τέλος του πίνακα.
pop	Διαγραφή στοιχείου από το τέλος του πίνακα.
shift	Προσθέτει στοιχείο στην πρώτη θέση του πίνακα.
unshift	Διαγραφή του πρώτου στοιχείου του πίνακα
reverse	Αντιστρέφει τη σειρά των στοιχείων του πίνακα.
sort	Ταξινομεί τον πίνακα με αύξουσα σειρά. Εάν ο πίνακας είναι αλφαριθμητικός τον ταξινομεί κατά ASCII.
splice	splice(@table,2,1); Από τον πίνακα @table αφαιρεί 1 στοιχείο ξεκινώντας από την θέση 2.

Πίνακας 12.4: *Συναρτήσεις διαχείρισης πινάκων στην Perl*

Τέλος, μια πολύ χρήσιμη λειτουργία των πινάκων, είναι η μεταβλητή $#. Αν υπάρχει ένας πίνακας @name, τότε η βαθμωτή μεταβλητή $#name περιέχει αυτόματα (αποτελεί δηλαδή μια δεσμευμένη ονομασία και η γλώσσα την κατασκευάζει αυτόματα) την τιμή του μεγαλύτερου δείκτη του πίνακα αυτού. Έτσι, αν ορίσουμε π.χ. τον πίνακα @table=(1,2,3), τότε το $#table, θα έχει την τιμή 2. Το $#table είναι μια κανονική βαθμωτή μεταβλητή, άρα μπορούμε να κάνουμε χρήση του, π.χ. να ζητήσουμε το στοιχείο $table[$#table] το οποίο στη συγκεκριμένη περίπτωση θα είναι ίσο με 3. Προσοχή χρειάζεται στο εξής: λόγω του τρόπου ορισμού των πινάκων, αν ο πίνακας έχει οριστεί με «άναρχο» τρόπο, τότε είναι δυνατό να περιέχει κενά στοιχεία. Αν για παράδειγμα στον παραπάνω πίνακα δώσουμε την εντολή $table[5]=12, τότε «παρακάμπτουμε» τα στοιχεία με δείκτες (index) 3 και 4, και τοποθετούμε το στοιχείο 12 στη θέση με δείκτη 5. Αν τώρα ζητήσουμε το $#table θα πάρουμε την τιμή 5, και μπορεί λανθασμένα να θεωρήσουμε ότι ο πίνακας έχει 6 στοιχεία.

12.4. Ευρετήρια

Ευρετήριο (Hash), ή αλλιώς κατακερματισμός ή συσχετιστικός πίνακας, είναι ένα σύνολο από μεταβλητές οι οποίες επιλέγονται με βάση την τιμή κάποιου δείκτη. Αυτοί οι δείκτες, που ονομάζονται κλειδιά (keys), μπορεί να είναι βαθμωτές τιμές ή συμβολοσειρές, και χρησιμοποιούνται για την ανάκτηση των τιμών του ευρετηρίου (values). Σε ένα ευρετήριο, όμοια με το τι συμβαίνει σε έναν πίνακα, δεν χρειάζεται να ορίσουμε μέγεθος, απλώς αναθέτουμε τα ζεύγη τιμών. Τα ευρετήρια όμως σε αντίθεση με τους πίνακες, δεν έχουν σειρά στην αρίθμηση. Επίσης, τα ονόματα των ευρετηρίων ξεκινάνε πάντα με το σύμβολο %. Ουσιαστικά θα μπορούσαμε να παρομοιάσουμε ένα ευρετήριο με ένα σύνολο από δεδομένα ίδιας μορφής τα οποία πέραν της τιμής τους, έχουν και ένα μοναδικό αναγνωριστικό ώστε να ξεχωρίζονται από τα υπόλοιπα του συνόλου.

Ένα ευρετήριο αποτελείται (και κατά συνέπεια, μπορεί να οριστεί) από μία λίστα με ζεύγη τιμών. Τα ζεύγη αυτά αποτελούνται από το κλειδί και την τιμή. Το κλειδί περιγράφει την τιμή και με αυτό την προσπελαύνουμε και την χρησιμοποιούμε. Κατά την δημιουργία ενός ευρετηρίου πρέπει να δοθούν και τα δύο, και το κλειδί και η τιμή ενός ζευγαριού χωρίς αυτό να σημάνει πως δεν είναι δυνατή η μετέπειτα τροποποίηση τους. Για να ορίσουμε ένα ευρετήριο, ένας εύκολος τρόπος είναι να ακολουθούμε την εξής σύνταξη:

```
%day = ("Sun", "Sunday", "Mon", "Monday", "Tue", "Tuesday", "Wed",
"Wednesday", "Thu", "Thursday", "Fri", "Friday", "Sat", "Saturday");
```

Όπως παρατηρούμε ο τρόπος μοιάζει με τον τρόπο ορισμού ενός πίνακα, παρ' όλα αυτά κάθε ζεύγος (σύμφωνα με την θέση τους κατά τον ορισμό) αποτελεί ένα ζεύγος κλειδιού-τιμής. Ένας άλλος τρόπος για να ορίσουμε ένα ευρετήριο έτσι ώστε να μην υπάρξει περίπτωση σύγχυσης μεταξύ των ζευγών είναι:

```
%day = ( "Sun" => "Sunday", "Mon" => "Monday", "Tue" => "Tuesday", "Wed" =>
"Wednesday", "Thu" => "Thursday", "Fri" => "Friday", "Sat" => "Saturday" );
```

Πρέπει να γνωρίζουμε πως εάν έχουμε μεταβλητή τύπου πίνακα μπορούμε να την μετατρέψουμε σε ευρετήριο, για παράδειγμα:

```
%table=@table;
```

Αυτό όμως, είναι δυνατό μόνο με την βασική προϋπόθεση ότι ο αριθμός των στοιχείων του πίνακα είναι άρτιος και ότι τα περιττά στοιχεία (1, 3, 5 κ.ο.κ.) τα οποία θα αντιστοιχιθούν με τα κλειδιά, θα είναι μοναδικά (δεν είναι δυνατόν να υπάρχει δυο φορές το ίδιο κλειδί!). Το ανάποδο, δηλαδή η κατασκευή ενός πίνακα που να περιέχει τα στοιχεία ενός ευρετηρίου, μπορεί να συμβεί σε κάθε περίπτωση:

```
@table=%table;
```

Πρέπει να διευκρινίσουμε εδώ, πως οι μεταβλητές που είναι χρησιμοποιήσιμες είναι οι τιμές (values) από κάθε ζεύγος, ενώ τα κλειδιά (keys) υπάρχουν για να προσφέρουν πρόσβαση στις τιμές αυτές. Για να το κάνουμε αυτό, πρέπει να χρησιμοποιήσουμε το όνομα του κλειδιού ανάμεσα σε αγκύλες ({}) ενώ το όνομα του ευρετηρίου προηγείται με το σύμβολο $ μπροστά του. Για παράδειγμα:

```
$hash{"key"}="value";
```

Με τον παραπάνω τρόπο, μπορούμε να κάνουμε τόσο ανάθεση κλειδιών-τιμών σε ένα ευρετήριο, όσο και προσπέλαση των τιμών αν ξέρουμε το κλειδί. Προφανώς, η έκφραση $hash{"key"} είναι μια κανονική βαθμωτή μεταβλητή, την οποία μπορούμε να χρησιμοποιήσουμε όπως ακριβώς κάναμε και στην περίπτωση των στοιχείων του πίνακα. Με βάση τα όσα είδαμε, καταλαβαίνουμε πως αν θέλαμε να το περιγράψουμε με κάποιο μαθηματικό ανάλογο, το ευρετήριο είναι μια «συνάρτηση», δηλαδή μια αντιστοίχιση των στοιχείων ενός συνόλου Α (κλειδιά) στα μέλη ενός συνόλου Β (τιμές), στην οποία σε κάθε στοιχείο του Α θα πρέπει να αντιστοιχεί ένα μόνο στοιχείο του Β (έτσι καταλαβαίνουμε και το γιατί τα κλειδιά πρέπει να είναι μοναδικά). Αν τώρα, τύχει και τα στοιχεία του Β να είναι μοναδικά, τότε η συνάρτηση θα είναι αντιστρέψιμη (και όντως, υπάρχει συνάρτηση που κάνει αυτή τη δουλειά στα ευρετήρια, βλ. Πίνακα 12.5)

each	Επιστρέφει, ένα προς ένα, τα ζεύγη από τα κλειδιά και τις τιμές του ευρετηρίου (τα επιστρέφει σε λίστα)
delete	Διαγραφή του κλειδιού που δίνεται ως όρισμα. Η αντίστοιχη τιμή διαγράφεται επίσης.
reverse	Αντιστρέφει τις τιμές και τα κλειδιά (ισχύει όμως μόνο όταν και οι τιμές ειναι μοναδικές)

Πίνακας 12.5: *Συναρτήσεις διαχείρισης ευρετηρίων στην Perl*

12.5. Δομές Ελέγχου

Όπως και σε όλες τις γνωστές γλώσσες προγραμματισμού, έτσι και στην Perl υπάρχουν δομές ελέγχου και επανάληψης. Παρακάτω φαίνονται οι κυριότερες από αυτές και η λειτουργία τους.

12.5.1 If/else/elsif

Με την δομή if/else ελέγχουμε μια συνθήκη. Στο σκέλος της if τοποθετούμε την μία περίπτωση και στο σκέλος της else ελέγχεται αυτόματα η εναλλακτικής της χωρίς να ορίσουμε εμείς κάτι στη συνθήκη. Παρ'

όλα αυτά, έχουμε τη δυνατότητα να περιορίσουμε και την αντίθετη συνθήκη που ελέγχεται ορίζοντας διαφορετικές εναλλακτικές. Αυτή την λειτουργία εξυπηρετεί η ύπαρξη του σετ εντολών if/elsif/else. Στην if, όπως και προηγουμένως, βάζουμε την πρώτη μας συνθήκη, στα επόμενα μπλοκ elsif τοποθετούμε τις ενναλακτικές συνθήκες ελέγχου (οι οποίες όμως αφορούν το αρχικό σύνολο μεταβλητών) και στην περίπτωση που επιθυμούμε την τελική συνθήκη, την τοποθετούμε σε ένα μπλοκ else χωρίς να ορίσουμε περιορισμό. Παράδειγμα:

```
if (condition)
{
      …
}
elsif
{
      …
}
else
{
      …
}
```

Ένα σημείο που χρειάζεται προσοχή, είναι ότι το elsif σε άλλες γλώσσες συντάσσεται ως «elseif» ή «else if». Επίσης, στην Perl κάθε κομμάτι κώδικα (block) ή συνθήκη ελέγχου, θα πρέπει να κλείνεται σε αγκύλες ({}) ακόμα και αν περιέχει μόνο μία γραμμή. Γενικά στην Perl, οι συνθήκες ελέγχου εκτός από τις προφανείς περιπτώσεις, επιστρέφουν ψευδή τιμή (false), όταν η τιμή που πρέπει να ελεγχθεί περιέχει το 0 αλλά και όταν περιέχει την κενή συμβολοσειρά.

12.5.2. While/until

Η Perl μπορεί να επαναλάβει την εκτέλεση μιας ομάδας εντολών με τη δομή while ή until. Η συνθήκη ελέγχεται στην αρχή της δομής ελέγχου στην πρώτη επανάληψη οπότε στην περίπτωση που αυτή δεν ικανοποιείται, παραλείπεται χωρίς να εκτελεστεί ποτέ. Για παράδειγμα:

```
while (condition)
{
      …
}

until (condition)
{
      …
}
```

12.5.3. Do –while/until

Στην Perl συναντάμε και τη δομή επανάληψης do/while. Σε αυτή την δομή τοποθετούμε τη δομή ελέγχου στο τέλος, χρησιμοποιείται δηλαδή όταν θέλουμε να εκτελεσθεί τουλάχιστον μια φορά το σύνολο των εντολών πριν πραγματοποιηθεί ο έλεγχος. Στην Perl μπορούμε να αντικαταστήσουμε την εντολή while με την εντολή until τροποποιώντας την συνθήκη μας ανάλογα. Παράδειγμα:

```
do
{
      …
} while (condition)


do
{
      …
```

```
} until (condition)
```

12.5.4. For

Η εντολή for λειτουργεί όπως και στις υπόλοιπες γνωστές γλώσσες. Η συνθήκη της αποτελείται από τρία μέρη. Το πρώτο μέρος ορίζει την αρχική τιμή της μεταβλητής της συνθήκης, στο δεύτερο ορίζουμε το βήμα με το οποίο θα γίνει η μεταβολή και στο τρίτο την τελική τιμή στην οποία θα πρέπει να τερματίσει η επανάληψή μας. Παράδειγμα:

```
for ($i = 1; $i <= 10; $i ++)
{
      print "$i\n";
}
```

Η εντολή αυτή, είναι η πιο ισχυρή, καθώς μπορούμε να την παραμετροποιήσουμε με διαφορετικούς τρόπους (π.χ. στη συνθήκη μπορούμε να κάνουμε χρήση εκφράσεων όπως $x+$y<10 κ.ο.κ.), αλλα πολλές φορές για πρακτικούς λόγους, η ίδια (απλή) δουλειά γίνεται πιο εύκολα με κάποια άλλη εντολή (όπως η foreach που θα δούμε στην επόμενη ενότητα).

12.5.5. Foreach

Η συγκεκριμένη εντολή εκτελεί πανομοιότυπη λειτουργία με την for με την μόνη διαφορά πως δέχεται ένα μόνο όρισμα (μια λίστα, στα στοιχεία της οποίας ανατρέχει η επαναληπτική διαδικασία). Εκτελεί τον κώδικα που έχουμε ορίσει στο εσωτερικό, για κάθε εγγραφή της δομής μας χωρίς εμείς να ξέρουμε καν το πλήθος. Παράδειγμα:

```
@a = (1,2,3,4,5);
foreash $i (@a)
{
      print "$i\n";
}
```

Ένα ιδιαίτερο χαρακτηριστικό που χρειάζεται προσοχή, είναι ότι αν μέσα στην επανάληψη, αλλάξουμε την τιμή της προσωρινής μεταβλητής (στο παράδειγμα, της $i), τότε αλλάζει και η ίδια η τιμή του πίνακα που έχουμε δώσει σαν όρισμα (αν δώσαμε πίνακα φυσικά).

12.5.6. Last/next/redo

Τα παραπάνω αποτελούν εντολές που προσφέρουν λειτουργικότητες επανάληψης ή εξόδου σε κάποιες από τις προαναφερθείσες δομές επανάληψης.
Last: Η εντολή last χρησιμοποιείται με ανάλογο τόπο με την εντολή break την γλώσσας C, δηλαδή για να σταματήσει την επανάληψη και να βγει από το βρόχο όταν ικανοποιείται κάποια συνθήκη. Έχει εφαρμογή στις δομές ελέγχου:

- for
- foreach
- while
- until

Παράδειγμα:

```
while (condition 1)
{
      …
      if(condition 2)
      {
            …
```

```
        last;
        }
}
```

Next: Η εντολή αυτή έχει την ίδια λειτουργία που έχει η εντολή continue στην C. Τοποθετείται στο τέλος ενός μπλοκ εντολών ώστε να σηματοδοτήσει την μετάβαση, είτε στις επόμενες ελεύθερες εντολές, είτε στο επόμενο μπλοκ εντολών. Παράδειγμα:

```
while (condition 1)
{
        …
        if(condition 2)
        {
                …
        next;
}
```

Redo: Με την τοποθέτηση της εντολής redo στο τέλος ενός μπλοκ εντολών πραγματοποιείται η επανάληψη ολόκληρου του μπλοκ ακόμα και αν θεωρητικά έχει τελειώσει ο κύκλος ικανοποίησης της αρχικής συνθήκης. Παράδειγμα:

```
while (condition 1)
{
        #
        …
        if(condition 2)
        {
                …
                redo;
        }
}
```

12.6. Διαχειριστές Αρχείων (Filehandles) – Είσοδος/Έξοδος

Ένα από τα ισχυρά χαρακτηριστικά της Perl, είναι ο τρόπος με τον οποίο επεξεργάζεται αρχεία. Στην περίπτωση που τα δεδομένα εισόδου ενός προγράμματος της Perl προέρχονται από ένα εξωτερικό αρχείο, ή όταν η έξοδος αποστέλλεται σε ένα άλλο αρχείο, είναι απαραίτητη η χρήση των διαχειριστών αρχείων (filehandles). Ο διαχειριστής αρχείων είναι μια σύνδεση μεταξύ ενός προγράμματος και ενός αρχείου. Ένα πρόγραμμα μπορεί να διαβάζει από ένα διαχειριστή αρχείων ανακτώντας από το περιεχόμενό του μια γραμμή κάθε φορά (αν και όπως θα δούμε, και αυτό μπορεί να αλλάξει), και να τυπώνει σε ένα διαχειριστή προσθέτοντας με αυτόν τον τρόπο δεδομένα σε ένα αρχείο. Ένας διαχειριστής αρχείων μπορεί να έχει ένα οποιοδήποτε όνομα με κεφαλαία, και ορίζεται με τον παρακάτω τρόπο:

```
open MYFILE, 'file.txt';
```

Η συνάρτηση open δέχεται ακριβώς δύο ορίσματα. Το πρώτο είναι το όνομα του διαχειριστή αρχείων και το δεύτερο όρισμα είναι το όνομα του αρχείου που θα ανοιχθεί. Το όνομα του διαχειριστή αρχείων μπορεί να είναι ένα οποιοδήποτε όνομα, αρκεί όλα τα γράμματα του να είναι κεφαλαία. Ιδανικά κάθε αρχείο που ανοίγεται πρέπει να κλείνεται στην συνέχεια. Όταν δημιουργούμε ένα διαχειριστή αρχείων και ανοίγουμε ένα αρχείο, μπορούμε να ορίσουμε και τον τρόπο με τον οποίο θα επεξεργαστούμε το αρχείο αυτό, μέσω αυτού του διαχειριστή. Οι λειτουργίες αυτές και η σύνταξή τους, όσον αφορά τα αρχεία, φαίνονται στον Πίνακα 12.6:

open (IN, "filename")	Ανοίγει ένα υπάρχον αρχείο με το ονομα filename και ονομαζει το διαχειριστή, IN
open (OUT, "> filename")	Δημιουργεί ένα νέο αρχείο με το όνομα filename και γράφει σε αυτό χρησιμοποιώντας το όνομα διαχειριστή OUT
open (IN, ">> filename")	Προσθέτει στο τέλος ενός αρχείου με το όνομα filename και χρησιμοποιεί το όνομα διαχειριστή OUT

Πίνακας 12.6: *Τρόποι με τους οποίους συντάσσεται η εντολή open*

Παραδείγματα διαχειριστών αρχείων για άνοιγμα αρχείου και εκτύπωση σε άλλο αρχείο, φαίνονται στο παρακάτω πρόγραμμα:

```
open IN, "/etc/passwd";
$x=<IN>;
print $x;
close IN;
open OUT, ">tempfile";
print OUT "bla bla bla\n";
```

Το πρόγραμμα αυτό, ανοίγει το αρχείο /etc/passwd (αρχείο του συστήματος στο Linux), μέσω του διαχειριστή IN, διαβάζει μια γραμμή, την τυπώνει, κλείνει το διαχειριστή, ανοίγει το διαχειριστή για το tempfile και τυπώνει σε αυτό την πρόταση «bla bla bla». Η μεταβλητή <IN> είναι ο ειδικός τρόπος που έχει η γλώσσα για να συμβολίζει το τι διαβάζει από το αρχείο μέσω του διαχειριστή (προσέξτε την ομοιότητα με το STDIN). Προσοχή χρειάζεται στο γεγονός ότι αν η μεταβλητή χρησιμοποιηθεί δύο ή περισσότερες φορές, θα προχωράει κάθε φορά και ένα «βήμα», διαβάζονται κάθε φορά την επόμενη γραμμή του αρχείου. Ένας εναλλακτικός τρόπος για να προσπελάσουμε αρκετά πιο εύκολα ένα αρχείο είναι και ο εξής:

```
while(<>)
{
      print $_;
}
```

Με αυτό το απλό πρόγραμμα, που κάνει τη δουλειά της εντολής cat του Unix (και μάλιστα, ίσως την κάνει και πιο γρήγορα), προσπελαύνουμε όλο το αρχείο μέχρι να συναντήσουμε End Of File (EOF) αποθηκεύοντας σε κάθε επανάληψη μια γραμμή στην μεταβλητή $_. Η συνθήκη στο while, ικανοποιείται όσο η τιμή του <> είναι μη κενή, δηλαδή όσο διαβάζει διαδοχικές γραμμές από το αρχείο. Το <> (το «διαμάντι») είναι η πιο απλή περίπτωση διαχειριστή που χρησιμοποιεί η Perl αυτόματα. Για να πάρει τιμές, αρκεί απλά να εισάγουμε ένα αρχείο ως όρισμα εισόδου κατά την κλήση του προγράμματος από την γραμμή εντολών. Για να το κάνουμε αυτό αρκεί να γράψουμε στην γραμμή εντολών μας την κλήση ως εξής:

```
perl programme.pl file.txt
```

Με αυτόν τον τρόπο, η Perl θα ανοίξει αυτόματα το αρχείο file.txt, και θα αρχίσει να στέλνει τα περιεχόμενά του στο <>, γραμμή-γραμμή (αν και αυτό ακόμα μπορεί να αλλάξει όπως θα δούμε). Το $_ είναι το ειδικό όνομα που δίνει η Perl στη μεταβλητή που περιέχει κάθε φορά τα στοιχεία αυτά, και είναι μια δεσμευμένη ονομασία, όπως είπαμε στην αρχή. Ο χρήστης δεν μπορεί να τη δημιουργήσει αλλά ούτε και να την τροποποιήσει. Το παραπάνω πρόγραμμα και ο συγκεκριμένος τρόπος, είναι ιδιαίτερα εύχρηστος και προτιμάται για απλές δουλειές. Όταν όμως θέλουμε να κάνουμε πιο σύνθετες εργασίες (όπως π.χ. να επεξεργαστούμε παράλληλα πολλά αρχεία), ή να κάνουμε το πρόγραμμα πιο αυστηρό (π.χ. να βγάζει μήνυμα σφάλματος αν το αρχείο που δώσαμε σαν όρισμα δεν υπάρχει), τότε πρέπει να χρησιμοποιήσουμε τον κλασικό τρόπο με τους διαχειριστές αρχείων. Στην περίπτωση αυτή, τα πολλαπλά ορίσματα που θα περάσουμε στο πρόγραμμα, αποθηκεύονται αυτόματα στον πίνακα @ARGV. Έτσι, τα ονόματα των αρχείων είναι $ARGV[0], $ARGV[1] κ.ο.κ. τα οποία μπορούμε να τα χρησιμοποιήσουμε στην open. Φυσικά, με τον

ίδιο τρόπο μπορούμε να περάσουμε ως όρισμα, οποιαδήποτε άλλη βαθμωτή τιμή που θα χρησιμοποιηθεί εσωτερικά στο πρόγραμμα (και όχι μόνο ονόματα αρχείων).

12.7. Κανονικές Εκφράσεις

Με τις κανονικές εκφράσεις (regular expressions) μπορούμε να ελέγξουμε εάν μια συμβολοσειρά είναι ίδια με κάποια άλλη, ή περιέχει ένα συγκεκριμένο μοτίβο. Η λειτουργία αυτή είναι ιδιαίτερα χρήσιμη καθώς όπως είδαμε στο κεφάλαιο 4, τα μοτίβα είναι ιδιαίτερα διαδεδομένα στην ανάλυση αλληλουχιών στη βιοπληροφορική. Οι κανονικές εκφράσεις οροθετούνται από τις καθέτους (/) και περιέχουν μια ακολουθία από χαρακτήρες που πρέπει να ταιριάξουν με τους χαρακτήρες μέσα στο σώμα μιας συμβολοσειράς. Για παράδειγμα η εντολή:

```
$dna=~/GAATTC/;
```

ελέγχει αν στη συμβολοσειρά $dna περιέχεται η αλληλουχία GAATTC η οποία αντιστοιχεί σε θέση δράσης μιας περιοριστικής ενδονουκλεάσης. Όταν σε μια συνθήκη βρούμε κάτι σαν /GAATTC/, έχουμε μια απλή συνθήκη ταιριάσματος συμβολοσειρών. Επίσης είναι δυνατό μέσω των κανονικών εκφράσεων όταν βρεθεί μια ακολουθία μέσα σε μια συμβολοσειρά να αντικατασταθεί με κάτι άλλο, με την απλή εντολή /GAATTC/CTTAAG/ (σε αυτή την περίπτωση αντικαθιστούμε το GAATTC με CTTAAG).

Στην Perl οι κανονικές εκφράσεις είναι τόσο ευρέως χρησιμοποιούμενες που οι συναρτήσεις και ο τρόπος επεξεργασίας τους έχει απλοποιηθεί στο ελάχιστο διότι ενισχύονται με τη χρήση μεταχαρακτήρων και ποσοδεικτών. Ένας μεταχαρακτήρας εκπροσωπεί μια ολόκληρη κλάση χαρακτήρων. Παραδείγματος χάριν, η τελεία (.) ταιριάζει με οποιοδήποτε χαρακτήρα εκτός από την αλλαγή γραμμής, ενώ το \d δηλώνει οποιοδήποτε ψηφίο. Στον Πίνακα 12.7 παρουσιάζονται οι πιο συχνά χρησιμοποιούμενοι μεταχαρακτήρες.

Μεταχαρακτήρας	**Περιγραφή**
.	Οποιοσδήποτε χαρακτήρας εκτός από την αλλαγή γραμμής
^	Αρχή μιας γραμμής
$	Τέλος μιας γραμμής
\w	Οποιοσδήποτε χαρακτήρας λέξης
\W	Οποιοσδήποτε χαρακτήρας εκτός από χαρακτήρα λέξης
\s	Χαρακτήρας διαστήματος
\S	Οποιοσδήποτε χαρακτήρας εκτός από χαρακτήρα διαστήματος
\d	Οποιοδήποτε ψηφίο
\D	Οποιοσδήποτε χαρακτήρας εκτός από ψηφίο

Πίνακας 12.7: *Μεταχαρακτήρες κανονικών εκφράσεων*

Εξ' ορισμού, οποιοσδήποτε χαρακτήρας ή μεταχαρακτήρας σε μια κανονική έκφραση ταιριάζει ακριβώς μια φορά. Μια αναζήτηση με κανονική έκφραση μπορεί να προσπαθεί να ταιριάξει όσο το δυνατόν περισσότερες φορές μέσα στο «κείμενο» την ακολουθία μας, ή όσο το δυνατόν λιγότερες αντίστοιχα. Με την τοποθέτηση ενός ποσοδείκτη (Quantifier) μετά το χαρακτήρα, η Perl μπορεί να ταιριάξει το χαρακτήρα αυτό για συγκεκριμένο αριθμό επαναλήψεων, ή και διάστημα επαναλήψεων. Ο απλούστερος ποσοδείκτης είναι ο {n}, ο οποίος προσπαθεί να ταιριάζει το πρότυπο ακριβώς n φορές. Για παράδειγμα η εντολή:

```
$sequence=~/AAAT{5}CCG/;
```

ελέγχει αν η συμβολοσειρά $sequence ταιριάζει στην αλληλουχία AAATTTTTCCG (δηλαδή, ελέγχει το πρότυπο PROSITE A-A-A-T(5)-C-C-G). Προσέξτε, ότι αυτή είναι μια έκφραση που μπορεί να έχει αληθείς (true) ή ψευδείς (false) τιμες, και κατά συνέπεια θα πρέπει να ελεγχθεί, συνήθως σε κάποια δομή if. Φυσικά, όπως είδαμε στο κεφάλαιο 4, όλες οι εκφράσεις PROSITE αντιστοιχούν σε μια κανονική έκφραση, οπότε, μπορούμε να χρησιμοποιήσουμε αυτούσιους τους κανόνες που είδαμε εκεί για να πραγματοποιήσουμε αναζητήσεις. Στον Πίνακα 12.8 παρουσιάζονται οι ποσοδείκτες κανονικών εκφράσεων.

Ποσοδείκτης	Περιγραφή
?	0 ή 1 εμφάνιση (είναι σημαντικός, γιατί αναγκάζει το πρότυπο να μην είναι «άπληστο»)
+	1 ή περισσότερες εμφανίσεις
.	0 ή περισσότερες εμφανίσεις
{n,m}	Μεταξύ n και m εμφανίσεων
{n, }	Τουλάχιστον n εμφανίσεις
{ ,m}	Όχι περισσότερες από m εμφανίσεις

Πίνακας 12.8: *Ποσοδείκτες κανονικών εκφράσεων*

12.8. Εφαρμογές της Perl στη Βιοπληροφορική

Στην ενότητα αυτή, θα παρουσιάσουμε βήμα-βήμα, κάποια βασικά προγράμματα σε Perl, τα οποία είναι μεν είναι χρήσιμα σε διάφορα στάδια της ανάλυσης βιολογικών αλληλουχιών, αλλά ταυτόχρονα αποτελούν και ιδανικά παραδείγματα για να παρουσιαστεί και ο τρόπος χρήσης των βασικών λειτουργιών της γλώσσας.

12.8.1 Μετατροπή αρχείου Uniprot σε μορφή Fasta

Μια διαδικασία ρουτίνας στις αναλύσεις αλληλουχιών, είναι η μετατροπή μιας μορφής αρχείων σε μια άλλη. Η πιο συνηθισμένη από τις περιπτώσεις, είναι να θέλουμε να μετατρέψουμε την πληροφορία από ένα αρχείο Uniprot σε μια μορφή που θα μπορεί να χρησιμοποιηθεί από τα περισσότερα προγράμματα ανάλυσης αλληλουχιών, δηλαδή στη μορφή fasta. Για να γράψουμε ένα τέτοιο πρόγραμμα, θα πρέπει να παρατηρήσουμε το αρχείο της Uniprot και να εντοπίσουμε τη δομή του. Για το fasta θα χρειαστούμε και ένα όνομα για την αλληλουχία, οπότε η καλύτερη επιλογή είναι να χρησιμοποιήσουμε το AC ή το ID. Όλες οι γραμμές ενός αρχείου Uniprot ξεκινάνε με δυο χαρακτήρες που καθορίζουν το πεδίο (π.χ. το AC), ενώ μόνο οι γραμμές που περιέχουν την αλληλουχία ξεκινάνε με δύο κενά. Έπειτα, όλες οι γραμμές έχουν 3 κενά και μετά αρχίζει η πληροφορία. Το πρόγραμμα αυτό (uniprot2fasta.pl), είναι πολύ απλό και δίνεται παρακάτω:

```
while (<>)
{
      if ($_=~/^AC\s{3}(.*?)\;/)
      {
          print ">$1\n";
      }
      if ($_=~/^\s{5}(.*)/)
      {
          $sequence=$1;
          $sequence=~s/\s//g;
          print "$sequence\n";
      }
}
```

Για ευκολία, χρησιμοποιούμε το <>. Το πρόγραμμα διαβάζει γραμμή-γραμμή και πραγματοποιεί έλεγχο στη μεταβλητή $_. Στο πρώτο if, γίνεται ο έλεγχος για το AC. Ψάχνουμε για μία γραμμή που να ξεκινάει με AC, να ακολουθούν 3 κενά και μετά οσεσδήποτε επαναλήψεις οποιουδήποτε χαρακτήρα, μέχρι να εμφανιστεί το (;) το οποίο διαχωρίζει τα πολλαπλά AC (θέλουμε να κρατήσουμε μόνο το πρώτο από αυτά). Το ? χρησιμοποιείται εδώ για να κάνει το πρότυπο όχι άπληστο (non-greedy), δηλαδή για να το αναγκάσει να σταματήσει στη μικρότερη επανάληψη του μοτίβου που θα βρει. Το $1 στη συνέχεια, είναι μια άλλη δεσμευμένη μεταβλητή στην οποία αποθηκεύεται το περιεχόμενο της παρένθεσης που έχουμε ορίσει μέσα στο πρότυπο. Με αυτόν τον τρόπο μπορούμε να απομονώσουμε από ένα μεγάλο πρότυπο, μόνο το κομμάτι που χρειαζόμαστε (εδώ, το AC). Προφανώς, αν είχαμε περισσότερες παρενθέσεις, θα είχαμε και $2, $3 κ.ο.κ.

Στον δεύτερο έλεγχο, επιχειρούμε να εντοπίσουμε τις γραμμές που έχουν στην αρχή 5 κενούς χαρακτήρες (δηλαδή, τις γραμμές με την αλληλουχία) και να κρατήσουμε όλους τους υπόλοιπους (δηλαδή την ίδια την αλληλουχία). Επειδή στη μορφή Uniprot η αλληλουχία δίνεται σε διαδοχικές δεκάδες με κενά

ανάμεσα, το περιεχόμενο της μεταβλητής $sequence θα πρέπει σε κάθε βήμα να το αλλάζουμε. Έτσι, χρησιμοποιούμε το ~s/\s//g το οποίο αλλάζει τα κενά με τον κενό χαρακτήρα (το g στο τέλος, συμβολίζει ότι πρέπει να γίνει σε όλη τη συμβολοσειρά, globally- αν δεν το είχαμε βάλει, θα άλλαζε μόνο το πρώτο κενό). Σε κάθε επανάληψη, το πρόγραμμα τυπώνει και μία γραμμή της αλληλουχίας, μέχρι να σταματήσει να βρίσκει αντίστοιχες γραμμές. Προσέξτε ότι τα δύο μπλοκ με τα if δεν είναι απαραίτητο να βρίσκονται καν στη συγκεκριμένη σειρά. Το πρόγραμμα διαβάζει μια γραμμή και κάνει όλους τους ελέγχους ανεξάρτητα, άρα, θα μπορούσε η σειρά να είναι και η αντίστροφη.

Σε κάποιες περιπτώσεις, ιδιαίτερα αν τις αλληλουχίες πρόκειται να τις επεξεργαστούμε εμείς με κάποιο άλλο πρόγραμμα μας, ίσως είναι προτιμότερο να κατασκευάσουμε μια ειδική μορφή fasta, στην οποία η αλληλουχία θα δίνεται σε μόνο μία γραμμή. Με τον τρόπο αυτό θα μπορεί να διαβαστεί πολύ πιο εύκολα. Το παρακάτω πρόγραμμα (uniprot2line.pl) κάνει αυτήν ακριβώς τη δουλειά.

```
$/="\/\/\n";
while (<>)
{
    if ($_=~/^AC\s{3}(.*?)\;/m)
    {
        print ">$1\n";
    }
    while ($_=~/^\s{5}(.*)/mg)
    {
        $sequence=$1;
        $sequence=~s/\s//g;
        print "$sequence";
    }
print "\n";
}
```

Το πρόγραμμα αυτό, μοιάζει αρκετά με το προηγούμενο αλλά έχει κάποιες σημαντικές διαφοροποιήσεις. Στην αρχή η εντολή $/="\/\/\n" λέει στο πρόγραμμα ότι θα πρέπει να αλλάξει ο προκαθορισμένος τρόπος ανάγνωσης από αρχείο (γραμμή-γραμμή) και το πρόγραμμα να διαβάζει πλέον μέχρι να συναντήσει το «//\n». Με τον τρόπο αυτό, το πρόγραμμα σε κάθε επανάληψη θα διαβάζει μια ολόκληρη εγγραφή της Uniprot και θα την αποθηκεύει στο $_. Η άλλη σημαντική διαφορά, βρίσκεται στον χαρακτήρα m στον έλεγχο των κανονικών εκφράσεων. Ο χαρακτήρας αυτός προειδοποιεί την κανονική έκφραση ότι η συμβολοσειρά ($_) περιέχει πολλαπλές γραμμές (multilne). Τέλος, το if έχει αντικατασταθεί από το while, όπως επίσης έχει προστεθεί και το g (global) στην ταύτιση των προτύπων, για να μπορέσει να ανταποκριθεί το πρότυπο και να γίνει η σωστή ταύτιση του σε πολλαπλές γραμμές.

Τέλος, θα μπορούσε να υπάρξει και η περίπτωση στην οποία θα θέλαμε να μετατρέψουμε το αρχείο από fasta με πολλές γραμμές σε fasta με μία. Αυτή τη δουλειά κάνει το παρακάτω πρόγραμμα (fasta2line.pl):

```
$/=">";

while (<>)
{
      $entry=$_;
      chop $entry;
      $entry= ">"."$entry";
      $entry=~/>(.+?)\n(\C*)/g;
      $name=$1;
      $sequence=$2;
      $sequence=~s/\n//g;

      if ($name ne "")
      {
            print ">$name\n$sequence\n";
      }
}
```

```
$/="\n";
```

Όμοια με παραπάνω, το αρχείο διαβάζεται καταγραφή-καταγραφή (μέχρι να συναντηθεί το «>»). Στη συνέχεια, διαχωρίζει το όνομα της πρωτεΐνης από την αλληλουχία, αφαιρεί τις αλλαγές γραμμής που υπάρχουν και τυπώνει την αλληλουχία συνεχόμενη.

12.8.2. Προσομοίωση με τυχαίες αλληλουχίες

Οι προσομοιώσεις για την παραγωγή τυχαίων αλληλουχιών αποτελούν μια συνηθισμένη πρακτική σε πολλές διαδικασίες στη βιοπληροφορική, κυρίως για να εντοπιστούν οι στατιστικές ιδιότητες κάποιου φαινομένου (ροές, στατιστική σημαντικότητα στοίχισης κ.ο.κ.). Το παρακάτω πρόγραμμα φτιάχνει 500 τυχαίες ακολουθίες αποτελούμενες από 200 αμινοξέα η καθεμιά:

```
@aa = (A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y);
for ( $i=0; $i<500; $i++ ) {
  print '>Random', "$i\n";
      for( $j=0; $j<200; $j++ ) {
            $r = $aa[ int (rand 20)];
            print $r;
            print "\n" if ($j+1)%60 == 0 and $j;
       }
  print "\n";
}
```

Το πρόγραμμα στην αρχή ορίζει έναν πίνακα που περιέχει τα αμινοξέα (η σειρά τους δεν παίζει ρόλο). Μετά, εκτελεί μια επανάληψη στην οποίο χρησιμοποιείται η συνάρτηση rand για να παράγει τυχαίους αριθμούς στο διάστημα 0-20, οι οποίοι στρογγυλοποιούνται και χρησιμοποιούνται σαν δείκτες (index) στον πίνακα για να ανακτηθεί το τυχαίο αμινοξύ που παράγεται. Πριν κλείσει ο βρόχος γίνεται και ένας (προαιρετικός) έλεγχος για το αν έχει συμπληρωθεί ο αριθμός 60 που χαρακτηρίζει τις περισσότερες μορφές του αρχείο fasta. Διάφορες τροποποιήσεις θα μπορούσαν να γίνουν σε αυτό το πρόγραμμα, όπως π.χ. να δέχεται το μήκος των αλληλουχιών και τον αριθμό τους ως όρισμα, να τα παράγει από μια συγκεκριμένη κατανομή. ή να παράγει αμινοξέα όχι ισοπίθανα αλλά με βάση κάποιες προκαθορισμένες συχνότητες. Οι τροποποιήσεις αυτές αφήνονται σαν άσκηση.

12.8.3. Υπολογισμός συχνοτήτων αμινοξέων

Μια πολύ συνηθισμένη ανάλυση, είναι να υπολογίσουμε το ποσοστό των αμινοξέων σε μια αλληλουχία πρωτεϊνών, ή το ποσοστό των νουκλεοτιδίων σε μια αλληλουχία DNA/RNA. Όπως είδαμε σε προηγούμενα κεφάλαια, τέτοιες αναλύσεις μπορούν να χρησιμοποιηθούν σε προγνωστικές μεθόδους ή στην υπολογιστική γονιδιωματική. Το παρακάτω πρόγραμμα δέχεται ως είσοδο ένα αρχείο με πρωτεΐνες σε μορφή FASTA με μία γραμμή και υπολογίζει το ποσοστό των αμινοξέων κάθε πρωτεΐνης.

```
@aa = (A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y);
while (<>)
{
      if ($_=~/^>/)
      {
            $id=$_;
            chomp $id;
            print $id."\t";
            $seq=<>;
            chomp $seq;
      }
      $length=length($seq)+1;
      foreach $z(@aa)
      {
```

```
            $count = $seq =~s/$z//g;
            $diairesi = $count/$length;
            $pososto=sprintf( "%.3f", $diairesi );
            print $z."\t".$count.'/'.$length."\t[".$pososto."]\n";
      }
      print "\n";
}
```

Το πρόγραμμα στην αρχή ορίζει τον πίνακα με τα 20 αμινοξέα και στη συνέχεια διαβάζει το αρχείο εισόδου γραμμή-γραμμή. Σε κάθε γραμμή ελέγχει την ύπαρξη του χαρακτήρα > που σηματοδοτεί το header και τότε κρατάει το όνομα της πρωτεΐνης και διαβάζει την επόμενη γραμμή που είναι η αλληλουχία η ίδια (προσέξτε τη χρήση του @seq=<>). Μετά, αφού μετρήσει το μήκος της κάθε αλληλουχίας, το πρόγραμμα εκτελεί ένα βρόχο foreach στον οποίο μετράει ένα-ένα τα αμινοξέα του κάθε τύπου. Αυτό γίνεται πολύ εύκολα με την εντολή $count = $seq =~s/$z//g. Στο δεξί σκέλος, κάνει μια αντικατάσταση όλων των άλλων αμινοξέων (εκτός από αυτό που μετράει κάθε φορά) με τον κενό χαρακτήρα, και τελικά μετράει το πόσες φορές το βρήκε στην αλληλουχία (η εντολή αυτή θα μπορούσε να γίνει και με τη συνάρτηση length αλλά και με άλλους τρόπους - πχ $seq =~s/$z//g; $count=length($seq);). Τέλος, τυπώνει τα αποτελέσματα σε μία γραμμή για κάθε πρωτεΐνη χρησιμοποιώντας την sprintf για καλύτερη μορφοποίηση.

12.8.4. Εύρεση ανοιχτών πλαισίων ανάγνωσης σε αλληλουχίες DNA

Ένα πρώτο βήμα στην εύρεση γονιδίων και στη γονιδιωματική ανάλυση είναι η αναγνώριση των ανοιχτών πλαισίων ανάγνωσης (open reading frames) σε αλληλουχίες DNA. Σε αυτή την ενότητα θα παρουσιάσουμε ένα απλό τέτοιο πρόγραμμα το οποίο θα πραγματοποιεί αυτού του είδους την ανάλυση. Το πρόγραμμα αυτό δέχεται ως είσοδο μια ακολουθία DNA και αρχικά βρίσκει τη συμπληρωματική της. Στη συνέχεια, μεταφράζει και τις 2 ακολουθίες χρησιμοποιώντας τα 6 πιθανά πλαίσια ανάγνωσης και τυπώνει τις αμινοξικές αλληλουχίες των υποθετικών πρωτεϊνών που προκύπτουν από αυτό.

```
%genetic_code = (
'GCA'=>'A', #Alanine
'GCC'=>'A', #Alanine
'GCG'=>'A', #Alanine
'GCT'=>'A', #Alanine
'AGA'=>'R', #Arginine
'AGG'=>'R', #Arginine
'CGA'=>'R', #Arginine
'CGC'=>'R', #Arginine
'CGG'=>'R', #Arginine
'CGT'=>'R', #Arginine
'AAC'=>'N', #Asparagine
'AAT'=>'N', #Asparagine
'GAC'=>'D', #Aspartic acid
'GAT'=>'D', #Aspartic acid
'TGC'=>'C', #Cysteine
'TGT'=>'C', #Cysteine
'GAA'=>'E', #Glutamic acid
'GAG'=>'E', #Glutamic acid
'CAA'=>'Q', #Glutamine
'CAG'=>'Q', #Glutamine
'GGA'=>'G', #Glycine
'GGC'=>'G', #Glycine
'GGG'=>'G', #Glycine
'GGT'=>'G', #Glycine
'CAC'=>'H', #Histidine
'CAT'=>'H', #Histidine
'ATA'=>'I', #Isoleucine
'ATC'=>'I', #Isoleucine
'ATT'=>'I', #Isoleucine
```

```
'TTA'=>'L', #Leucine
'TTG'=>'L', #Leucine
'CTA'=>'L', #Leucine
'CTC'=>'L', #Leucine
'CTG'=>'L', #Leucine
'CTT'=>'L', #Leucine
'AAA'=>'K', #Lysine
'AAG'=>'K', #Lysine
'ATG'=>'M', #Methionine
'TTC'=>'F', #Phenylalanine
'TTT'=>'F', #Phenylalanine
'CCA'=>'P', #Proline
'CCC'=>'P', #Proline
'CCG'=>'P', #Proline
'CCT'=>'P', #Proline
'AGC'=>'S', #Serine
'AGT'=>'S', #Serine
'TCA'=>'S', #Serine
'TCC'=>'S', #Serine
'TCG'=>'S', #Serine
'TCT'=>'S', #Serine
'ACA'=>'T', #Threonine
'ACC'=>'T', #Threonine
'ACG'=>'T', #Threonine
'ACT'=>'T', #Threonine
'TGG'=>'W', #Tryptophan
'TAC'=>'Y', #Tyrosine
'TAT'=>'Y', #Tyrosine
'GTA'=>'V', #Valine
'GTC'=>'V', #Valine
'GTG'=>'V', #Valine
'GTT'=>'V', #Valine
'TAA'=>'-', #STOP
'TAG'=>'-', #STOP
'TGA'=>'-', #STOP
);

$seq="AAAAAAATTAATAGATGAACATATATATAGATTTTCTATATAGACCCTCTACCCGATAAGGCTAC";
$seq2=$seq;
$seq2=~tr/ATCG/TAGC/;
$seq2=reverse($seq2);

print "Forward Strand\n";
Translate($seq);

print "Reverse Strand\n";
Translate($seq2);

sub Translate{
$sub_seq=$_[0];
      for($i=0;$i<=length($sub_seq)-3;$i++)
      {
            $x=substr($sub_seq,$i,3);
            if ($x eq 'ATG')
            {
                  $pos=$i+1;
                  print "position $pos\n";
                  for($j=$i;$j<=length($sub_seq)-3;$j=$j+3)
                  {
                        $y=substr($sub_seq,$j,3);
                        $k=$genetic_code{$y};
```

```
                        if($k eq '-')
                        {
                              print"\n";
                              last;
                        }
                        print "$k";
                  }
            }
      }
}
```

Για ευκολία, στο πρόγραμμα αυτό θα έχουμε την αλληλουχία αποθηκευμένη εσωτερικά σε μια μεταβλητή. Φυσικά, μπορεί να γίνει τροποποίηση και το πρόγραμμα να την διαβάζει από αρχείο. Μεγάλο μέρος του κώδικα αναγκαστικά χρησιμοποιείται για να οριστεί ένα ευρετήριο για τη συσχέτιση των κωδικονίων με τα αμινοξέα (αλλά φυσικά, και αυτό θα μπορούσε να γίνει μέσα από εισαγωγή αρχείου). Στη συνέχεια το tr βρίσκει με μια απλή εντολή τη συμπληρωματική αλυσίδα, ενώ η reverse, αντιστρέφει τη σειρά έτσι ώστε να διαβαστεί από την ορθή κατεύθυνση (υποθέτουμε ότι το 5' άκρο είναι στα αριστερά).

Το επόμενο σημαντικό χαρακτηριστικό αυτού του προγράμματος, είναι ο ορισμός μια υπορουτίνας (sub-routine). Τέτοιες περιπτώσεις συναντάμε όταν έχουμε να πραγματοποιήσουμε μια δεδομένη ενέργεια πολλές φορές (εδώ, έχουμε τη μετάφραση των δύο αλυσίδων) και δεν θέλουμε να έχουμε διπλό κώδικα μέσα στο πρόγραμμα (τόσο για αποφυγή λαθών, όσο και για να είναι πιο ευανάγνωστο). Οι υπορουτίνες είναι πολύ βολικές γιατί κρατάνε τον κώδικα ευανάγνωστο, ενώ μπορούν να τοποθετηθούν σε οποιοδήποτε σημείο του προγράμματος και να κληθούν από οποιοδήποτε σημείο (λόγω του μεταγλωττιστή). Ο ορισμός μιας υπορουτίνας αλλά και η κλήση της, είναι απλή υπόθεση όπως φαίνεται στο παραπάνω πρόγραμμα. Το μόνο σημείο που χρειάζεται διευκρίνιση είναι στο πως η Perl περνάει τα ορίσματα της υπορουτίνας. Αυτό γίνεται με τον ειδικού σκοπού πίνακα @_ στον οποίο αυτά αποθηκεύονται. Έτσι, εσωτερικά στην υπορουτίνα το πρώτο όρισμα είναι το $_[0], το δεύτερο (αν υπάρχει) είναι το $_[1] κ.ο.κ. Προσοχή χρειάζεται στο γεγονός ότι οι μεταβλητές στην Perl είναι εξ'ορισμού "παγκόσμιες" (global), δηλαδή είναι προσβάσιμες από κάθε σημείο του κώδικα. Έτσι, για να αποφύγουμε λάθη καλό είναι να δώσουμε διαφορετικό όνομα στην ακολουθία μέσα στην υπορουτίνα. Αν θέλαμε να έχουμε ιδιωτικές μεταβλητές (private), δηλαδή μεταβλητές που θα είναι προσβάσιμες μόνο στο συγκεκριμένο σημείο (βρόχο ή υπορουτίνα), θα έπρεπε να τις ορίσουμε με το my (πχ my $seq= κ.ο.κ.).

Επί της ουσίας, η υπορουτίνα πραγματοποιεί με απλό τρόπο την μετάφραση του γενετικού υλικού. Στην αρχή διαβάζει επικαλυπτόμενα παράθυρα μήκους 3 (δηλαδή, τριπλέτες) στην αλληλουχία, για να εντοπίσει το κωδικόνιο έναρξης (ATG). Για να το κάνει αυτό, χρησιμοποιεί το substr έτσι ώστε να εξάγει την τριπλέτα και μετά την περνάει στο ευρετήριο σαν κλειδί για να βρει την αντίστοιχη τιμή. Μετά, καθώς έχει ξεκινήσει η μετάφραση εξακολουθεί να διαβάζει τριπλέτες, αλλά πλέον, μη επικαλυπτόμενες, και τυπώνει τα διαδοχικά αμινοξέα έως ότου συναντήσει κωδικόνιο λήξης (προσέξτε εδώ τη χρήση του last). Όταν τελειώσει αυτός ο κύκλος, το πρόγραμμα επιστρέφει και διαβάζει ξανά επικαλυπτόμενες τριπλέτες από το σημείο που είχε σταματήσει. Αυτό είναι πολύ σημαντικό γιατί τα ανοιχτά πλαίσια ανάγνωσης μπορεί να είναι και επικαλυπτόμενα.

12.8.5. Εύρεση βακτηριακών λιποπρωτεϊνών

Όπως είδαμε στο κεφάλαιο 5, οι βακτηριακές λιποπρωτεΐνες έχουν μια σηματοδοτική αλληλουχία (signal peptide) η οποία μοιάζει αρκετά με αυτή των εκκρινόμενων πρωτεϊνών, αλλά στο καρβοξυτελικό της άκρο φέρει μια χαρακτηριστική αλληλουχία η οποία αναγνωρίζεται από ειδικό ένζυμο, το οποίο αποκόπτει το πεπτίδιο αυτό και ακολούθως η ώριμη πρωτεΐνη προσκολλάται στα λιπίδια της μεμβράνης με ομοιοπολικό δεσμό. Η αλληλουχία που αναγνωρίζει το ένζυμο, έχει μια συντηρημένη κυστεϊνη στο καρβοξυτελικό της άκρο (περίπου στη θέση 17-30 της πρόδρομης πρωτεΐνης, εκεί που γίνεται και η τροποποίηση), ενώ στις προηγούμενες θέσεις υπάρχουν κυρίως Αλανίνες και Βαλίνες. Διάφορα πρότυπα είχαν περιγραφεί από τη δεκαετία του 1980, αλλά το πιο γνωστό είναι το λεγόμενο PS00013, όπως είναι γνωστό από τον κωδικό της PROSITE. Παρ' όλα αυτά, έχουν περιγραφεί και εναλλακτικά πρότυπα πολλές φορές ακόμα και χρόνια αργότερα, όπως το [LVI]-[ASTVI]-[GAS]-C το οποίο χρησιμοποιήθηκε για να κατασκευαστεί η βάση των βακτηριακών λιποπρωτεϊνών (DOLOP). Το 2002, οι Sutcliffe και Harrington μελετώντας λιποπρωτεΐνες από

βακτήρια θετικά κατά Gram, κατέληξαν σε ένα πιο αυστηρό αλλά ταυτόχρονα και πιο περιεκτικό πρότυπο, το οποίο περιγράφει καλύτερα τις λιποπρωτεΐνες αυτών των βακτηρίων (Sutcliffe & Harrington, 2002).

Θα χρησιμοποιήσουμε ένα αρχείο με 63 αμινοξικές αλληλουχίες πρωτεϊνών όπως αναφέρθηκαν στην εργασία των (Juncker et al., 2003). Το πρόγραμμα αυτό δέχεται ως είσοδο ένα αρχείο με πρωτεΐνες σε μορφή FASTA με μία γραμμή, ελέγχει την ύπαρξη του συγκεκριμένου κάθε φορά μοτίβου και τυπώνει τον αριθμό των πρωτεϊνών στις οποίες αυτό βρέθηκε (προσέξτε τη διαφορά των κανονικών εκφράσεων, από τα πρότυπα PROSITE).

```
while (<>){
	if	($_=~/^>(.*)/)
	{
		$name=$1;
		$seq=<>;
		if($seq=~/(.*LA[GA]C)/)
		{
			$x=length($1);
			$a=$a+1;
		}
	}
}
print "$a LIPOPROTEINS FOUND";
```

Για τα άλλα μοτίβα το περιεχόμενο της δομή ελέγχου if είναι το μόνο που αλλάζει στον κώδικα της Perl:

```
if($seq=~/(.*[LVI][ASTG][GA]C)/)
```

```
if($seq=~/(.*[^DERK]{6}[LIVMFWSTAG]{2}[LIVMFYSTAGCQ][AGS]C)/)
```

```
if($seq=~/^([MV].{0,13}[RK][^DERK]{6,20}[LIVMFESTAG][LVIAM][IVMSTAFG][AG]C)/)
```

Θα μπορούσε βέβαια να φτιαχτεί και μια υπορουτίνα για να κάνει το ίδιο και να ελέγχει όλα τα πιθανά πρότυπα (αφήνεται ως άσκηση). Τέλος, για να δείξουμε με γραφικό τρόπο την ύπαρξη των παραπάνω μοτίβων θα πρέπει να κάνουμε μια ειδικής μορφής πολλαπλή στοίχιση, στην οποία όλες οι αλληλουχίες που έχουν το συγκεκριμένο μοτίβο θα έχουν στοιχιθεί στο καρβοξυτελικό άκρο του πεπτιδίου οδηγητή, έτσι ώστε να μπορέσουμε να οπτικοποιήσουμε τη συντήρηση στην περιοχή αυτή. Το παρακάτω πρόγραμμα, για το μοτίβο το οποίο βρέθηκε στις περισσότερες πρωτεΐνες, τυπώνει την αλληλουχία των πεπτιδίων οδογητών και προσθέτει μπροστά όσους χαρακτήρες κενού (δηλαδή "–") είναι απαραίτητοι έτσι ώστε όλα τα πεπτίδια να στοιχιθούν στο καρβοξυτελικό άκρο, δηλαδή στην κυστεϊνη. Το αποτέλεσμα του προγράμματος τυπώνεται στην οθόνη και θα χρησιμοποιηθεί ως είσοδος στο πρόγραμμα **WebLogo** (http://weblogo.berkeley.edu/) (Crooks, Hon, Chandonia, & Brenner, 2004) για την οπτικοποίηση της στοίχισης.

```
while (<>)
{
if ($_=~/^>(.*)/)
	{
		$name=$1;
		$seq=<>;
		if ($seq=~/(.*[^DERK]{6}[LIVMFWSTAG]{2}[LIVMFYSTAGCQ][AGS]C)/)
		{
			$x=length($1);
			push @table,$1;
		}
	}
	if($x>$max)
	{
```

```
            $max=$x;
      }
}
foreach $signal(@table)
{
      $i="-" x ($max-length($signal));
      $signal=$i.$signal;
      print "$signal\n";
}
```

Το πρόγραμμα διαβάζει το αρχείο εισόδου γραμμή-γραμμή. Σε κάθε γραμμή ελέγχει την ύπαρξη του χαρακτήρα > που σηματοδοτεί το header και τότε κρατάει το όνομα της πρωτεΐνης και διαβάζει την επόμενη γραμμή που είναι η αλληλουχία η ίδια (προσέξτε τη χρήση του @seq=<>). Εντοπίζει το πρότυπο της λιποπρωτεΐνης, αποκόπτει το πεπτίδιο οδηγητή και το αποθηκεύει σε έναν πίνακα (όμοια έχει κάνει και για το αντίστοιχο όνομα της πρωτεΐνης). Στην επόμενη φάση, αφού έχει εντοπίσει την αλληλουχία με το μέγιστο μήκος, ανατρέχει στις αλληλουχίες του πίνακα και τυπώνει στην αρχή τους τόσα "-" όσα απαιτούνται για να επιτευχθεί η στοίχιση. Η στοίχιση των πεπτιδίων οδηγητών όπως προέκυψε από την εκτέλεση του προγράμματος αυτού, δίνεται παρακάτω.

```
      ------------------MRRCMPLVAASVAALMLAGC
      -----------------MKLKQLFAITAIASALVLTGC
      -----------------MKLLSKIMIIALAASMLQAC
      ------------MNKNRGFTPLAVVLMLSGSLALTGC
      -------------------MKRQALAAMIASLFALAAC
      -------------------MRLLPLVAAATAAFLVVAC
      ----------------------MRIVIFILGILLTSC
      --------------------MFKRFIFITLSLLVFAC
      --------------------MLKKVYYFLIFLFIVAC
      --------------------MKKILLTVSLGLALSAC
      -----------------MVKKAIVTAMAVISLFTLMGC
      -----------------MKQLIVNSVATVALASLVAGC
      -------------------MKLKTLALSLLAAGVLAGC
      --------------------MKAYLALISAAVIGLAAC
      -------------------MKLKATLTLAAATLVLAAC
      -----------------MQKTPKKLTALCHQQSTASC
      ---------------MPLPDFRLIRLLPLAALVLTAC
      ----------------MKNQVKKILGMSVVAAMVIVGC
      --------------------MKKFLPLSISITVLAAC
      -----------------MKRLFLSFVALALLAGSIAAC
      --------------------MCGKILLILFFIMTLSAC
      --------------------MSKRLLSLASLALLFGC
      -------------------MFKRRYVTLLPLFVLLAAC
      -----------------MKKIIKLSLLSLSIAGLASC
      -----------------MGRSKIVLGAVVLASALLAGC
      -------------------MKAKIVLGAVILASGLLAGC
      -----------------MNNVLKFSALALAAVLATGC
      --------------MKLTTHHLRTGAALLLAGILLAGC
      -------------MAYSVQKSRLAKVAGVSLVLLLAAC
      ------------MSAGSPKFTVRRIAALSLVSLWLAGC
      MDKGEGLRLAATLRQWTRLYGGCHLLLGAVVCSLLAAC
      -------------------MKPFLRWCFVATALTLAGC
      -----------------MNIATKLMASLVASVVLTAC
      ----------------MQNAKLMLTCLAFAGLAALAGC
      --------------------MKKYLLGIGLILALIAC
      ----------------------MRLLIGFALALALIGC
      --------------MFVTSKKMTAAVLAITLAMSLSAC
```

```
----------------MNKNMAGILSAAAVLTMLAGC
----------------MHVSSLKVVLFGVCCLSLAAC
---------------MYKNGFFKNYLSLFLIFLVIAC
-----------------MNKFVKSLLVAGSVAALAAC
------------------MKKTNMALALLVAFSVTGC
---------------MSLTHYSGLAAAVSMSLILTAC
-----------------MLRYTRNALVLGSLVLLSGC
-------------------MRNFILFPMMAVVLLSGC
-------------------MRKQWLGICIAAGMLAAC
------------------MRYLATLLLSLAVLITAGC
-----------------MNMTKGALILSLSFLLAAC
---------------MNKKIFTLFLVVAASAIFAVSC
-------------------MVKRGRFALCLAVLLGAC
-----------------MKVKYALLSAGALQLLVVGC
----------------MNNPLVNQAAMVLPVFLLSAC
---------------MNAHTLVYSGVALACAAMLGSC
---------------MKLKSLVFSLSALFLVLGFTGC
----------------MREKWVRAFAGVFCAMLLIGC
--------------MKHNVKLMAMTAVLSSVLVLSGC
-------------------MKLRLSALALGTTLLVGC
-----------MRKRISAIINKLNISIIIMTVVLMIGC
-----------------MRKRISAIIMTLFMVLVSC
-----------MRKRISAIINKLNISIMMMIVVLMIGC
```

Το αποτέλεσμα αυτό, τυπώνεται στην οθόνη, αλλά θα μπορούσαμε πολύ εύκολα να έχουμε τροποποιήσει το πρόγραμμα έτσι ώστε να γράφει σε αρχείο (με την εντολή open). Εναλλακτικά, μπορούμε να εκμεταλλευτούμε τις δυνατότητες του shell (ακόμα και των Windows) και να ανακατευθύνουμε το αποτέλεσμα σε ένα αρχείο της επιλογής μας. Για παράδειγμα, μπορούμε να εκτελέσουμε το πρόγραμμα με την εντολή:

```
perl program.pl input.file > output.file
```

Γενικότερα στο Linux (και στο UNIX) το shell επιτρέπει πιο σύνθετους τρόπους διασύνδεσης μεταξύ προγραμμάτων, τις λεγόμενες "σωληνώσεις" (pipes). Με τις σωληνώσεις, επιτρέπουμε σε μια διεργασία να επικοινωνεί με μια άλλη και το αποτέλεσμα της μίας να αποτελεί είσοδο στην επόμενη. Για παράδειγμα η εντολή:

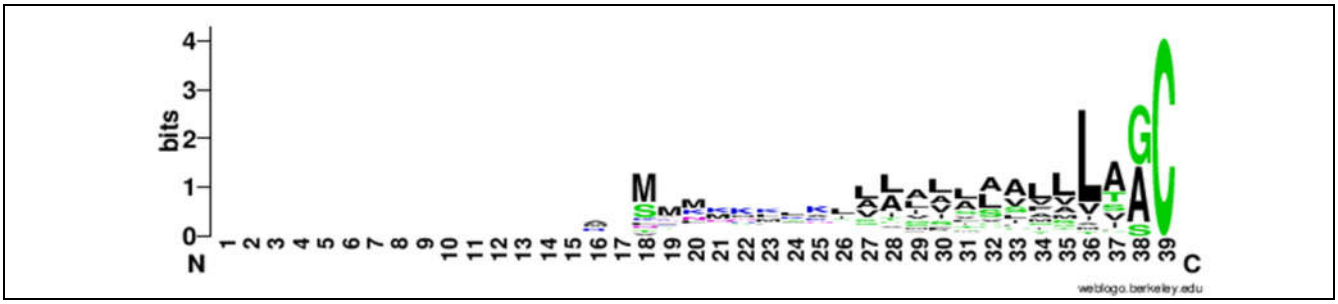
```
ls | wc
```

αποτελεί μια σωλήνωση που επιτρέπει στο αποτέλεσμα της ls να είναι είσοδος στην εντολή wc. Γενικά η Perl έχει πολλούς τρόπους να ελέγχει τις σωληνώσεις, αλλά ο πιο απλός είναι μέσω του shell. Για παράδειγμα μπορούμε να δώσουμε την εντολή:

```
./program.pl input.file | ./program2.pl
```

Με τον παραπάνω τρόπο, τα προγράμματα εκτελούνται σαν εντολές του συστήματος και το ένα δίνει το αποτέλεσμα στο άλλο. Κάτι που πρέπει να προσέξουμε σε αυτόν τον τρόπο, είναι ότι το δεύτερο πρόγραμμα θα πρέπει να επιτρέπει την εισαγωγή δεδομένων μέσα από το Standard Input (δηλαδή με χρήση του <STDIN>). Φυσικά, υπάρχουν και άλλοι τρόποι να χειριστούμε το θέμα, είτε μέσα από τη γλώσσα προγραμματισμού, όσο και από το ίδιο το περιβάλλον του shell αλλά η παρουσίαση τους ξεφεύγει από τους στόχους του παρόντος εισαγωγικού κειμένου.

Σε κάθε περίπτωση, το αποτέλεσμα του παραπάνω προγράμματος (η ιδιότυπη αυτή πολλαπλή στοίχιση), αν δοθεί σαν είσοδος στο WebLogo, θα δώσει το αποτέλεσμα που φαίνεται στην Εικόνα 12.1.



Εικόνα 12.1: *Το λογότυπο αλληλουχιών για την πολλαπλή στοίχιση των σηματοδοτικών αλληλουχιών των βακτηριακών λιποπρωτεϊνών όπως προέκυψε από το WebLogo (http://weblogo.berkeley.edu/)*

12.9. Περαιτέρω μελέτη

Όπως ήδη είπαμε, αυτό το κεφάλαιο προσφέρει μια απλή και όσο το δυνατό πιο κατανοητή εισαγωγή στην Perl, με ταυτόχρονη επίδειξη των ιδιοτήτων της γλώσσας στην επίλυση μερικών απλών προβλημάτων, τα οποία είναι εμπνευσμένα από πρακτικά προβλήματα της βιοπληροφορικής. Κάποιος που επιθυμεί να εμβαθύνει περισσότερο στην κατανόηση της γλώσσας θα πρέπει να ανατρέξει σε περισσότερο κατατοπιστικά και αναλυτικά βιβλία, όπως το *Programming Perl*, το οποίο θεωρείται το εγχειρίδιο αναφοράς της γλώσσας (Wall & Schwartz, 1991), ή το *Learning Perl* το οποίο αποτελεί μια πιο βατή εισαγωγή, καλύτερη για αρχάριους προγραμματιστές (Schwartz & Phoenix, 2001), το οποίο υπάρχει και στα Ελληνικά. Φυσικά, υπάρχουν πάρα πολλά εξειδικευμένα βιβλία, άλλα ειδικά εστιασμένα σε εφαρμογές βιοπληροφορικής (Moorhouse & Barry, 2005; Tisdall, 2001, 2003), άλλα σε αλγόριθμους (Orwant, Hietaniemi, & Macdonald, 1999) και άλλα και σε εφαρμογές διαδικτύου (Castro, 2001). Φυσικά, υπάρχουν άπειρα tutorials αλλά και online ebooks που διατίθενται δωρεάν όπως το *Picking Up Perl*, το οποίο είναι διαθέσιμο στη διεύθυνση http://www.ebb.org/PickingUpPerl/ (Kuhn, 2002), αλλά και πολλά ακόμα που μπορούν να βρεθούν στη διεύθυνση https://www.perl.org/books/library.html και http://www.perlmonks.org/index.pl/Tutorials.

Δεν πρέπει ακόμα να παραλείψουμε να κάνουμε αναφορά στο πρόγραμμα της **BioPerl** (http://www.bioperl.org/wiki/Main_Page), το οποίο είναι ένα μεγάλο έργο που σκοπό έχει να προσφέρει ειδικά προγράμματα για αναλύσεις βιοπληροφορικής σε μορφή βιβλιοθηκών (modules) της Perl. Η BioPerl είναι ένα τεράστιο έργο, που έχει χρησιμοποιηθεί από εκατοντάδες χρήστες και περιέχει εντολές για σχεδόν κάθε είδους ανάλυση (Stajich et al., 2002). Η ιδιαιτερότητα της, είναι ότι χρησιμοποιεί αντικειμενοστραφή προγραμματισμό (object-oriented Perl), κάτι που ίσως δυσκολέψει τον αρχάριο χρήστη. Μια εισαγωγή στην BioPerl, με κάποια απλά παραδείγματα όμοια με αυτά που αντιμετωπίσαμε σε αυτό το κεφάλαιο, υπάρχει στη διεύθυνση: http://www.bioperl.org/wiki/HOWTO:Beginners.

Βιβλιογραφία

Castro, Elizabeth. (2001). Perl and CGI for the world wide web: Visual quickstart guide: Peachpit Press.

Crooks, G. E., Hon, G., Chandonia, J. M., & Brenner, S. E. (2004). WebLogo: a sequence logo generator. Genome Res, 14(6), 1188-1190. doi: 10.1101/gr.84900414/6/1188 [pii]

Juncker, A. S., Willenbrock, H., Von Heijne, G., Brunak, S., Nielsen, H., & Krogh, A. (2003). Prediction of lipoprotein signal peptides in Gram-negative bacteria. Protein Sci, 12(8), 1652-1662.

Kuhn, Bradley M. (2002). Picking Up Perl: B. Kuhn.

Moorhouse, Michael, & Barry, Paul. (2005). Bioinformatics biocomputing and Perl: an introduction to bioinformatics computing skills and practice: John Wiley & Sons.

Orwant, Jon, Hietaniemi, Jarkko, & Macdonald, John. (1999). Mastering algorithms with Perl: " O'Reilly Media, Inc.".

Schwartz, Randal L, & Phoenix, Tom. (2001). Learning perl: O'Reilly & Associates, Inc.

Stajich, Jason E, Block, David, Boulez, Kris, Brenner, Steven E, Chervitz, Stephen A, Dagdigian, Chris, . . . Lapp, Hilmar. (2002). The Bioperl toolkit: Perl modules for the life sciences. Genome research, 12(10), 1611-1618.

Sutcliffe, I. C., & Harrington, D. J. (2002). Pattern searches for the identification of putative lipoprotein genes in Gram-positive bacterial genomes. Microbiology, 148(Pt 7), 2065-2077.

Tisdall, James. (2001). Beginning Perl for bioinformatics: " O'Reilly Media, Inc.".

Tisdall, James. (2003). Mastering Perl for bioinformatics: " O'Reilly Media, Inc.".

Wall, Larry, & Schwartz, Randal L. (1991). Programming perl: O'Reilly & Associates Sebastopol, CA.

Ασκήσεις

1) Τροποποιήστε το πρόγραμμα της προσομοίωσης έτσι ώστε να παράγει αμινοξέα ή νουκλεοτίδια σύμφωνα με κάποια προκαθορισμένη σύσταση (δηλαδή, να μην είναι ισοπίθανα). Για παράδειγμα, στην περίπτωση των πρωτεϊνών, μπορείτε να χρησιμοποιήσετε τη σύσταση ολόκληρης της Uniprot για να πάρετε πιο ρεαλιστικά αποτελέσματα (θεωρούμε εδώ ότι τα αμινοξέα δίνονται με τη σειρά που είδαμε ήδη, A, C, κ.ο.κ.):

```
@counts = ('0.077', '0.016', '0.053', '0.065', '0.041', '0.069', '0.023', '0.059',
'0.059', '0.095', '0.024', '0.043', '0.049', '0.039', '0.052', '0.070', '0.055',
'0.066', '0.012', '0.031');
```

2) Χρησιμοποιήστε το πρόγραμμα της προσομοίωσης τυχαίων αλληλουχιών DNA για να ελέγξετε τα αποτελέσματα του κεφαλαίου 3 που αφορούν τις ροές. Κατασκευάστε μια μεγάλη σειρά (π.χ. 1000) τυχαίων αλληλουχιών με γνωστό μήκος, και εντοπίστε τη μέγιστη ροή από A που εμφανίζεται σε κάθε μία από αυτές. Έπειτα, κατασκευάστε το ιστόγραμμα συχνοτήτων για τις 1000 αλληλουχίες και επιβεβαιώστε το νόμο των Erdos και Renyi και την κατανομή των ακραίων τιμών. Επαναλάβετε το πείραμα για διαφορετικά μήκη αλληλουχιών (π.χ. 1000, 5000, 10000 κ.ο.κ.).

3) Κατασκευάστε ένα πρόγραμμα που θα πραγματοποιεί πρόγνωση της θέσης μεμβρανικών τμημάτων σε διαμεμβρανικές πρωτεΐνες και ελέξτε την αποτελεσματικότητα του σε μια ομάδα πρωτεϊνών από την Uniprot. Οι γνωστής δομής μεμβρανικές πρωτεΐνες, έχουν στο πεδίο FT το χαρακτηριστικό TRANSMEM το οποίο δείχνει την θέση των διαμεμβρανικών περιοχών. Στο πρώτο στάδιο, το πρόγραμμα θα πρέπει να διαβάζει την εγγραφή και θα πρέπει να τυπώνει μια ιδιαίτερη μορφή fasta oneline, στην οποία θα υπάρχει και σήμανση. Για παράδειγμα το αρχείο θα είναι κάπως έτσι:

```
ID   140U_DROME              Reviewed;         261 AA.
AC   P81928; Q9VFM8;
FT   CHAIN         1    261       RPII140-upstream gene protein.
FT                                /FTId=PRO_0000064352.
FT   TRANSMEM     67     87       Potential.
FT   TRANSMEM    131    151       Potential.
FT   TRANSMEM    183    203       Potential.
FT   CONFLICT     64     64       S -> F (in Ref. 1).
SQ   SEQUENCE   261 AA;  29182 MW;  5DB78CF6CFC4435A CRC64;
     MNFLWKGRRF LIAGILPTFE GAADEIVDKE NKTYKAFLAS KPPEETGLER LKQMFTIDEF
     GSISSELNSV YQAGFLGFLI GAIYGGVTQS RVAYMNFMEN NQATAFKSHF DAKKKLQDQF
     TVNFAKGGFK WGWRVGLFTT SYFGIITCMS VYRGKSSIYE YLAAGSITGS LYKVSLGLRG
     MAAGGIIGGF LGGVAGVTSL LLMKASGTSM EEVRYWQYKW RLDRDENIQQ AFKKLTEDEN
     PELFKAHDEK TSEHVSLDTI K
```

Το πρόγραμμα θα πρέπει να μετατρέπει τις αλληλουχίες στη μορφή:

```
>P81928
MNFLWKGRRFLIAGILPTFEGAADEIVDKENKTYKAFLASKPPEETGLERLKQMFTIDEFGSISSELNSVYQAGFLGFL
------------------------------------------------------------------------MMMMMMMMMMM
```

Για την πρόγνωση, θα στηριχθείτε στη μέθοδο των παραθύρων, με χρήση κάποιας κλίμακας υδροφοβικότητας, όπως των Kyte-Doolitle που δίνεται παρακάτω:

```
%hyd =('A' => 0.100,
       'C' => -1.420,
       'D' => 0.780,
       'E' => 0.830,
       'F' => -2.120,
       'G' => 0.330,
       'H' => -0.500,
       'I' => -1.130,
       'K' => 1.400,
       'L' => -1.180,
```

```
            'M' => -1.590,
            'N' => 0.480,
            'P' => 0.730,
            'Q' => 0.950,
            'R' => 1.910,
            'S' => 0.520,
            'T' => 0.070,
            'V' => -1.270,
            'W' => -0.510,
            'Y' => -0.210
            );
```

Θα πρέπει να πειραματιστείτε με τις πιθανές τιμές του παραθύρου που δίνουν το καλύτερο αποτέλεσμα, αλλά και να προσπαθήσετε να αναπαραστήσετε τα αποτελέσματα γραφικά (με χρήση χαρακτήρων όπως το *). Μπορείτε επίσης να χρησιμοποιήσετε αντί της κλίμακας υδροφοβικότητας, και έναν εμπειρικό πίνακα του σκορ, με τιμές από συχνότητες αμινοξέων, όπως λ.χ. ο Πίνακας 3.1 στο κεφάλαιο 3, και να συγκρίνετε τα αποτελέσματα.

4) Κατασκευάστε ένα πρόγραμμα που θα παράγει τυχαίες αλληλουχίες DNA και θα ελέγχει το μήκος των πρωτεϊνών που παράγονται από τα τυχαία ανοιχτά πλαίσια ανάγνωσης που θα εντοπιστούν. Θα πρέπει να συνδυάσετε το πρόγραμμα της προσομοίωσης και το πρόγραμμα εύρεσης γονιδίων. Θα πρέπει να δημιουργήσετε ένα μεγάλο αριθμό τυχαίων γονιδιωμάτων (π.χ. 1000), με μεγάλο μήκος όμως (>10.000), και να αποθηκεύσετε το μήκος των παραγόμενων πρωτεϊνών για να γίνει ένα ιστόγραμμα που θα μας δείξει την κατανομή τους. Θα πρέπει να πειραματιστείτε τόσο με το μήκος του γονιδιώματος, όσο και με τη σύσταση σε νουκλεοτίδια (βλ. και άσκηση 1).

5) Κατασκευάστε ένα πρόγραμμα το οποίο θα διαβάζει τις αλληλουχίες πρωτεϊνών από ένα αρχείο FASTA (με μία γραμμή), και θα τις κωδικοποιεί σε παράθυρα με μήκος που θα ορίζει ο χρήστης, χρησιμοποιώντας το sparse encoding.