Introduction to Perl Programming
– Control Structures, Input/Output

Beginning Perl, Chap 4,6

Example 1

```
#!/usr/bin/perl -w
use strict;

# version 1:
my @nt = ('A', 'C', 'G', 'T');
my $dna;
for (1 .. 100) {
    $dna .= $nt[rand 4];
}

# version 2:
my @nt = qw(A C G T);
my $dna;
for (1 .. 100) {
 $dna .= $nt[rand 4];
}
# version 3: most Perlish
my @nt = qw(A C G T);
my $dna;
for (1 .. 100) {
  $dna .= $nt[rand(length(@nt))];
}

print $dna,"\n";
```

```
# version 4: with compositional bias

my %comp = ( A => 0.20,
             C => 0.35,
             G => 0.25,
             T => 0.20
           );
my @nt = keys %comp;
my @bucket;

# fill the bucket to draw from:
for my $nt (@nt) {
    push @bucket,
         ($nt) x ($comp{$nt} * 100);
}

my $dna;
for (0 .. 99) {
  $dna .= $bucket[rand length(@bucket)];
}
```

Example 2

```
#!/usr/bin/perl -w
use strict;

# version 1;
# same as nt construction
my @aa = qw(A C D E F G H I
        K L M N P Q R S T V W Y);
my $protein;
for (1 .. 200) {
 $protein .= $aa[rand length(@aa)];
}

print $protein,"\n";
```

```
# version 2: very Perlish;

my @letters = ( A .. Z );

my %comp;
# hash slice:
@comp{@letters} = ( 1 / 20 ) x @letters;
# remove letter that aren't aa's
$comp{'B'} = $comp{'J'} =
    $comp{'O'} = $comp{'X'} =
        $comp{'U'} = $comp{'Z'} = 0;

my @bucket;
for my $l (@letters) {
    push @bucket ($l) x ($comp{$l} * 100);
}

my $length = 200 + rand(100);
my $protein;
for (1 .. $length) {
    push $protein, $bucket[rand @bucket];
}
```

Example 3

```
#!/usr/bin/perl -w
use strict;

my %codontable = (
 'AAA' => 'K', 'AAC' => 'N', 'AAG' => 'K', 'AAT' => 'N',
 'ACA' => 'T', 'ACC' => 'T', 'ACG' => 'T', 'ACT' => 'T',
 'AGA' => 'R', 'AGC' => 'S', 'AGG' => 'R', 'AGT' => 'S',
 'ATA' => 'I', 'ATC' => 'I', 'ATG' => 'M', 'ATT' => 'I',
 'CAA' => 'Q', 'CAC' => 'H', 'CAG' => 'Q', 'CAT' => 'H',
 'CCA' => 'P', 'CCC' => 'P', 'CCG' => 'P', 'CCT' => 'P',
 'CGA' => 'R', 'CGC' => 'R', 'CGG' => 'R', 'CGT' => 'R',
 'CTA' => 'L', 'CTC' => 'L', 'CTG' => 'L', 'CTT' => 'L',
 'GAA' => 'E', 'GAC' => 'D', 'GAG' => 'E', 'GAT' => 'D',
 'GCA' => 'A', 'GCC' => 'A', 'GCG' => 'A', 'GCT' => 'A',
 'GGA' => 'G', 'GGC' => 'G', 'GGG' => 'G', 'GGT' => 'G',
 'GTA' => 'V', 'GTC' => 'V', 'GTG' => 'V', 'GTT' => 'V',
 'TAA' => '*', 'TAC' => 'Y', 'TAG' => '*', 'TAT' => 'Y',
 'TCA' => 'S', 'TCC' => 'S', 'TCG' => 'S', 'TCT' => 'S',
 'TGA' => '*', 'TGC' => 'C', 'TGG' => 'W', 'TGT' => 'C',
 'TTA' => 'L', 'TTC' => 'F', 'TTG' => 'L', 'TTT' => 'F'
);

# make DNA the boring way:
my @nt = qw(A C G T);
my $dna;
my $length = 500 + rand(500);
for (1 .. $length) {
    $dna .= $nt[rand @nt];
}
# make DNA the boring way:
my @nt = qw(A C G T);
my $dna;
my $length = 500 + rand(500);
for (1 .. $length) {
    $dna .= $nt[rand @nt];
}
```

```
# OK, now translate:
my $protein;
my @seq = split('', $dna);

my $codon = '';
for my $i (0 .. $#seq) {
    $codon .= $seq[$i];
    if (($i + 1) % 3 == 0) {
        $protein .= $codontable{$codon};
        $codon = '';
    }
}
# another way, using "length()":
for my $nt (@seq) {
    $codon .= $nt;
    if (length($codon) == 3) {
        $protein .= $codontable{$codon};
        $codon = '';}
}
# yet another variation on the theme:
while (@seq) {
    $codon .= shift @seq;
    if (length $codon == 3) {
        $protein .= $codontable{$codon};
        $codon = '';
    }
}
# even better:
while (@seq) {
   $protein .= $codontable{ shift @seq . shift
@seq . shift @seq};
}
```

Control Structures

- if (logical_expr) {statement;}
  elsif (expr) { }
  else { }
- for ($i=0; $i<10; $i++) { }
  for $i (0 .. 10) { }
  for (@list) { } for my $element (@list) { }
- while () { }

Logical expressions

- `if ( $this eq “that”) { }`
  string compare
- `while ( $count < 10 ) { }`
  numeric compare
- `if ( $line =~ m/^>/ ) { }`
  string matching

Control Structures - “if” variations

- if ($a==$b) { statement1; }
  else { statement2; }
- unless ($a >= $b) { statement;}
- statement1 if ($a == $b)

```
die (“usage - program file”)
   unless (defined $ARGV[0]);
```

Control structures – loops

-
```
while ( $line = <FILE>) {
    if ($line =~ m/^>/) {
#   have a description header
    last;
    }
    elsif ($line =~ m/^;/) { next;}
}
# where “last” goes
```
-
```
until ( ) { statement; }
```

Loops and labels (no goto's)

```
$i = $j = 0;
loop1:
  while ($i < 10 ) {
loop2:
     while ($j < 20 ) {
          if ($j % 5 == 4 ) { next loop2;}
          if ($j % 10 == 9) { next loop1;}
     }
  }
```

Input/Output

- `open (INFILE, "< gtm1_human.aa");` – open file for reading
- `$file = $ARGV[0];`
  `open (INFILE, "< $file")||`
  `die ("Cannot open $file");`
- `open (OUTFILE, "> $file);` – open file for writing
- `$line = <FILE>;` – read a line
- `chomp($line);` – remove "\n"
- `while ($line=<FILE>) { … }`
- `print FILE "$line\n";`
- `close(FILE);`

Exercises

1. Take a set of FASTA format files and make a FASTA library, combining those files
2. Take a FASTA library, and break it into individual FASTA files
3. Write a perl equivalent to "`mrtrans`" that does not care about the order of sequence input. Given two files, a FASTA protein library (with '-' for gaps) and a FASTA DNA library with the same sequence names, generate a FASTA DNA alignment with '---' inserted into the DNA sequence for each '-' in the protein sequence