Case Study 1.1 Translation of a DNA sequence to an amino acid sequence using the standard genetic code

```
#!/usr/bin/perl
#translate.pl -- translate nucleic acid sequence to protein sequence
#                  according to standard genetic code

#  set up table of standard genetic code
 "ttt"=> "Phe",   "tct"=> "Ser", "tat"=> "Tyr",   "tgt"=> "Cys",
 "ttc"=> "Phe",   "tcc"=> "Ser", "tac"=> "Tyr",   "tgc"=> "Cys",
 "tta"=> "Leu",   "tca"=> "Ser", "taa"=> "TER",   "tga"=> "TER",
 "ttg"=> "Leu",   "tcg"=> "Ser", "tag"=> "TER",   "tgg"=> "Trp",
 "ctt"=> "Leu",   "cct"=> "Pro", "cat"=> "His",   "cgt"=> "Arg",
 "ctc"=> "Leu",   "ccc"=> "Pro", "cac"=> "His",   "cgc"=> "Arg",
 "cta"=> "Leu",   "cca"=> "Pro", "caa"=> "Gln",   "cga"=> "Arg",
 "ctg"=> "Leu",   "ccg"=> "Pro", "cag"=> "Gln",   "cgg"=> "Arg",
 "att"=> "Ile",   "act"=> "Thr", "aat"=> "Asn",   "agt"=> "Ser",
 "atc"=> "Ile",   "acc"=> "Thr", "aac"=> "Asn",   "agc"=> "Ser",
 "ata"=> "Ile",   "aca"=> "Thr", "aaa"=> "Lys",   "aga"=> "Arg",
 "atg"=> "Met",   "acg"=> "Thr", "aag"=> "Lys",   "agg"=> "Arg",
 "gtt"=> "Val",   "gct"=> "Ala", "gat"=> "Asp",   "ggt"=> "Gly",
 "gtc"=> "Val",   "gcc"=> "Ala", "gac"=> "Asp",   "ggc"=> "Gly",
 "gta"=> "Val",   "gca"=> "Ala", "gaa"=> "Glu",   "gga"=> "Gly",
 "gtg"=> "Val",   "gcg"=> "Ala", "gag"=> "Glu",   "ggg"=> "Gly"
);

#  process input data

while ($line = <DATA>) {                                    # read in line of input
     print "$line";                                         # transcribe to output
     chop();                                                # remove end-of-line character
     @triplets = unpack("a3" x (length($line)/3), $line); # pull out successive triplets
     foreach $codon (@triplets) {                           # loop over triplets
       print "$standardgeneticcode{$codon}";           # print out translation of each
     }                                                      # end loop on triplets
     print "\n\n";                                          # skip line on output
}                                                           # end loop on input lines}
#    what follows is input data
__END__
atgcatccctttaat
tctgtctga
```

Case Study 1.2 Assembly of overlapping fragments

```
#!/usr/bin/perl
#assemble.pl -- assemble overlapping fragments of strings

# input of fragments
while ($line = <DATA>) {                  # read in fragments, 1 per line
  chop($line);                            # remove trailing carriage return
  push(@fragments,$line);                 # copy each fragment into array
}
# now array @fragments contains fragments

# we need two relationships between fragments:
# (1) which fragment shares no prefix with suffix of another fragment
#     * This tells us which fragment comes first
# (2) which fragment shares longest suffix with a prefix of another
#     * This tells us which fragment follows any fragment

# First set array of prefixes to the default value "noprefixfound".
#     Later, change this default value when a prefix is found.
#     The one fragment that retains the default value must be come first.

# Then loop over pairs of fragments to determine maximal overlap.
#     This determines successor of each fragment
#     Note in passing that if a fragment has a successor then the
#         successor must have a prefix

foreach $i (@fragments) {                 # initially set prefix of each fragment
    $prefix{$i} = "noprefixfound";        # to "noprefixfound"
    }                                     # this will be overwritten when a prefix is found

#  for each pair, find longest overlap of suffix of one with prefix of the other
#         This tells us which fragment FOLLOWS any fragment

foreach $i (@fragments) {                 # loop over fragments
    $longestsuffix = "";                  # initialize longest suffix to null

    foreach $j (@fragments) {             # loop over fragment pairs
        unless ($i eq $j) {               # don't check fragment against itself
            $combine = $i . "XXX" . $j;   # concatenate fragments, with fence XXX
            $combine =~ /([\S ]{2,})XXX\1/;   #   check for repeated sequence
            if (length($1) > length($longestsuffix)) {       # keep longest overlap
                $longestsuffix = $1;      # retain longest suffix
                $successor{$i} = $j;      # record that $j follows $i
            }
        }
    }
    $prefix{$successor{$i}} = "found";    # if $j follows $i then $j must have a prefix
}
foreach (@fragments) {                    # find fragment that has no prefix; that's the start
     if ($prefix{$_} eq "noprefixfound") {$outstring = $_;}
}
$test = $outstring;                       # start with fragment without prefix
while ($successor{$test}) {               # append fragments in order
     $test = $successor{$test};           # choose next fragment
     $outstring = $outstring . "XXX" . $test;  # append to string
     $outstring =~ s/([\S ]+)XXX\1/\1/;   # remove overlapping segment
}
$outstring =~ s/\\n/\n/g;                 # change signal \n to real carriage return
print "$outstring\n";                     # print final result

__END__
```

```
the men and women merely players;\n
one man in his time
All the world's
their entrances,$\backslash $nand one man
stage,\nAnd all the men and women
They have their exits and their entrances,\n
world's a stage,\nAnd all
their entrances,\nand one man
in his time plays many parts.
merely players;\nThey have
```

Problem 1.7. An alternative, concise, version of the program to assemble overlapping fragments (see Case Study 1.2)

```
#!/usr/bin/perl

$/ = "";
@fragments = split("\n",<DATA>);

foreach (@fragments) { $firstfragment{$_} = $_; }

foreach $i (@fragments) {
    foreach $j (@fragments) { unless ($i eq $j) {
        ($combine = $i . "XXX" . $j) =~ /([\S ]{2,})XXX\1/;
        (length($1) <= length($successor{$i})) || { $successor{$i} = $j };
    }                          }
    undef $firstfragment{$successor{$i}};
}

$test = $outstring = join "", values(%firstfragment);
while ($test = $successor{$test}) { ($outstring .= "XXX" . $test) =~ s/([\S ]+)XXX\1/\1/; }

$outstring =~ s/\\n/\n/g; print "$outstring\n";

__END__
the men and women merely players;\n
one man in his time
All the world's
their entrances,\nand one man
stage,\nAnd all the men and women
They have their exits and their entrances,\n
world's a stage,\nAnd all
their entrances,\nand one man
in his time plays many parts.
merely players;\nThey have
```

Box 4.1 A PERL program to draw dotplots

```
#!/usr/bin/perl -w
#dotplot.pl -- reads two sequences and prints dotplot

#format of input data: (if any input line contains a #, the program
#                       will ignore the # and everything to the right)
#    line 1   title of dotplot
#    line 2   two integers specifying window and threshold
#    line 3   title of first sequence
#    lines 4 through a line ending in *    first sequence
#    next line   title of second sequence
#    following lines, until a line ending in *     second sequence

# read input

$/ = "";                              # set switch to read entire paragraph
$_ = <DATA>; $_ =~ s/#(.*)\n/\n/g; # kill comments (introduced by #)
                                      #      at right of each line

# split input into lines and parse them
$_ =~ /^(.*)\n\s*(\d+)\s+(\d+)\s*\n(.*)\n([A-Z\n]*)\*\s*\n(.*)\n([A-Z\n]*)\*/;

#extract title, window size and threshold
$title = $1; $nwind = $2; $thresh = $3;

#extract sequences
$seqt1 = $4; $seq1 = $5; $seqt2 = $6; $seq2 = $7;

#collect sequences by deleting end-of-line characters
$seq1 =~ s/\n//g; $seq2 =~ s/\n//g; $n = length($seq1); $m = length($seq2);

# print postscript header

print <<EOF;

/s /stroke load def /l /lineto load def /m /moveto load def
/r /rlineto load def /n /newpath load def /c /closepath load def
/f /fill load def
1.75 setlinewidth 30 30 translate /Helvetica findfont 16 scalefont setfont
EOF

# print matrix

$dx = 500.0/$n;  $mdx = -$dx; $dy = 500.0/$m;
if ($dy < $dx) {$dx = $dy;} $dy = $dx; $xmx = $n*$dx; $ymx = $m*$dx;

print "3 510 m ($title   window = $nwind   threshold = $thresh) show\n";
printf "0 0 m 0 %9.2f l %9.2f %9.2f l %9.2f 0 l c s\n", $ymx,$xmx,$ymx,$xmx;

# loop through both sequences recording runs of identical characters

for ($k = $nwind - $m + 1; $k < $n - $nwind; $k++) {
   $i = $k; $j = 1; if ($k < 1) {$i = 1; $j = 2 - $k;}

   while ($i <= $n - $nwind && $j <= $m - $nwind) {
       $_ = (substr($seq1,$i -1,$nwind) ^ substr($seq2,$j -1,$nwind));
       $mismatch = ($_ =~ s/[^\x0]//g);
       if ($mismatch < $thresh) {
           $xl = ($i - 1)*$dx; $yb = ($m - $j)*$dy;
           printf "n %9.2f %9.2f m %9.2f 0 r 0 %9.2f r %9.2f 0 r c f\n",
                   $xl,$yb,$dx,$dy,$mdx;
```

```
        }
        $i++; $j++;
    }
}

print "showpage\n";                    # end postscript file

# material following __END__ is the input data to the program


__END__
ATPases lamprey / dogfish                    #TITLE
15 6                                         #WINDOW, THRESHOLD
Petromyzon marinus mitochondrion             #SEQUENCE 1
ATGACACTAGATATCTTTGACCAATTTACCTCCCCAACA
ATATTTGGGCTTCCACTAGCCTGATTAGCTATACTAGCCCCTAGCTTA
ATATTAGTTTCACAAACACCAAAATTTATCAAATCTCGTTATCACACACTA
CTTACACCCATCTTAACATCTATTGCCAAACAACTCTTTCTTCCAATAAAC
CAACAAGGGCATAAATGAGCCTTAATTTGTATAGCCTCTATAATATTTATC
TTAATAATTAATCTTTTAGGATTATTACCATATACTTATACACCAACTACC
CAATTATCAATAAACATAGGATTAGCAGTGCCACTATGACTAGCTACTGTC
CTCATTGGGTTACAAAAAAAACCAACAGAAGCCCTAGCCCACTTATTACCA
GAAGGTACCCCAGCAGCACTCATTCCCATATTAATTATCATTGAAACTATT
AGTCTTTTTATCCGACCTATCGCCCTAGGAGTCCGACTAACCGCTAATTTA
ACAGCTGGTCACTTACTTATACAACTAGTTTCTATAACAACCTTTGTAATA
ATTCCTGTCATTTCAATTTCAATTATTACCTCACTACTTCTTCTATTA
CTAACAATTCTGGAGTTAGCTGTTGCTGTAATCCAGGCATATGTATTTATT
CTACTTTTAACTCTTTATCTGCAAGAAAACGTTT*
Scyliorhinus canicula mitochondrion          #SEQUENCE 2
ATGATTATAAGCTTTTTTGATCAATTCCTAAGTCCCTCCTTTCTAGGA
ATCCCACTAATTGCCCTAGCTATTTCAATTCCATGATTAATATTTCCAACACCAACC
AATCGTTGACTTAATAATCGATTATTAACTCTTCAAGCATGATTTATTAACCGATTTATT
TATCAACTAATACAACCCATAAATTTAGGAGGACATAAATGAGCTATCTTATTTACAGCC
CTAATATTATTTTTAATTACCATCAATCTTCTAGGTCTCCTTCCATATACTTTTACGCCT
ACAACTCAACTTTCTCTTAATATAGCCTTTGCCCTGCCCTTATGGCTTACAACTGTATTA
ATTGGTATATTTAATCAACCAACCATTGCCCTAGGGCACTTATTACCTGAAGGTACCCCA
ACCCCTTTAGTACCAGTACTAATCATTATCGAAACCATCAGTTTATTTATTCGACCATTA
GCCTTAGGAGTCCGATTAACAGCCAACTTAACAGCTGGACATCTCCTTATACAATTAATC
GCAACTGCGGCCTTTGTCCTTTTAACTATAATACCAACCGTGGCCTTACTAACCTCCCTA
GTCCTGTTCCTATTGACTATTTTAGAAGTGGCTGTAGCTATAATTCAAGCATACGTATTT
GTCCTTCTTTTAAGCTTATATCTACAAGAAAACGTATAA*
```

Box 5.4 A PERL program to draw binary trees

```
#!/usr/bin/perl
#drawtree.pl -- draws binary trees (root at top)
#usage:  echo '(A,((B,C),D),(E,F));' | drawtree.pl > output.ps

print <<EOF;


/n /newpath load def /m /moveto load def /l /lineto load def
/rm /rmoveto load def /rl /rlineto load def /s /stroke load def
1.0 setlinewidth 50 100 translate 2 2 scale
/Helvetica findfont 10 scalefont setfont
EOF

$tree = <>; chop($tree);
s/[,;]//g;                  # Simplify notation from full Newick
$_ = reverse($tree); s/[()]//g;

$x = 0; $y = 0;
while ($nd = chop()) {
    print "$x $y m ($nd) stringwidth pop -0.5 mul 0 rm ($nd) show\n";
    $xx{$nd} = $x; $x+=20; $yy{$nd} = 10;
}

while ($tree =~ s/\(?([A-Z])([A-Z])\)?/$1/) {
    print "n $xx{$1} $yy{$1} m\n";
    ($yy{$1} > $yy{$2}) || {$yy{$1} = $yy{$2}}; $yy{$1} += 20;
    print "$xx{$1} $yy{$1} l $xx{$2} $yy{$1} l $xx{$2} $yy{$2} l s\n";
    $xx{$1} = 0.5*($xx{$1} + $xx{$2});
}
print "n $xx{$tree} $yy{$tree} m 0 20 rl s showpage\n";

$rx = 2*$x + 30; $yt = 2*$yy{$tree} + 146;
print "%%BoundingBox: 40 95 $rx $yt\n";



\begin{center}
\textbf{Box \arabic{count15}.\arabic{count11}
A PERL program to draw helical wheels}
\end{center}

\begin{verbatim}
#!/usr/bin/perl
#helwheel.pl -- draw helical wheel
#usage: echo DVAGHGQDILIRLFKSH | helwheel.prl > output.ps
# or    echo 20DVAGHGQDILIRLFKSH | helwheel.prl > output.ps
#       the numerical prefix sets the first residue number

# The output of this program is in PostScript (TM),
#      a general-purpose graphical language

# The next section prints a header for the PostScript file
print <<EOF;
%!PS-Adobe-
297.5 421. translate 2 setlinewidth 1 setlinecap
/Helvetica findfont 20 scalefont setfont 0 0 moveto
EOF

# Define fonts to associate with each amino acid
$font{"G"} = "Helvetica";      $font{"A"} = "Helvetica";      $font{"S"} = "Helvetica";
```

```
$font{"T"} = "Helvetica";      $font{"C"} = "Helvetica";      $font{"V"} = "Helvetica";
$font{"I"} = "Helvetica";      $font{"L"} = "Helvetica";      $font{"F"} = "Helvetica";
$font{"Y"} = "Helvetica";      $font{"P"} = "Helvetica";      $font{"M"} = "Helvetica";
$font{"W"} = "Helvetica";      $font{"H"} = "Helvetica-Bold"; $font{"N"} = "Helvetica-Bold";
$font{"Q"} = "Helvetica-Bold"; $font{"D"} = "Helvetica-Bold"; $font{"E"} = "Helvetica-Bold";
$font{"K"} = "Helvetica-Bold"; $font{"R"} = "Helvetica-Bold";
$_= <>;                                       # read line of input
chop();$_ =~ s/\s//g;                         # remove terminal carriage return and blanks
if ($_ =~ s/^(\d+)//)                         # if input begins with integer
     {$resno = $1;}                           # extract it as initial residue number
else {$resno = 1}                             # if not, set initial residue number = 1
$radius = 50;                                 # initialize values for radius,
$x = 0; $y = -50; $theta = -90;               # x, y and angle theta
#  print light gray spiral arc as succession of line segments, 10 per residue
$npoints = 10*(length($_) - 1);
print "0.8 0.8 0.8 setrgbcolor\n";           # set colour to light gray
print "newpath\n";                            # draw spiral arc
printf("%8.3f %8.3f moveto\n",$x,$y);

foreach  $d (1 .. $npoints) {                 # 10 points per residue
    $theta += 10; $radius += 0.6;             # increase radius and theta
    $x = $radius*cos($theta*0.01747737);      # calculate new value of x
    $y = $radius*sin($theta*0.01747737);      #   and y
    printf("%8.3f %8.3f lineto\n",$x,$y);
}
print "stroke\n";

# print residues and residue numbers

$radius = 50;                                 # reinitialize values for radius,
$x = 0; $y = -50; $theta = -90;               # x, y and angle theta
print "0 setgray\n";                          # set colour to black

foreach (split ("",$_)) {                     # loop over characters from input line
    print "/$font{$_}  findfont ";            # set font appropriate
    print "20 scalefont setfont\n";           # for this amino acid
    printf("%8.3f %8.3f moveto\n",$x,$y);     # move to current point
    print " ($resno$_) stringwidth";          # adjust position to centre residue
    print " pop -0.5 mul -7 rmoveto\n";       #    identification on point on spiral
    print " ($resno$_) show\n";               # print residue number and id
    print "% $theta $resno$_\n";
    $theta += 100; $radius += 6;              # set new values of angle, radius
    $x = $radius*cos($theta*0.01747737);      # compute new values of x
    $y = $radius*sin($theta*0.01747737);      #    and y
    $resno++;                                 # increase residue number
}
print "showpage\n";                           # postscript signals to
print "%%BoundingBox:";                       # print
$xl = 297.5 - 1.05*$radius;                   # x
$xr = 297.5 + 1.05*$radius;                   #  and
$yb = 421.  - 1.05*$radius;                   #   y
$yt = 421.  + 1.05*$radius;                   #    limits
printf("%8.3f %8.3f %8.3f %8.3f\n",  $xl,$xr,$yb,$yt);

print "%%EOF\n";                              # and wind up
```