Genomic Perl

From Bioinformatics Basics to Working Code

REX A. DWYER
Genomic Perl Consultancy, Inc.

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published 2002

Printed in the United States of America

Typeface Times 10/13 pt. *System* AMS-TEX [FH]

A catalog record for this book is available from the British Library.

Library of Congress Cataloging in Publication data available

ISBN 0 521 80177 X hardback

Contents

CHAPTER ONE

The Central Dogma

1.1 DNA and RNA

Each of us has observed physical and other similarities among members of human families. While some of these similarities are due to the common environment these families share, others are *inherited*, that is, passed on from parent to child as part of the reproductive process. Traits such as eye color and blood type and certain diseases such as red–green color blindness and Huntington's disease are among those known to be heritable. In humans and all other nonviral organisms, heritable traits are encoded and passed on in the form of *deoxyribonucleic acid*, or DNA for short. The DNA encoding a single trait is often referred to as a *gene*.[1] Most human DNA encodes not traits that distinguish one human from another but rather traits we have in common with all other members of the human family. Although I do not share my adopted children's beautiful brown eyes and black hair, we do share more than 99.9% of our DNA. Speaking less sentimentally, all three of us share perhaps 95% of our DNA with the chimpanzees.

DNA consists of long chains of molecules of the modified sugar deoxyribose, to which are joined the *nucleotides* adenine, cytosine, guanine, and thymine. The scientific significance of these names is minimal – guanine, for example, is named after the bird guano from which it was first isolated – and we will normally refer to these nucleotides or *bases* by the letters `A`, `C`, `G`, and `T`. For computational purposes, a strand of DNA can be represented by a string of `A`s, `C`s, `G`s, and `T`s.

Adenine and guanine are *purines* and share a similar double-ring molecular structure. Cytosine and thymine are *pyrimidines* with a smaller single-ring structure. Deoxyribose has five carbons. The conventions of organic chemistry assign numbers to the carbon atoms of organic molecules. In DNA, the carbon atoms of the nucleotides are numbered 1–9 or 1–6, while those of the sugar are numbered 1′ ("one prime"), 2′, 3′, 4′, and 5′. As it happens, the long chains of sugar molecules in DNA are formed

[1] We will refine this definition later.

by joining the 3′ carbon of one sugar to the 5′ carbon of the next by a *phosphodiester bond*. The end of the DNA chain with the unbound 5′ carbon is referred to as the *5′ end*; the other end is the *3′ end*. For our purposes, it is enough to know two things: that single DNA strands have an orientation, since two 3′ ends (or two 5′ ends) cannot be joined by a phosphodiester bond; and that strings representing DNA are almost always written beginning at the 5′ end.

Ribonucleic acid, or RNA, is similar to DNA, but the sugar "backbone" consists of ribose rather than deoxyribose, and uracil (`U`) appears instead of thymine. In a few simple organisms, such as HIV,[2] RNA substitutes for DNA as the medium for transmitting genetic information to new generations. In most, however, the main function of RNA is to mediate the production of proteins according to the instructions stored in DNA.

As its name suggests, deoxyribose can be formed by removing an oxygen atom from ribose. Although RNA is itself an accomplished molecular contortionist, a chain or *polymer* made of deoxyribose can assume a peculiar coiled shape. Furthermore, pairs composed of adenine and thymine joined by *hydrogen bonds* and similarly joined pairs of cytosine and guanine have similar shapes; `(A,T)` and `(C,G)` are said to be *complementary* base pairs (see Figure 1.1). Taken together, these two characteristics allow DNA to assume the famous *double helix* form, in which two arbitrarily long strands of complementary DNA base pairs entwine to form a very stable molecular spiral staircase.[3] Each end of a double helix has the 3′ end of one strand and the 5′ end of the other. This means that two strands are *complementary* if one strand can be formed from the other by substituting `A` for `T`, `T` for `A`, `C` for `G`, and `G` for `C` – and then reversing the result. For example, `ATTCCTCCA`[4] and `TGGAGGAAT` are complementary:

```
5'-ATTCCTCCA-3'
3'-TAAGGAGGT-5'
```

The double helix was first revealed by the efforts of Watson and Crick; for this reason, complementary base pairs are sometimes referred to *Watson–Crick pairs.* In fact, the names "Watson" and "Crick" are sometimes used to refer to a strand of DNA and its complement.

1.2 Chromosomes

Each cell's DNA is organized into *chromosomes*, though that organization differs tremendously from species to species.

[2] Human immunodeficiency virus, the cause of AIDS.

[3] A spiral staircase is, in fact, no spiral at all. A spiral is defined in cylindrical coordinates by variations of the equations $z = 0$; $r = \theta$. The equations $z = \theta$; $r = 1$ define a *helix*.

[4] This sequence, known as the *Shine–Dalgarno sequence*, plays an important role in the initiation of translation in the bacterium *E. coli*.

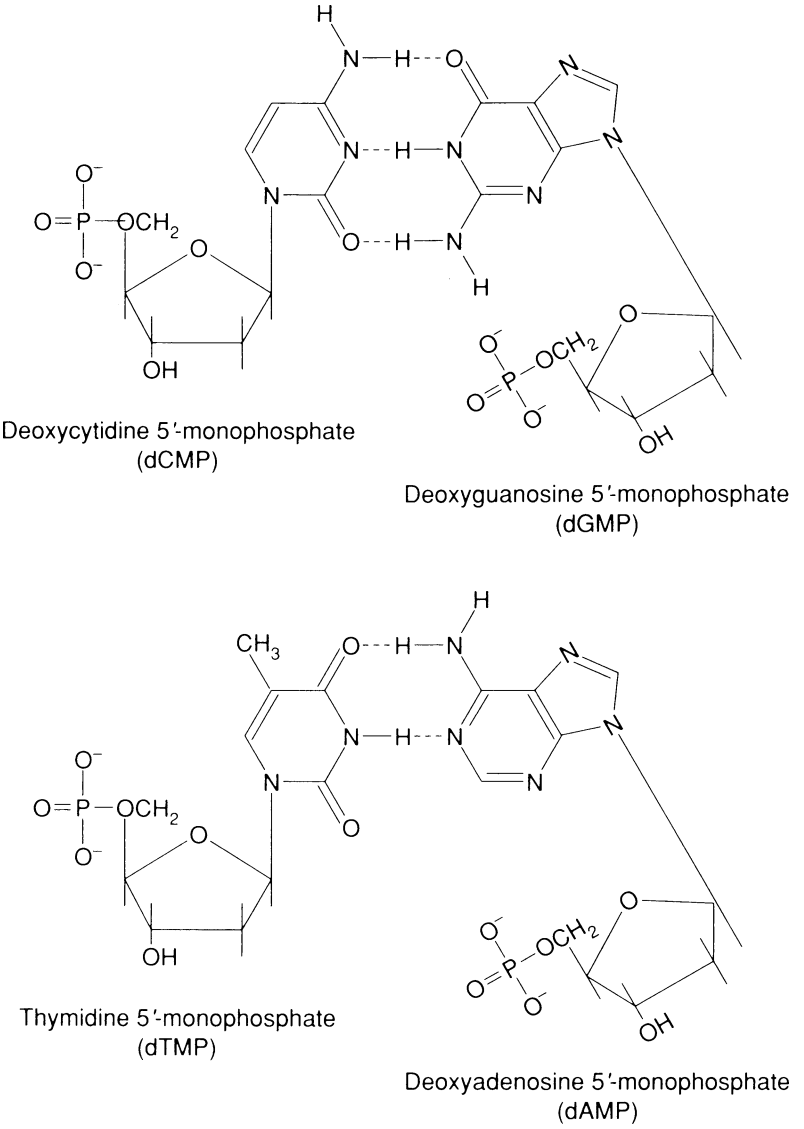

Figure 1.1: The nucleotides C and G (above) and A and T (below), showing how they can form hydrogen bonds (dotted lines). (Reproduced from Hawkins 1996.)

Human cells have 24 distinct types of chromosomes, with a total of about three billion (3×10^9) base pairs of DNA.[5] Among these, the *autosomes* are numbered 1–22 from largest to smallest, and the *sex chromosomes* are named X and Y. Each cell contains a pair of each autosome and either two X chromosomes (females) or one

[5] If denatured and stretched out, the DNA in each cell's nucleus would be about one yard (94 cm) long.

X and one Y chromosome (males). Egg and sperm cells, collectively known as *germ cells*, are exceptions to this rule; each contains only one of each autosome and one sex chromosome. Taken together, the 24 types of human chromosome constitute the human *genome.*

The pairs of autosomes in a cell should not be confused with the double-stranded nature of DNA. Each Chromosome 1 is double-stranded. Furthermore, the two Chromosomes 1 are nearly identical but not completely so. Wherever one contains a gene received from the mother, the other contains a gene from the father. This state of affairs is called *diploidy* and is characteristic of species that can reproduce sexually.[6]

Multiple, linear chromosomes are characteristic of the cells of *eukaryotes*, organisms whose chromosomes are sequestered in the cell's *nucleus.*[7] However, not all eukaryotes are diploid. The bread mold *Neurospora crassa* is *haploid*, meaning that each cell has only a single copy of each of its seven types of chromosomee. Mold cells reproduce asexually by dividing.

Simpler organisms called *prokaryotes* lack a cell nucleus. The bacterium *Escherichia coli*, a well-studied inhabitant of the human bowel, has a single, circular chromosome with about 4.5 million base pairs. *Viruses* are simplest of all, consisting only of genetic material – RNA, or either single- or double-stranded DNA – in a container. Viruses cannot reproduce on their own. Instead, like molecular cuckoos, they co-opt the genetic machinery of other organisms to reproduce their kind by inserting their genetic material into their host's cells. The genetic material of the virus ΦX174, which infects *E. coli*, consists of only 5386 bases in a single-stranded ring of DNA.[8]

1.3 Proteins

Like DNA and RNA, proteins are polymers constructed of a small number of distinct kinds of "beads" known as *peptides, amino acids, residues,* or – most accurately but least commonly – *amino acid residues.* Proteins, too, are oriented, and they are normally written down from the *N-terminus* to the *C-terminus.* The names of the 20 "natural"[9] amino acids, together with common three- and one-letter abbreviations, are noted in Figure 1.2.

Some proteins give organisms their physical structure; good examples are the keratins forming hair and feathers and the collagen and elastin of ligaments and tendons. Others, much greater in variety if lesser in mass, catalyze the many chemical reactions required to sustain life. Protein catalysts are called *enzymes,* and their names can be recognized by the suffix *-ase.* Proteins do not assume a predictable, uniform

[6] Not all diploid species have distinct sex chromosomes, however.
[7] Greek *karyos* is equivalent to Latin *nucleus*; *eu-* means "good, complete".
[8] Viruses that infect bacteria are also called *bacteriophages*, or simply *phages.*
[9] Selenocysteine, abbreviated by `U`, has been recently recognized as a rare 21st naturally occurring amino acid. When occurring, it is encoded in RNA by `UGA`, which is normally a stop codon.

Alanine	A	Ala	Leucine	L	Leu
Arginine	R	Arg	Lysine	K	Lys
Asparagine	N	Asn	Methionine	M	Met
Aspartic acid	D	Asp	Phenylalanine	F	Phe
Cysteine	C	Cys	Proline	P	Pro
Glutamine	Q	Gln	Serine	S	Ser
Glutamic acid	E	Glu	Threonine	T	Thr
Glycine	G	Gly	Tryptophan	W	Trp
Histidine	H	His	Tyrosine	Y	Tyr
Isoleucine	I	Ile	Valine	V	Val

Figure 1.2: Amino acids and their abbreviations.

shape analogous to DNA's double helix. Instead, protein shapes are determined by complicated interactions among the various residues in the chain. A protein's shape and electrical charge distribution, in turn, determine its function.

Predicting the shape a given amino acid sequence will assume *in vivo*[10] – the *protein-folding problem* – is one of the most important and most difficult tasks of computational molecular biology. Unfortunately, its study lies beyond the scope of this book, owing to the extensive knowledge of chemistry it presupposes.

1.4 The Central Dogma

The Central Dogma of molecular biology relates DNA, RNA, and proteins. Briefly put, the Central Dogma makes the following claims.

- The amino acid sequence of a protein provides an adequate "blueprint" for the protein's production.
- Protein blueprints are encoded in DNA in the chromosomes. The encoded blueprint for a single protein is called a *gene.*
- A dividing cell passes on the blueprints to its daughter cells by making copies of its DNA in a process called *replication*.
- The blueprints are transmitted from the chromosomes to the protein factories in the cell in the form of RNA. The process of copying the DNA into RNA is called *transcription*.
- The RNA blueprints are read and used to assemble proteins from amino acids in a process known as *translation*.

We will look at each of these steps in a little more detail.

The Genetic Code. A series of experiments in the 1960s cracked the genetic code by synthesizing chains of amino acids from artificially constructed RNAs.

[10] "In life" – as opposed to *in vitro* or "in glass" (in the laboratory). The process of predicting the shape computationally is sometimes called protein folding *in silico.*

Amino acids are encoded by blocks of three nucleotides known as *codons.* There are $4 \times 4 \times 4 = 64$ possible codons, and (except for methionine and tryptophan) each amino acid is encoded by more than one codon, although each codon encodes only one amino acid. The end of a protein is signaled by any of three *stop codons.*

DNA comprises more than just codons for protein production. Along with *coding regions,* DNA contains *regulatory regions* such as *promoters, enhancers,* and *silencers* that help the machinery of protein production find its way to the coding regions often enough – but not too often – to keep the cell adequately supplied with the proteins it needs. DNA also encodes some RNAs that catalyze reactions rather than encoding proteins. Most mammalian DNA, however, has no known function and is often referred to as *junk DNA.* The computational process of sifting out probable coding and regulatory regions from what we presently call "junk" is called *gene prediction*.

Replication. The hydrogen bonds that join the complementtary pairs in DNA's double helix are much weaker than the covalent bonds between the atoms within each of its two strands. Under the right conditions, the two strands can be untwisted and separated without destroying the individual strands. A new complementary strand can be constructed on each of the old strands, giving two new double strands identical to the original.

Replication is accomplished with the assistance of two types of enzymes. *DNA helicases* untwist and separate the double helix, and *DNA polymerases* catalyze the addition of free nucleotides to the growing complementary strands. These enzymes work together at the *replication fork,* where the original double strand parts into two single strands.

Transcription. *RNA polymerase* catalyzes the production of RNA from DNA during transcription. The two strands of DNA are separated, and an RNA strand complementary to one of the DNA strands is constructed.

Transcription begins at a site determined by certain *promoter elements* located in the noncoding region at the $5'$ end of the coding region, the best-known of which is the "`TATA` box". How transcription terminates is not well understood in all cases.

In prokaryotes, translation is begun at the free end of the RNA while the other end is still being transcribed from the DNA. But in eukaryotes, the RNA (referred to as a *primary transcript*) is first subjected to a process in the nucleus called *splicing.* Splicing removes certain untranslatable parts of the RNA called *introns.* After splicing, the final *messenger RNA* (mRNA) passes from the nucleus to the cell's *cytoplasm*. Here mRNAs are translated, and many of the resulting proteins remain here to perform their functions.

Translation. Proteins are assembled by *ribosomes* that attach to RNA and advance three bases at a time, adding an amino acid to the protein chain at each step. Ribosomes consist of several proteins as well as small ribosomal RNA molecules (rRNAs) that fold like proteins and act as catalysts. Ribosomes are also assisted by so-called tRNAs.

Translation is initiated when one of the rRNAs in the ribosome binds to a particular sequence of about ten bases in the mRNA.[11] The first codon translated is always AUG (methionine). However, not every AUG codon marks the beginning of a coding region. Translation ends at the first stop codon encountered. A single mRNA molecule can be translated many times to make many identical protein molecules. However, mRNAs eventually degrade until translation is no longer possible.

1.5 Transcription and Translation in Perl

Now we develop a Perl program that can tell us what proteins a given DNA sequence can encode. There are two main steps.

1. Read in a table of codons and amino acids in text form, and construct a Perl data structure that allows codons to be translated quickly.
2. Read strands of DNA from input and write out the corresponding RNA and protein sequences.

We begin every program with

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

The first line tells the operating system where to find the Perl interpreter.[12] You should consult your system administrator to learn its precise location on your system. The second line tells the Perl system to do as much error checking as possible as it compiles.

Our first programming task is to create and fill a data structure that can be used to look up the amino acid residue corresponding to a single codon. The most convenient structure in Perl is the *hash table,* or just *hash* for short.[13] An empty hash is declared (i.e., brought into existence) by the statement

```
my %codonMap;
```

Once the hash is filled, we will be able, for example, to print Arg, the residue encoded by CGA, with the statement

[11] This is the Shine–Dalgarno sequence given previously. The exact sequence varies among species.

[12] Many sources suggest that this line should always be #!/usr/bin/perl -w, which asks the Perl system to issue warnings about constructs in the program that are not strictly errors but that appear suspect. Unfortunately, the -w "switch" is a little too suspicious, and I recommend it only as a debugging tool. Perl also offers a way to turn the warning feature on and off as the program progresses.

[13] The terms "associative array", "look-up table", and "dictionary" are roughly synonymous with "hash".

```
print $codonMap{"CGA"};
```

Here CGA is the *hash key* and Arg is the corresponding *value.* When we refer to the whole hash, the hash's name is preceded by a percent sign. When referring to an individual element, we precede the name by a dollar sign and follow it by braces.

Although we could fill the hash by listing every codon–residue pair explicitly in our program, we will use Perl's DATA feature to save some typing. This feature allows free-form text to be included at the end of the program file for reading and processing like any other text file during program execution. Our text will have one line for each of the 20 residues plus one for "Stop". Each line will include a three-letter residue abbreviation followed by each of the 1–6 corresponding codons in the genetic code. The first three of these 21 lines are:

```
Ala GCU GCC GCA GCG
Arg CGU CGC CGA CGG AGA AGG
Asn AAU AAC
```

Next we must write code to read in these lines and use them to fill the hash:

```
my $in;                                  ## 1
while ($in=<DATA>) {                     ## 2
    chomp($in);                          ## 3
    my @codons = split " ",$in;          ## 4
    my $residue = shift @codons;         ## 5
    foreach my $nnn (@codons) {          ## 6
        $codonMap{$nnn} = $residue;      ## 7
    }
}
```

Line 1 declares a *scalar* variable named $in. A scalar variable can hold a *single* value – an integer, a real number, a character string, Perl's special "undefined" value, or a *reference.* (We will discuss references in greater detail in later chapters.) The names of scalar variables always begin with the dollar sign. Unlike many familiar programming languages, Perl is not strongly typed. During the execution of a program, the *same* scalar variable can hold first the undefined value, later a string, and later a real number. The **my** keyword appears again in Lines 4, 5, and 6; these lines illustrate that a variable can be declared by adding **my** to its first use rather than in a separate statement.

The $<>$ notation on Line 2 causes one line to be read from input. Here, we use the *filehandle* DATA to tell the program to read our residue–codon table from the end of the program file. Each execution of Line 2 reads the next text line, turns it into a

Perl string, and stores it in $in; then, the execution of Lines 3–8 proceeds. With the data given above, the first execution of Line 2 has the same effect as the assignment

```
$in = "Ala GCU GCC GCA GCG\n";
```

(The two symbols \n appearing together in a string represent in visible form the single end-of-line or *newline* character that separates lines in a text file.)

If no more lines of text remain, then the "undefined" value (**undef**) is assigned to $in and the **while**-loop terminates. In general, Perl's **while**-loops terminate when the value of the parenthesized condition is undefined, the number 0, the string "0", or the empty string ""; any other value causes the loop to continue. (The nonempty strings "00" and " " do *not* terminate loops!) Since the value of an assignment is the same as the value on its right-hand side, this loop will terminate when the input at the end of the program is exhausted.

Line 3 uses Perl's built-in **chomp** operator to remove the newline character from the string stored in $in.

Line 4 uses the **split** operator to break the string in $in into a *list* of strings and then assigns the resulting list to the list variable @codons. The names of list variables begin with the "at" sign (@) when referring to the whole list. When processing the first line of our data, the effect is the same as

```
@codons = ("Ala","GCU","GCC","GCA","GCG");
```

The first operand of **split** is a *pattern* or *regular expression* describing the positions at which the second operand is to be split. The pattern here, " ", is just about the simplest possible; it matches at positions containing a blank-space character. A slightly more sophisticated pattern is /[AU]/, which matches positions containing either A or U. If we had written **split** /[AU]/, $in; then the result would have been the list of four strings ("", "la GC", " GCC GC", " GCG"). We will see other more complicated patterns in future chapters.

The **shift** operator in Line 5 removes the first item from the list @codons and assigns it to $residue, and it has the same effect as

```
$residue = "Ala";
@codons = ("GCU","GCC","GCA","GCG");
```

Lines 6 through 8 use the **foreach** construct to repeat the same group of statements on each item in the list @codons. To provide uniform access to the list items, the **foreach**-loop assigns each element of the list to the loop-variable $nnn before executing the body of the loop. The ultimate effect is the same as

```
$codonMap{"GCU"} = "Ala";
$codonMap{"GCC"} = "Ala";
$codonMap{"GCA"} = "Ala";
$codonMap{"GCG"} = "Ala";
```

If we repeat this process for every line of input, then it is clear we will fill the hash %codonMap as desired.

Having filled our hash, we can now read and process DNA strings entered by our user at the terminal:

```
while (my $dna=<STDIN>) {          ## 1
    chomp($dna);                   ## 2
    print "DNA: ", $dna, "\n";     ## 3
    my $rna = transcribe($dna);    ## 4
    print "RNA: ", $rna, "\n";     ## 5
    my $protein = translate($rna); ## 6
    print "RF1: ", $protein, "\n"; ## 7
    $rna =~ s/.//;                 ## 8
    $protein = translate($rna);    ## 9
    print "RF2: ", $protein, "\n"; ## 10
    $rna =~ s/.//;                 ## 11
    $protein = translate($rna);    ## 12
    print "RF3: ", $protein, "\n\n"; ## 13
}                                  ## 14
```

Lines 1 and 2 are similar to Lines 3 and 4 in the previous code fragment; they read a line from the terminal (STDIN), assign it to $dna, and remove the newline. Line 3 echoes the user's input. Line 4 calls the *subroutine* transcribe. This subroutine takes a string representing DNA as an *argument* and returns a string representing RNA as its *result*. In this case, the result is stored in the variable $rna. Of course, Perl doesn't have a built-in operator for transcribing DNA to RNA; we will write this subroutine ourselves shortly. Line 5 prints the RNA.

Line 6 calls the subroutine translate, which takes an RNA string as an argument and returns a string representing an amino acid sequence. Line 7 prints the sequence.

RNA is translated in blocks of three, and it is possible that the user's input begins or ends in the middle of one of these blocks. Therefore, we would like to print the amino acid sequence encoded by each of the three possible *reading frames.* To print the second reading frame, we delete the first base from the RNA and translate again. Line 8 accomplishes the deletion using Perl's pattern-matching operator =~. The string to be operated on is the one stored in $rna, and the operation will change the value of $rna. In this case, the operation is a substitution, signaled by **s**. Between the first two slashes is the pattern to be replaced, and between the second and third

slashes is the replacement. The dot . is a special pattern that matches any single character. The replacement is empty. Since substitution finds and replaces only the *first* occurrence of the pattern, this substitution deletes the first character of $rna. Translating this new string gives the second reading frame, and repeating the process once more gives the third.

Now we must write the subroutines transcribe and translate. We begin with transcribe:

```
sub transcribe {                          ## 1
    my ($dna) = @_;                       ## 2
    my $rna = scalar reverse $dna;        ## 3
    $rna =~ tr/ACGT/UGCA/;                ## 4
    return $rna;                          ## 5
}                                         ## 6
```

Line 1 begins the definition of the subroutine. Line 2 is Perl's peculiar way of relating formal parameters (the variable names used inside the subroutine definition) to actual arguments (the values passed to the subroutine when it is called). For the moment, it will suffice to say that (a) transcribe expects a single value and (b) that value is assigned to the local variable $dna when the subroutine is called.

Since transcription produces RNA that is complementary to the DNA, we must both reverse the string and change bases to their complements. Perl's **reverse** operator can be used both to reverse the order of the characters in a string and to reverse the order of the items in a list, depending on context. To avoid potential confusion, we use the keyword **scalar** to specify the first option explicitly. Line 3 does not change the value of $dna; instead, it creates a new string and assigns it to $rna.

Line 4 uses the pattern-matching operator. This time, the letters **tr** indicate a *transliteration*. Transliteration replaces *every* occurrence of a character between the first two slashes by the corresponding character between the second and third, so every A is replaced by U, every C by G, and so on.

Line 5 returns the RNA string to the main program as the value of the subroutine.

Finally, we come to our RNA-to-protein translation subroutine:

```
sub translate {                           ## 1
    my ($mrna) = @_;                      ## 2
    my $pro = "";                         ## 3
    while ($mrna =~ s/(...)//) {          ## 4
        $pro = $pro . $codonMap{$1};      ## 5
    }                                     ## 6
    return $pro;                          ## 7
}                                         ## 8
```

To translate RNA to protein, we break nucleotides off the RNA string three at a time and look these codons up in %codonMap. The results are collected in the variable $pro using ., the concatenation operator. Our substitution pattern has *three* dots, meaning that the pattern will match the first three bases of the RNA. The empty replacement will delete the three bases. However, since we have wrapped parentheses around the pattern, the portion of the string matched by the pattern will be saved in the special variable $1. We use the saved codon for look-up in %codonMap in Line 5.

The **while**-loop terminates as soon as the pattern match fails – that is, when fewer than three characters remain in $mrna.

If this program is saved in the file `cendog.pl`, we can execute it on a Unix system by moving to the directory containing the file and typing:[14]

```
./cendog.pl
```

Under either Unix or MS/DOS, the following command line works:

```
perl cendog.pl
```

Here is output from a sample execution of the program on the input `ACGGTC CTACCTTTA`, including original DNA, RNA transcript, and translations in reading frames 1, 2, and 3:

```
DNA: ACGGTCCTACCTTTA
RNA: UAAAGGUAGGACCGU
RF1: ...Arg...AspArg
RF2:  LysGlyArgThr
RF3:   LysValGlyPro
```

1.6 Exercise

1. One of Perl's mottos is "There's always more than one way to do it." Find at least three other ways to write Lines 4–6 of translate. *Hint:* Check out $&, .=, and **substr** in your favorite Perl reference.

1.7 Complete Program Listings

To illustrate the overall layout of a Perl program, we include a complete listing here as well as in file `chap1/cendog.pl` in the software distribution.

```
#!/usr/bin/perl
use strict;              ## request conservative error checking
```

[14] We must also be sure that the file is "executable", using the `chmod` command if not.

```
my %codonMap;      ## declare a hash table

## transcribe translates DNA strings to RNA
sub transcribe {
    my ($dna) = @_;
    my $rna = scalar reverse $dna;
    $rna =~ tr/ACGT/UGCA/;
    return $rna;
}

## translate translates mRNA strings to proteins
sub translate {
    my ($mrna) = @_;
    my $pro = "";
    while ($mrna =~ s/(...)//) {
        $pro = $pro . $codonMap{$1};
    }
    return $pro;
}

## construct hash that maps codons to amino acids by reading table
## from DATA at the end of the program, which have the following form:
## Residue Codon1 Codon2 ...

while (my $in=<DATA>) {      ## assigns next line of DATA to $in; fails if none
    chomp($in);              ## remove line feed at end of $in
    my @codons = split " ",$in;
    my $residue
        = shift @codons;     ## remove first item from @codons and assign
    foreach my $nnn (@codons) {
        $codonMap{$nnn} = $residue;
    }
}

## now read DNA strands from input <STDIN> and print translations in all six
## possible reading frames

while (my $dna=<STDIN>) {
    chomp($dna);
    print "DNA: ", $dna, "\n";
    my $rna = transcribe($dna);
    print "RNA: ", $rna, "\n";
    my $protein = translate($rna);
    print "RF1: ", $protein, "\n";
```

```
    $rna =~ s/.//;
    $protein = translate($rna);
    print "RF2: ", $protein, "\n";
    $rna =~ s/.//;
    $protein = translate($rna);
    print "RF3: ", $protein, "\n\n";
}

## The lines below are not Perl statements and are not executed as part of the
## program. Instead, they are available to be read as input by the program
## using the filehandle DATA.
__END__
Ala GCU GCC GCA GCG
Arg CGU CGC CGA CGG AGA AGG
Asn AAU AAC
Asp GAU GAC
Cys UGU UGC
Gln CAA CAG
Glu GAA GAG
Gly GGU GGC GGA GGG
His CAU CAC
Ile AUU AUC AUA
Leu UUA UUG CUU CUC CUA CUG
Lys AAA AAG
Met AUG
Phe UUU UUC
Pro CCU CCC CCA CCG
Ser UCU UCC UCA UCG AGU AGC
Thr ACU ACC ACA ACG
Trp UGG
Tyr UAU UAC
Val GUU GUC GUA GUG
... UAA UAG UGA
```

1.8 Bibliographic Notes

Two widely used but weighty introductory genetics textbooks are Lewin's (1999) and Snustad and Simmons's (1999). A lighter but still informative approach is taken in Gonick and Wheelis's *Cartoon Guide* (1991). Goodsell's (1996) pencil-drawn protein portraits are no less artistic and extremely effective in conveying both the wide range of functions performed by proteins and the relationship of their forms and functions.

A visit to the local chain book store will turn up a book on Perl for each of the Arabian Nights. Those in O'Reilly's series are among the most frequently seen on the bookshelves of Perl programmers. Vroman's (2000) slim pocket reference may provide supplement enough to readers who already program well but just not in Perl.[15] The "camel book" (Wall, Christiansen, and Orwant 1997) will be more appropriate for those less comfortable at the keyboard.[16] Srinivasan's (1997) more advanced book may be useful to both types of reader as an accompaniment to the more complex data structures of our later chapters.

A small sense of the history of genetic research can be gained from the following articles. Pelagra once ravaged the poor of the American South; its history (Akst 2000) provides one of many possible illustrations of the unfortunate social consequences of ignorance of the mechanisms of heredity and roles of heredity and environment in the propagation of disease. The discovery of the structure of DNA is famously documented by one of its most active participants in Watson's *Double Helix* (1998). A decade later, his senior partner wrote about the cracking of the genetic code (Crick 1966) for the educated general reader. Frenkel (1991) described the promise of bioinformatics around the time of the field's christening. The cataloging of the three billion nucleotides of the human genome (the collection of all 24 chromosomes) was recently completed; the announcement (Pennisi 2001) was accompanied by widespread discussion about the future importance of the data.

[15] It is an elaboration of a free version downloadable from `www.perl.com`.
[16] One of its authors, Larry Wall, is the inventor of Perl.