Genomic Perl

From Bioinformatics Basics to Working Code

REX A. DWYER
Genomic Perl Consultancy, Inc.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>xiii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>xvii</td>
</tr>
<tr>
<td>1 The Central Dogma</td>
<td>1</td>
</tr>
<tr>
<td>1.1 DNA and RNA</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Chromosomes</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Proteins</td>
<td>4</td>
</tr>
<tr>
<td>1.4 The Central Dogma</td>
<td>5</td>
</tr>
<tr>
<td>1.5 Transcription and Translation in Perl</td>
<td>7</td>
</tr>
<tr>
<td>1.6 Exercise</td>
<td>12</td>
</tr>
<tr>
<td>1.7 Complete Program Listings</td>
<td>12</td>
</tr>
<tr>
<td>1.8 Bibliographic Notes</td>
<td>14</td>
</tr>
<tr>
<td>2 RNA Secondary Structure</td>
<td>16</td>
</tr>
<tr>
<td>2.1 Messenger and Catalytic RNA</td>
<td>16</td>
</tr>
<tr>
<td>2.2 Levels of RNA Structure</td>
<td>17</td>
</tr>
<tr>
<td>2.3 Constraints on Secondary Structure</td>
<td>18</td>
</tr>
<tr>
<td>2.4 RNA Secondary Structures in Perl</td>
<td>20</td>
</tr>
<tr>
<td>2.4.1 Counting Hydrogen Bonds</td>
<td>21</td>
</tr>
<tr>
<td>2.4.2 Folding RNA</td>
<td>24</td>
</tr>
<tr>
<td>2.5 Exercises</td>
<td>28</td>
</tr>
<tr>
<td>2.6 Complete Program Listings</td>
<td>29</td>
</tr>
<tr>
<td>2.7 Bibliographic Notes</td>
<td>29</td>
</tr>
<tr>
<td>3 Comparing DNA Sequences</td>
<td>30</td>
</tr>
<tr>
<td>3.1 DNA Sequencing and Sequence Assembly</td>
<td>30</td>
</tr>
<tr>
<td>3.2 Alignments and Similarity</td>
<td>32</td>
</tr>
<tr>
<td>3.3 Alignment and Similarity in Perl</td>
<td>36</td>
</tr>
<tr>
<td>3.4 Exercises</td>
<td>40</td>
</tr>
<tr>
<td>3.5 Complete Program Listings</td>
<td>42</td>
</tr>
<tr>
<td>3.6 Bibliographic Notes</td>
<td>43</td>
</tr>
<tr>
<td>4 Predicting Species: Statistical Models</td>
<td>44</td>
</tr>
<tr>
<td>4.1 Perl Subroutine Libraries</td>
<td>49</td>
</tr>
<tr>
<td>4.2 Species Prediction in Perl</td>
<td>51</td>
</tr>
</tbody>
</table>
4.3 Exercises 53
4.4 Complete Program Listings 53
4.5 Bibliographic Note 54

5 Substitution Matrices for Amino Acids 55
5.1 More on Homology 57
5.2 Deriving Substitution Matrices from Alignments 57
5.3 Substitution Matrices in Perl 60
5.4 The PAM Matrices 65
5.5 PAM Matrices in Perl 68
5.6 Exercises 70
5.7 Complete Program Listings 71
5.8 Bibliographic Notes 71

6 Sequence Databases 72
6.1 FASTA Format 73
6.2 GenBank Format 73
6.3 GenBank’s Feature Locations 75
6.4 Reading Sequence Files in Perl 79
   6.4.1 Object-Oriented Programming in Perl 80
   6.4.2 The SimpleReader Class 81
   6.4.3 Hiding File Formats with Method Inheritance 85
6.5 Exercises 89
6.6 Complete Program Listings 91
6.7 Bibliographic Notes 92

7 Local Alignment and the BLAST Heuristic 93
7.1 The Smith–Waterman Algorithm 94
7.2 The BLAST Heuristic 96
   7.2.1 Preprocessing the Query String 98
   7.2.2 Scanning the Target String 99
7.3 Implementing BLAST in Perl 100
7.4 Exercises 106
7.5 Complete Program Listings 108
7.6 Bibliographic Notes 108

8 Statistics of BLAST Database Searches 109
8.1 BLAST Scores for Random DNA 109
8.2 BLAST Scores for Random Residues 114
8.3 BLAST Statistics in Perl 116
8.4 Interpreting BLAST Output 123
8.5 Exercise 125
8.6 Complete Program Listings 126
8.7 Bibliographic Notes 126

9 Multiple Sequence Alignment I 127
9.1 Extending the Needleman–Wunsch Algorithm 128
9.2 NP-Completeness 131
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.3</td>
<td>Alignment Merging: A Building Block for Heuristics</td>
<td>132</td>
</tr>
<tr>
<td>9.4</td>
<td>Merging Alignments in Perl</td>
<td>133</td>
</tr>
<tr>
<td>9.5</td>
<td>Finding a Good Merge Order</td>
<td>137</td>
</tr>
<tr>
<td>9.6</td>
<td>Exercises</td>
<td>139</td>
</tr>
<tr>
<td>9.7</td>
<td>Complete Program Listings</td>
<td>139</td>
</tr>
<tr>
<td>9.8</td>
<td>Bibliographic Notes</td>
<td>139</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10</th>
<th>Multiple Sequence Alignment II</th>
<th>141</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1</td>
<td>Pushing through the Matrix by Layers</td>
<td>141</td>
</tr>
<tr>
<td>10.2</td>
<td>Tunnel Alignments</td>
<td>147</td>
</tr>
<tr>
<td>10.3</td>
<td>A Branch-and-Bound Method</td>
<td>149</td>
</tr>
<tr>
<td>10.4</td>
<td>The Branch-and-Bound Method in Perl</td>
<td>152</td>
</tr>
<tr>
<td>10.5</td>
<td>Exercises</td>
<td>153</td>
</tr>
<tr>
<td>10.6</td>
<td>Complete Program Listings</td>
<td>154</td>
</tr>
<tr>
<td>10.7</td>
<td>Bibliographic Notes</td>
<td>154</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>11</th>
<th>Phylogeny Reconstruction</th>
<th>155</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1</td>
<td>Parsimonious Phylogenies</td>
<td>155</td>
</tr>
<tr>
<td>11.2</td>
<td>Assigning Sequences to Branch Nodes</td>
<td>157</td>
</tr>
<tr>
<td>11.3</td>
<td>Pruning the Trees</td>
<td>160</td>
</tr>
<tr>
<td>11.4</td>
<td>Implementing Phylogenies in Perl</td>
<td>162</td>
</tr>
<tr>
<td>11.5</td>
<td>Building the Trees in Perl</td>
<td>168</td>
</tr>
<tr>
<td>11.6</td>
<td>Exercise</td>
<td>171</td>
</tr>
<tr>
<td>11.7</td>
<td>Complete Program Listings</td>
<td>171</td>
</tr>
<tr>
<td>11.8</td>
<td>Bibliographic Notes</td>
<td>171</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12</th>
<th>Protein Motifs and PROSITE</th>
<th>173</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.1</td>
<td>The PROSITE Database Format</td>
<td>174</td>
</tr>
<tr>
<td>12.2</td>
<td>Patterns in PROSITE and Perl</td>
<td>175</td>
</tr>
<tr>
<td>12.3</td>
<td>Suffix Trees</td>
<td>177</td>
</tr>
<tr>
<td>12.3.1</td>
<td>Suffix Links</td>
<td>184</td>
</tr>
<tr>
<td>12.3.2</td>
<td>The Efficiency of Adding</td>
<td>188</td>
</tr>
<tr>
<td>12.4</td>
<td>Suffix Trees for PROSITE Searching</td>
<td>189</td>
</tr>
<tr>
<td>12.5</td>
<td>Exercises</td>
<td>193</td>
</tr>
<tr>
<td>12.6</td>
<td>Complete Program Listings</td>
<td>195</td>
</tr>
<tr>
<td>12.7</td>
<td>Bibliographic Notes</td>
<td>195</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>13</th>
<th>Fragment Assembly</th>
<th>196</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.1</td>
<td>Shortest Common Superstrings</td>
<td>196</td>
</tr>
<tr>
<td>13.2</td>
<td>Practical Issues and the PHRAP Program</td>
<td>202</td>
</tr>
<tr>
<td>13.3</td>
<td>Reading Inputs for Assembly</td>
<td>204</td>
</tr>
<tr>
<td>13.4</td>
<td>Aligning Reads</td>
<td>207</td>
</tr>
<tr>
<td>13.5</td>
<td>Adjusting Qualities</td>
<td>212</td>
</tr>
<tr>
<td>13.6</td>
<td>Assigning Reads to Contigs</td>
<td>217</td>
</tr>
<tr>
<td>13.7</td>
<td>Developing Consensus Sequences</td>
<td>222</td>
</tr>
<tr>
<td>13.8</td>
<td>Exercises</td>
<td>227</td>
</tr>
<tr>
<td>13.9</td>
<td>Complete Program Listings</td>
<td>230</td>
</tr>
<tr>
<td>13.10</td>
<td>Bibliographic Notes</td>
<td>230</td>
</tr>
</tbody>
</table>
14 Coding Sequence Prediction with Dicodons
14.1 A Simple Trigram Model
14.2 A Hexagram Model
14.3 Predicting All Genes
14.4 Gene Finding in Perl
14.5 Exercises
14.6 Complete Program Listings
14.7 Bibliographic Notes

15 Satellite Identification
15.1 Finding Satellites Efficiently
  15.1.1 Suffix Testing
  15.1.2 Satellite Testing
15.2 Finding Satellites in Perl
15.3 Exercises
15.4 Complete Program Listings
15.5 Bibliographic Notes

16 Restriction Mapping
16.1 A Backtracking Algorithm for Partial Digests
16.2 Partial Digests in Perl
16.3 Uncertain Measurement and Interval Arithmetic
  16.3.1 Backtracking with Intervals
  16.3.2 Interval Arithmetic in Perl
  16.3.3 Partial Digests with Uncertainty in Perl
  16.3.4 A Final Check for Interval Consistency
16.4 Exercises
16.5 Complete Program Listings
16.6 Bibliographic Notes

17 Rearranging Genomes: Gates and Hurdles
17.1 Sorting by Reversals
17.2 Making a Wish List
17.3 Analyzing the Interaction Relation
17.4 Clearing the Hurdles
17.5 Happy Cliques
17.6 Sorting by Reversals in Perl
17.7 Exercise
17.8 Appendix: Correctness of Choice of Wish from Happy Clique
17.9 Complete Program Listings
17.10 Bibliographic Notes

A Drawing RNA Cloverleaves
A.1 Exercises
A.2 Complete Program Listings
A.3 Bibliographic Notes
Contents

B  Space-Saving Strategies for Alignment  307
   B.1 Finding Similarity Scores Compactly  307
   B.2 Finding Alignments Compactly  309
   B.3 Exercises  312
   B.4 Complete Program Listings  312
   B.5 Bibliographic Note  312

C  A Data Structure for Disjoint Sets  313
   C.1 Union by Rank  314
   C.2 Path Compression  315
   C.3 Complete Program Listings  315
   C.4 Bibliographic Note  317

D  Suggestions for Further Reading  318

Bibliography  319
Index  325
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. 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 As, Cs, Gs, and Ts.

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.
The Central Dogma

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, 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. 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 and TGGAGGAAT are complementary:

\[ 5'\text{-ATTCCTCCA-3'} \]
\[ 3'\text{-TAAGGAGGT-5'} \]

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*. 
1.2 Chromosomes

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 \((3 \times 10^9)\) 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 chromosome. 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.
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.
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 complementar 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.\textsuperscript{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

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

The first line tells the operating system where to find the Perl interpreter.\textsuperscript{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.\textsuperscript{13} An empty hash is declared (i.e., brought into existence) by the statement

```perl
my %codonMap;
```

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

\textsuperscript{11} This is the Shine–Dalgarno sequence given previously. The exact sequence varies among species.
\textsuperscript{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.
\textsuperscript{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

\[ \text{$in = "Ala GCU GCC GCA GCG\n";} \]

(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

\[ \text{@codons = ("Ala","GCU","GCC","GCA","GCG");} \]

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

\[ \text{$residue = "Ala";} \]
\[ \text{@codons = ("GCU","GCC","GCA","GCG");} \]

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:

```perl
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";
}
```

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:

```perl
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:

```perl
sub translate {  ## 1
    my ($mrna) = @_;  ## 2
    my $pro = "";  ## 3
    while ($mrna =~ s/(...)/\{  ## 4
        $pro = $pro . $codonMap{$1};  ## 5
    \}  
    }
    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 \texttt{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 \texttt{cendog.pl}, we can execute it on a Unix system by moving to the directory containing the file and typing:

\texttt{./cendog.pl}

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

\texttt{perl cendog.pl}

Here is output from a sample execution of the program on the input \texttt{ACGGTCCTACCTTTA}, including original DNA, RNA transcript, and translations in reading frames 1, 2, and 3:

\begin{verbatim}
DNA: ACGGTCTACCTTTA
RNA: UAAAGGUAGGACCGU
RF1: ...Arg...AspArg
RF2: LysGlyArgThr
RF3: LysValGlyPro
\end{verbatim}

\subsection*{1.6 Exercise}

\begin{enumerate}
\item 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. \textit{Hint}: Check out $\&, .=, and \texttt{substr} in your favorite Perl reference.
\end{enumerate}

\subsection*{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 \texttt{chap1/cendog.pl} in the software distribution.

\begin{verbatim}
#!/usr/bin/perl
use strict; # request conservative error checking
\end{verbatim}

\footnote{We must also be sure that the file is “executable”, using the \texttt{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
GlnCAA CAG
Glu GAA GAG
Gly GGU GGC GGA GGG
His CAU CAC
Ile AUU AUC AUU
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 GUU 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
tein 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.\footnote{It is an elaboration of a free version downloadable from \url{www.perl.com}.} The “camel book” (Wall, Christiansen, and Orwant 1997) will be more appropriate for those less comfortable at the keyboard.\footnote{One of its authors, Larry Wall, is the inventor of Perl.} 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 \textit{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.