CSE 142, Autumn 2010
Programming Assignment #7: DNA (20 points)
Tuesday, November 23, 2010, 11:30 PM

Special thanks to UW professor Martin Tompa for his help with the development of this assignment.

This assignment focuses on arrays and file/text processing. Turn in a file named `DNA.java`. You will also need the two input files `dna.txt` and `ecoli.txt` from the course web site. Save these files in the same folder as your program.

The assignment involves processing data from genome files. Your program should work with the two given input files. If you are curious (this is not required), the National Center for Biotechnology Information publishes many other bacteria genome files. The last page tells you how to use your program to process other published genome files.

Background Information About DNA:

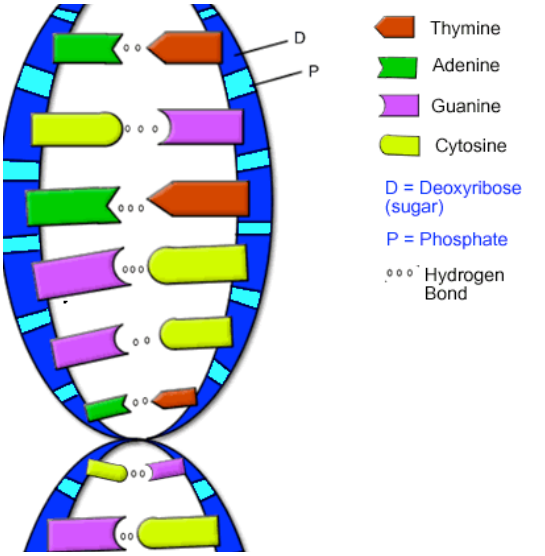
Note: This section explains the biology behind the assignment. It is for your information only; you need not fully understand it to complete the assignment.

Deoxyribonucleic acid (DNA) is a complex biochemical macromolecule that carries genetic information for cellular life forms and some viruses. DNA is also the mechanism through which genetic information from parents is passed on during reproduction. DNA consists of long chains of chemical compounds called *nucleotides*. Four nucleotides are present in DNA: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). DNA has a double-helix structure (see diagram below) containing complementary chains of these four nucleotides connected by hydrogen bonds.

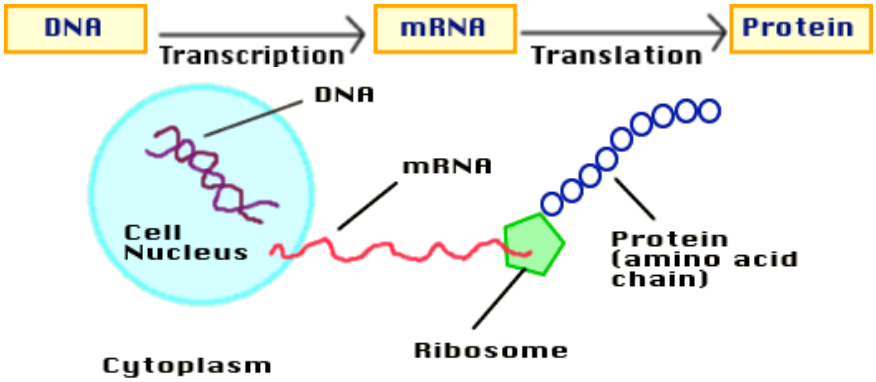
Certain regions of the DNA are called genes. Most genes encode instructions for building proteins (they're called "protein-coding" genes). These proteins are responsible for carrying out most of the life processes of the organism.

Nucleotides in a gene are organized into *codons*. Codons are groups of three nucleotides and are written as the first letters of their nucleotides (e.g., TAC or GGA). Each codon uniquely encodes a single amino acid, a building block of proteins.

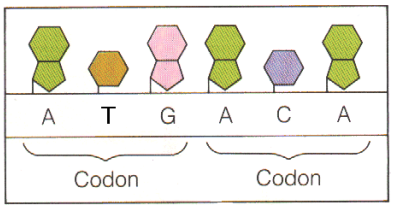
The process of building proteins from DNA has two major phases called *transcription* and *translation*, in which a gene is replicated into an intermediate form called *mRNA*, which is then processed by a structure called a *ribosome* to build the chain of amino acids encoded by the codons of the gene.



The chemical structure of DNA.



DNA translation.



The sequences of DNA that encode proteins occur between a *start codon* (which we will assume to be ATG) and a *stop codon* (which is any of TAA, TAG, or TGA). Not all regions of DNA are genes; large portions that do not lie between a valid start and stop codon are called *intergenic DNA* and have other (possibly unknown) function. Computational biologists examine large DNA data files to find patterns and important information, such as which regions are genes. Sometimes they are interested in the percentages of mass accounted for by each of the four nucleotide types. Often high percentages of Cytosine (C) and Guanine (G) are indicators of important genetic data.

For more information, visit the Wikipedia page about DNA at the following address, or use your favorite search engine:

http://en.wikipedia.org/wiki/DNA

Program Description:

In this assignment you read an input file containing named sequences of nucleotides and produce information about them. For each nucleotide sequence, your program counts the occurrences of each of the four nucleotides (A, C, G, and T). The program also computes the mass percentage occupied by each nucleotide type, rounded to one digit past the decimal point. Next the program reports the codons (trios of nucleotides) present in each sequence and predicts whether or not the sequence is a protein-coding gene. For us, a protein-coding gene is a string that matches the following constraints:

- begins with a valid *start codon* (ATG)
- ends with a valid *stop codon* (TAA, TAG, or TGA)
- contains at least 5 codons, including its initial start codon and final stop codon
- Cytosine (C) and Guanine (G) combined account for at least 30% of its mass

(These are approximations for our assignment, not exact constraints used in computational biology to identify proteins.)

Input Data:

The DNA data consists of line pairs. The first line has the name of the nucleotide sequence, and the second is the nucleotide sequence itself. Notice that the nucleotides in the input file can be either upper or lowercase.

Input file `dna.txt` (partial):

```
cure for cancer protein
ATGCCACTATGGTAG
captain picard hair growth protein
ATgCCAACATGgATGCCcGATAtGGATTgA
bogus protein
CCATtAATgATCaCAGTt
```

...

Program Behavior:

Your program begins with an introduction. Then your program asks for an input and output file. You may assume the user will type the name of an existing input file and is in the format described previously. Your program opens the input file to process the nucleotide sequences and outputs the results in the given output file. Notice the nucleotide sequence is output in uppercase, the nucleotide counts and mass percentages are in A, C, G, T order, and the codons are *not* unique.

Log of execution (user input underlined):

```
This program reports information about DNA
nucleotide sequences that may encode proteins.

Input file name? dna.txt
Output file name? output.txt
```

Output file `output.txt` after above execution (partial):

```
Name: cure for cancer protein
Nucleotides: ATGCCACTATGGTAG
Nucleotide counts: [4, 3, 4, 4]
Mass percentages: [27.3, 16.8, 30.6, 25.3]
Codons: [ATG, CCA, CTA, TGG, TAG]
Encodes a protein: yes

Name: captain picard hair growth protein
Nucleotides: ATGCCAACATGGATGCCCGATATGGATTGA
Nucleotide counts: [9, 6, 8, 7]
Mass percentages: [30.7, 16.8, 30.5, 22.1]
Codons: [ATG, CCA, ACA, TGG, ATG, CCC, GAT, ATG, GAT, TGA]
Encodes a protein: yes

Name: bogus protein
Nucleotides: CCATTAATGATCACAGTT
Nucleotide counts: [6, 4, 2, 6]
Mass percentages: [35.1, 19.3, 13.1, 32.5]
Codons: [CCA, TTA, ATG, ATC, ACA, GTT]
Encodes a protein: no
```

...

Implementation Guidelines and Hints:

The main purpose of this assignment is to demonstrate your understanding of arrays and array traversals with for loops. Therefore, you should use arrays to store the various data for each sequence. In particular, **your nucleotide counts, mass percentages, and codons should all be stored using arrays**. Additionally you should **use arrays and for loops** to transform the data from one form to another as follows:

- from the original nucleotide sequence string to nucleotide counts;
- from nucleotide counts to mass percentages; and
- from the original nucleotide sequence string to codon triplets.

You may find it useful to know that you can print any array in a convenient comma-separated format using the method `Arrays.toString`, which accepts an array as a parameter and returns a string representation of it. For example:

```
int[] numbers = {10, 20, 30, 40};
System.out.println("my data: " + Arrays.toString(numbers));
```

The preceding code produces the following output:

```
my data: [10, 20, 30, 40]
```

To compute mass percentages, use the following as the mass of each nucleotide (g/mol):

- Adenine (A): 135.128
- Cytosine (C): 111.103
- Guanine (G): 151.128
- Thymine (T): 125.107

For example, the mass of the sequence ATGGAC is (135.128 + 125.107 + 151.128 + 151.128 + 135.128 + 111.103) or 808.722. Of this, 270.256 (33.4%) is from the two Adenines, 111.103 (13.7%) is from the Cytosine, 302.256 (37.4%) is from the two Guanines, and 125.107 (15.5%) is from the Thymine.

We suggest that you start this program by writing the code to read the input file. Try writing code to simply read each protein's name and sequence of nucleotides and print them. Read each line from the input file using `Scanner`'s `nextLine` method. This will read an entire line of input and return it as a `String`.

Next, try writing the code that passes over the nucleotide sequence and counts the number of As, Cs, Gs, and Ts. You can use `String`'s `charAt` method to get individual characters of a string. Put your counts into an array of size 4. To map between nucleotides and array indexes, you may write a method that converts single characters (i.e. A, C, T, G) into indices (i.e. 0 to 3).

Once you have the counts working correctly, you can convert your counts into a new array of percentages of mass for each nucleotide using the preceding nucleotide mass values. If you've written code to map between nucleotide letters and array indexes, it may also help you to look up mass values in an array such as the following:

```
double[] masses = {135.128, 111.103, 151.128, 125.107};
```

You may store your mass percentages already rounded to one digit past the decimal or you can round when printing the mass percentages array using `printf`. If you choose to store your mass percentages pre-rounded, you may want to use `Math.round` as follows:

```
double d1 = 3.14159265;
d1 = Math.round(d1 * 10.0) / 10.0;
double d2 = 1.6666667;
d2 = Math.round(d2 * 10.0) / 10.0;
System.out.print("d1 = " + d1 + "; d2 = " + d2);
```

The preceding code produces the following output: `d1 = 3.1; d2 = 1.7`

After computing mass percentages, you must break apart the sequence into codons and examine each codon. You may wish to review the methods of `String` objects as presented in Chapters 3 and 4, such as `substring`, `charAt`, `indexOf`, `toUpperCase`, and `toLowerCase`.

We also suggest that you first get your program working correctly printing its output to the console before you save the output to a file. Once you have your program printing correct output to the console, save the output to a file by using a `PrintStream` as described in Section 6.4 of the textbook.

You may assume that the input file exists, is readable, and contains valid input. You may assume that each sequence's length will be a multiple of 3, although the nucleotides on a given line might be in either uppercase or lowercase form or a combination. Your program overwrites any existing data in the output file (the default `PrintStream` behavior).

Style Guidelines:

For this assignment you should have **four class constants**:

- one for the **minimum number of codons** a valid protein must have, as an integer (default of 5)
- a second for the **percentage** of mass from C and G in order for a protein to be valid, as an integer (default of 30)
- a third for the number of **unique nucleotides** (4, representing A, C, G, and T)
- a fourth for the number of **nucleotides per codon** (3)

For full credit it should be possible to change the first two constant values (minimum codons and minimum mass percentage) and cause your program to change its behavior for evaluating protein validity. The other two constants won't ever be changed but are still useful to make your program more readable. You should refer to these constants in your code and should never refer to the bare number such as 4 or 3 directly. You should use no other constants in your program other than the constants named above.

We will grade your method structure strictly on this assignment. Use at least **four nontrivial methods** besides `main`. These methods should use parameters and returns, including arrays, as appropriate. The methods should be well-structured and avoid redundancy. No one method should do too large a share of the overall task. The Case Study in book section 7.6 is a good example of a larger program with methods that pass arrays as parameters.

Your `main` method should be a concise summary of the overall program. It is okay for `main` to contain some code such as `println` statements. But the `main` method should not perform too large a share of the overall work itself, such as examining each of the characters representing nucleotides. Also avoid "chaining," when many methods call each other without ever returning to `main`. For reference, our solution is around 130 lines long and has 6 methods besides `main`, though you don't need to match this.

We will also check strictly for redundancy on this assignment. If you have a very similar piece of code that is repeated several times in your program, eliminate the redundancy such as by creating a method, by using `for` loops over the elements of arrays, and/or by factoring `if/else` code as described in section 4.3 of the textbook.

You are limited to features in Chapters 1 through 7. Follow past style guidelines such as indentation, names, variables, types, line lengths, and comments (at the beginning of your program, on each method, and on complex sections of code).

Optional Additional Input Files:

If you would like to generate additional input files to test your program, you can create them from actual NCBI genetic data. The following web site has many data files that contain complete genomes for bacterial organisms:

ftp://ftp.ncbi.nih.gov/genomes/Bacteria/

The site contains many directories with names of organisms. After entering a directory, you can find and save a genome file (a file whose name ends with .fna) and a protein table (a file whose name ends with .ptt). On the course web site we will provide you with a program to convert these .fna and .ptt files into input files suitable for your homework.