CSE 142, Autumn 2006
Programming Assignment #7: DNA (25 points)
Part A due: Tuesday, November 21, 2006, 2:00 PM (5 points)
Part B due: Wednesday, November 29, 2006, 9:00 PM (20 points)

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

This assignment will give you practice with arrays and text processing using genetic data.
Turn in files named `DnaA.java` (Part A) and `DnaB.java` (Part B).

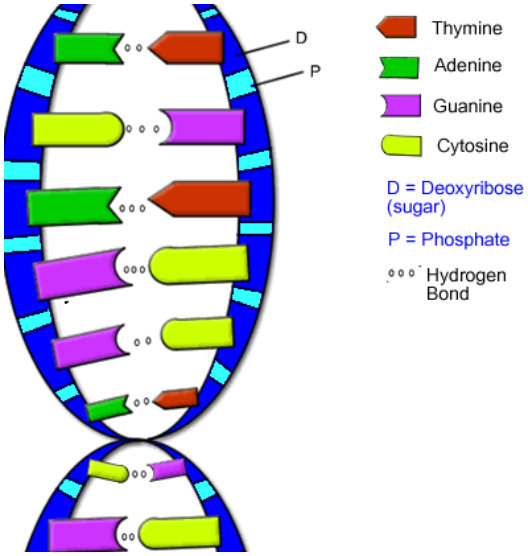
Background Information:

Deoxyribonucleic acid (DNA) is a complex biochemical macromolecule that carries genetic information for cellular life forms and some viruses. 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.

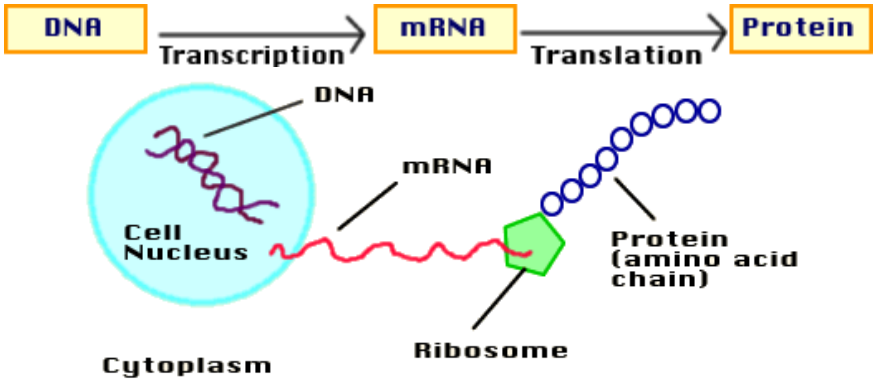
Certain portions of the DNA contain nucleotide sequences called *genes*, most of which encode instructions for building proteins. These proteins are responsible for carrying out most of the life processes of the organism. DNA is also the mechanism through which genetic information from parents is passed on during reproduction.

The nucleotides in a gene are organized into groups of three, called *codons*. Codons are typically written as the first letters of their three nucleotides, such as TAC or GGA. Each codon uniquely encodes a single amino acid, which is a building block of proteins.

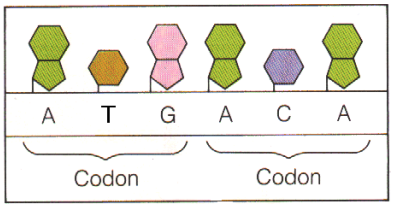
The process of building proteins from DNA has two major phases called *transcription* and *translation*, in which a gene is replicated into an intermediate form called *mRNA*, which is then processed by a structure called a *ribosome* to build the chain of amino acids encoded by the codons of the gene.



The chemical structure of DNA.



DNA translation.



The ranges 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 contain protein-encoding 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 encode particular proteins. They are also sometimes interested in the percentages of mass accounted for by each of the four nucleotide types. Often a high percentage of Cytosine (C) and Guanine (G) are found in regions of the DNA that contain 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 and Output:

In this assignment you will read an input file containing named sequences of nucleotides and produce information about each of them. For each nucleotide sequence, your program will count the occurrences of each of the four nucleotides (A, C, G, and T). The program will also compute the mass percentage occupied by each nucleotide type, rounded to one digit past the decimal point. Next the program will report the codons (trios of nucleotides) present in each sequence and a prediction of whether or not the sequence encodes a protein. You will conclude that a sequence encodes a protein if it meets the following constraints:

- begins with a valid *start codon* (ATG)
- ends with a valid *stop codon* (TAA, TAG, or TGA)
- contains at least 4 codons, including its initial start codon and final stop codon
- at least 30% of its mass is Cytosine (C) and Guanine (G)

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

The input file will contain data in the following format. For the moment, assume the data is stored in the file `dna.txt`:

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

...

Your program should produce a log of execution in the following format (user input is underlined):

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

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

The rest of your program's output will go into the output file specified on the console. For example, after the above run, the following output would be stored in the file `output.txt`:

```
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
```

...

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.

Part A (5 points):

For 5 points, turn in a version of this program that reads each nucleotide sequence but produces only the name, nucleotide sequence, and nucleotide counts. For Part A your program should print its output to the console and not to an output file; you should not prompt the user for an output file name in Part A.

To receive full credit, you must use an array to represent your nucleotide counts for each sequence. Otherwise, style will be ignored for Part A. Your code must have correct logic and approximately match output such as the following:

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

Input file name? dna.txt

Name: cure for cancer protein
Nucleotides: ATGCCACTATGGTAG
Nucleotide counts: [4, 3, 4, 4]

Name: captain picard hair growth protein
Nucleotides: ATGCCAACATGGATGCCCGATATGGATTGA
Nucleotide counts: [9, 6, 8, 7]

Name: bogus protein
Nucleotides: CCATTAATGATCACAGTT
Nucleotide counts: [6, 4, 2, 6]
...
```

Implementation Guidelines:

The main purpose of this assignment is to demonstrate your understanding of arrays. 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.

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:

- 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 its `nextLine` method. This will read an entire line of input and return it as a `String`.

From there, write the code that passes over the nucleotides in the sequence and counts the number of As, Cs, Gs, and Ts. You may use the string's `charAt` method to get the individual characters of this string. Put your counts into an array of size 4. To help you map between nucleotides and array indexes, you may wish to write a method that converts a single letter such as A, C, G, or T into an index between 0 and 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};
```

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 even while working on Part B, you may wish to make your program print its output to the console, rather than to the output file, for easier debugging. To produce output to a file, use 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).

Stylistic Guidelines for Part B:

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

- one for the minimum number of codons a valid protein must have, as an integer (default of 4)
- 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.

A major part of our stylistic grading for this assignment will involve redundancy. A naive implementation of this program might contain a great deal of redundant code and logic, but as we have stressed throughout the course, you should eliminate as much redundant code as possible.

One way to eliminate redundant code is through your use of arrays. For example, as mentioned previously, you should represent your nucleotide counts, mass percentages, and codons using arrays.

You should use methods in this program to provide structure and avoid redundancy. **For full credit on this program, you must have four non-trivial methods other than `main`.** You should decide for yourself exactly what methods to use, but for full credit, your methods should obey constraints such as (but not necessarily limited to) the following:

- No one method should be overly long.
- All of your file I/O should be done by one method.
- Each significant array in your program should be filled with data in its own individual method.
- Your `main` method should represent a concise outline of the behavior of your program; looking solely at `main` should present a clear idea of all of the major tasks your program is doing.
- Your methods should accept arrays as parameters or return arrays as their result values as appropriate.

You should follow past stylistic guidelines about indentation, whitespace, identifiers, and localizing variables. You should place comments at the beginning of your program, at the start of each method, and on complex sections of code.