Due date: Tuesday, November 26, 11:59pm.

Lab Assignment

Assignment Preparation

Lab type. This is a pair programming lab. You select your partner. As always, you are encouraged to select partners you have not worked with yet. You will stay with the same partner for the duration of Lab 12, both parts.

Purpose. The lab allows you to practice the use of strings in C programs. All lab assignments come from the area of bioinformatics, and all programs are designed to output information useful for biologists and biochemists. More about the bioinformatics aspects of this assignment below.

Programming Style. All submitted C programs must adhere to the programming style described in detail at

http://users.csc.calpoly.edu/~cstaley/General/CStyle.htm

When graded, the programs will be checked for style. Any stylistic violations are subject to a 10% penalty. Significant stylistic violations, especially those that make grading harder, may yield stricter penalties. Also note the the Lab 2 requirement for the content of the header comment in each file you submit applies to each assignment (lab, programming assignment, homework) in this course.

Testing and Submissions. Any submission that does not compile using the

gcc -ansi -Wall -Werror -lm

compiler settings will receive an automatic score of 0.

Program Outputs must co-incide. Any deviation in the output is subject to penalties. PLEASE, USE BINARY EXECUTABLES PROVIDED BY THE INSTRUCTOR! The exception is made in case of floating point computations leading to differences in the last few decimal digits. You can check whether or not a program produces correct output by running the diff command.

Please, make sure you test all your programs prior to submission! Feel free to test your programs on test cases you have invented on your own.

About this assignment

Among other disciplines Computer Science is unique in one important aspect. Computer Science (and, to large degree, Software Engineering,) study the use of computers for solving problems.
It is important to notice, that computer scientists and/or software engineers rarely are the sources of the problems themselves. More often than not, problems that can be solved via the use of computers (and via software development) come from other sciences, disciplines and fields.

As such, a majority of computer scientists, software developers and software engineers over course of their professional careers wind up working on problems in other subject areas: from business and finance, to agriculture, to arts, to natural sciences.

Working on problems of others and developing software solutions for them is what you will be doing a lot throughout your professional lives. Being able to do so successfully requires a certain set of skills, among which we find:

- **Ability to understand problem domains.** Be it warehousing or financial markets prediction or management of results of astronomical observations, if you have to develop software for it, you need to have understanding of important concepts from the problem area.

- **Ability to work across disciplines.** People who need software are likely NOT to be experts in computer science, software engineering or software development. They will rely on your expertise in that area. **However,** they are likely to be **experts in their fields,** and their knowledge and expertise is a crucial source of information for you during the software development process. To be efficient, you must be able to communicate well with experts from other fields.

  This is harder than one would believe. Occasionally, even the technical language you speak will be different. Details that are of importance to you will seem trivial to your customers, and vice versa – the customers will be expressing concerns over things that barely register on your radar.

It is never too early to start developing appropriate skills. This lab introduces you to the art and the craft of solving problems from a domain you may be unfamiliar with. While the problems you are asked to solve are *simple,* they are *real:* actual bioinformatics software that is used by biochemists all over the world solves all of these problems all the time as part of performing more complex analyses.

Enjoy!

**Bioinformatics in a Nutshell**

Wikipedia\(^1\) defines *bioinformatics* as follows:

**Bioinformatics** is the application of information technology to the field of molecular biology.

The article continues:

> Bioinformatics now entails the creation and advancement of databases, algorithms, computational and statistical techniques, and theory to solve formal and practical problems arising from the management and analysis of biological data.

While some of the problems addressed in the field of bioinformatics require years of education and advanced degrees in both Biology and Computer Science to study and solve, a *wide range* of bioinformatics-related tasks can be performed using relatively simple programs. This lab offers five such problems for you to solve.

**Bioinformatics: the Data**

Among the wide range of bioinformatics applications, we will concentrate on the problems associated with the field of *genomics,* i.e., the study of *genomes of organisms.* In particular, one of the key subject matters of *genomics* is the discovery of the *DNA Sequences* of various organisms.

---

\(^1\)http://en.wikipedia.org/wiki/Bioinformatics
DNA is a.k.a. deoxiribonucleic acid is a molecule that contains the genetic instructions used in the development and functioning of all known living organisms2.

DNA is an extremely large and complex molecule. It consists of a large number of simpler components called nucleotides. A nucleotide molecule consists of a three components, two of which, the phosphate and the sugar have the same chemical structure for all nucleotides. The third component, the base is the one that distinguishes different nucleotides. There are four types of nucleotide bases present in DNA molecules:

- Adenine
- Cytosine
- Guanine
- Thymine

DNA Structure. One of the greatest scientific discoveries of the 20th century was the discovery of the structure of a DNA molecule. A DNA molecule is a double helix consisting of two strands of nucleotides3. One of the key properties of DNA is base pairing: the four nucleotides form specific pairings:

- If one strand has an Adenine base, then the corresponding position on the other strand is occupied by the Thymine base and vice versa.
- If one strand has a Cytosine, the the corresponding position on the other strand is occupied by Guanine and vice versa.

The base pairing property means that

A single DNA strand uniquely determines the full structure of a DNA molecule.

The Adenine – Thymine and Cytosine – Guanine pairs of nucleotide bases are called base pairs. The two strands are called complementary to each other.

DNA Sequences. A DNA Sequence, is a representation of a DNA molecule as a string. In particular, a single DNA sequence represents one strand of a DNA molecule. The four nucleotides found in DNA are encoded with four letters according the following coding table:

<table>
<thead>
<tr>
<th>Nucleotide base</th>
<th>Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adenine</td>
<td>A</td>
</tr>
<tr>
<td>Cytosine</td>
<td>C</td>
</tr>
<tr>
<td>Guanine</td>
<td>G</td>
</tr>
<tr>
<td>Thymine</td>
<td>T</td>
</tr>
</tbody>
</table>

Thus, a sequence of Adenine, Adenine, Cytosine, Thymine, Guanine, Thymine will be encoded as a DNA sequence "AACTGT".

DNA Sequence Directionality and reverse complement. A strand on a DNA molecule has orientation. The two ends of a strand have different chemical structure, and this allows researchers to label them and to represent all DNA sequences as being headed in the same direction.

The two ends of a DNA strand are referred to as 5′ ("five prime") and 3′ ("three prime") – an allusion to the chemical structures found on each end.

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

3In addition to the double helix structure, a DNA molecule may have a non-trivial three-dimensional structure, but the problems we will be studying in this assignment do not take this into account.
DNA strands in a helix have opposite orientation. The two DNA strands have opposite orientation.

Figure 1 represents a DNA molecule fragment. The orientation of the light-gray DNA strand is left-to-right. The orientation of the dark-gray DNA strand is right-to-left. The light-gray DNA strand contains the following sequence of nucleotides:

$$\text{TCTTGATCGAAA}$$

The dark-gray DNA strand contains the complementary sequence $$\text{AGACTAGCTTT}$$, however, when shown in this order, the sequence runs from the $$3'$$ end to the $$5'$$ end. To correctly represent this sequence, we must reverse it, i.e., read it from right to left. The resulting sequence of nucleotides, $$\text{TTTCGATCAAGA}$$ has the correct orientation and is called the reverse complement of the sequence $$\text{TCTTGATCGAAA}$$.

Reverse complements. In general given a DNA sequence $$S = L_1 L_2 \ldots L_N$$, where $$L_i$$ represents one of the four letters $$A,T,C,G$$, its reverse complement is the DNA sequence that must occupy the corresponding bases on the other DNA strand, ordered from $$5'$$ to $$3'$$. Remember that we have the following complement relationship between the nucleotides:
<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>Three-letter code</th>
<th>One-letter code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alanine</td>
<td>Ala</td>
<td>A</td>
</tr>
<tr>
<td>Arginine</td>
<td>Arg</td>
<td>R</td>
</tr>
<tr>
<td>Asparagine</td>
<td>Asn</td>
<td>N</td>
</tr>
<tr>
<td>Aspartic acid</td>
<td>Asp</td>
<td>D</td>
</tr>
<tr>
<td>Cysteine</td>
<td>Cys</td>
<td>C</td>
</tr>
<tr>
<td>Glutamic acid</td>
<td>Glu</td>
<td>E</td>
</tr>
<tr>
<td>Glutamine</td>
<td>Glu</td>
<td>Q</td>
</tr>
<tr>
<td>Glycine</td>
<td>Gly</td>
<td>G</td>
</tr>
<tr>
<td>Histidine</td>
<td>His</td>
<td>H</td>
</tr>
<tr>
<td>Isoleucine</td>
<td>Ile</td>
<td>I</td>
</tr>
<tr>
<td>Leucine</td>
<td>Leu</td>
<td>L</td>
</tr>
<tr>
<td>Lysine</td>
<td>Lys</td>
<td>K</td>
</tr>
<tr>
<td>Methionine</td>
<td>Met</td>
<td>M</td>
</tr>
<tr>
<td>Phenylalanine</td>
<td>Phe</td>
<td>F</td>
</tr>
<tr>
<td>Proline</td>
<td>Pro</td>
<td>P</td>
</tr>
<tr>
<td>Serine</td>
<td>Ser</td>
<td>S</td>
</tr>
<tr>
<td>Threonine</td>
<td>Thr</td>
<td>T</td>
</tr>
<tr>
<td>Tryptophan</td>
<td>Trp</td>
<td>W</td>
</tr>
<tr>
<td>Tyrosine</td>
<td>Tyr</td>
<td>Y</td>
</tr>
<tr>
<td>Valine</td>
<td>Val</td>
<td>V</td>
</tr>
</tbody>
</table>

Table 1: Amino Acids.

\[
\begin{align*}
\text{compl}(A) &= T \\
\text{compl}(T) &= A \\
\text{compl}(C) &= G \\
\text{compl}(G) &= C
\end{align*}
\]

To construct an reverse complement of a sequence \( S = L_1L_2\ldots L_N \), we perform two steps:

1. **Step 1: complement.** Each letter \( L_i \) in the sequence is replaced with its complement \( \text{compl}(L_i) \). The resulting sequence is \( S^C = \text{compl}(L_1)\text{compl}(L_2)\ldots \text{compl}(L_N) \).

2. **Step 2: Reversion:** The sequence \( S^C \) is rewritten from right to left to form the reverse complement sequence

   \[
   S^t = \text{compl}(L_N)\text{compl}(L_{N-1})\ldots \text{compl}(L_2)\text{compl}(L_1).
   \]

**From Nucleotide Sequences to Amino Acids**

**Amino Acids.** Amino acids are molecules that are used in building proteins. Amino acids are found in many other compounds in living organisms, and they play important roles in a variety of biochemical processes. More importantly (for us), amino acids form the next layer in the hierarchy of DNA encoding.

There are twenty amino acids. The table with their full names, three-letter abbreviations and (more importantly), single-letter codes is shown in Table 1.

Certain sequences of amino acids form proteins.

**Genetic Code.** In DNA molecules, each triple of consecutive nucleotides encodes one amino acid or serves as a special marker. The triple of nucleotides is called a codon.
Table 2: Genetic Code.

The translation of the nucleotide triples (codones) into amino acid codes is called the genetic code.

With four nucleotides, there are 64 possible codons that encode 20 amino acids and two special markers. A single amino acid can be encoded by multiple codons.

Table 2 shows the genetic code.

**Start and Stop codons.** Three codons, TAA, TAG and TGA do not encode any amino acids. Instead, these codons represent the end of a protein molecule encoded by a DNA sequence. They are called stop codons.

Additionally, the ATG codon encodes Methionine, the amino acid that serves as the beginning of encodings of all proteins within a DNA sequence. Whenever Methionine is found at the beginning of a protein, the ATG sequence encoding it is called a start codon.

**Translation from Nucleotide sequences to Amino Acid sequences.** Using the genetic code table, any DNA sequence written in the alphabet of nucleotides can be translated into a sequence of amino acids. To do this,

1. Start at the 5' end of the DNA sequence.
2. For each triple of nucleotides, find, using the genetic code table, the matching amino acid (or start/stop codon).
3. Write out the amino acids and/or start/stop codons in a sequence following the 5' to 3' order.

**Example.** Consider the following DNA sequence:

```
TCTTGATCGAAA
```

We split it into triples of nucleotides:

```
TCT TGA TCG AAA
```
We then use the genetic code table, to substitute each triple with the matching amino acid:

\[
\begin{array}{cccc}
\text{TCT} & \text{TGA} & \text{TCG} & \text{AAA} \\
\text{S} & \text{Stop} & \text{S} & \text{K}
\end{array}
\]

**Frames.** Typically, DNA sequences represent portions of a DNA molecule. DNA molecules consist of millions of individual nucleotides, but current DNA sequencing equipment is capable of sequencing (i.e., discovering from a sample) only small “chunks” at a time. Given a DNA sequence in a nucleotide alphabet, there are three possibilities for translating it into a sequence of amino acids. These possibilities are called frames.

1. **Frame 1.** First codon starts at the first nucleotide.
2. **Frame 2.** First nucleotide is ignored; first codon starts at the second nucleotide.
3. **Frame 3.** First and second nucleotides are ignored; first codon start at the third nucleotide.

Intuitively, the frames can be explained as follows. Given a DNA sequence, we do not know whether it starts at the beginning of an amino acid encoding, or in the middle of it. There are three cases, and in order to translate a DNA fragment into a sequence of amino acids, we need to consider all possibilities. The three frames indicate where the first full amino acid of the fragment starts: at the beginning of the fragment, at the second nucleotide or at the third nucleotide.

**Example.** Consider the following DNA sequence:

\[
\text{TCTTAATCGAATCGAT}
\]

This sequence can be split into codons in three different ways:

Frame 1:  TCT TAA TCG AAT CGA T  
Frame 2:  T CTT AAT CGA ATC GAT  
Frame 3:  TC TTA ATC GAA TCG AT

In translating the DNA sequence in three frames, any nucleotides that are not part of a codon are ignored. The remaining codons are translated into amino acids using the genetic code table:

Frame 1:  TCT TAA TCG AAT CGA T  
\[
\begin{array}{cccc}
\text{S} & \text{Stop} & \text{S} & \text{N} \quad \text{R}
\end{array}
\]

Frame 2:  T CTT AAT CGA ATC GAT  
\[
\begin{array}{cccc}
\text{L} & \text{N} & \text{R} & \text{I} \quad \text{D}
\end{array}
\]

Frame 3:  TC TTA ATC GAA TCG AT  
\[
\begin{array}{cccc}
\text{L} & \text{I} & \text{E} & \text{S}
\end{array}
\]

So, the same DNA sequence, may give rise to three different sequenced amino acids: ”**StopSNR**”, ”**LNRID**” and ”**LIES**”\(^4\), depending on which frame is actually correct.

\(^4\)This was not intentional.
**Three more frames.** DNA molecule has two strands. Given a DNA sequence that comes from one of the strands, it is possible that either this sequence, or the corresponding sequence from the other strand participates in describing a protein. Therefore, given a DNA sequence, we may need to convert both it, and its reverse complement to the amino acid sequences. Each of the two sequences (direct and the reverse complement) can be converted using three frames. Therefore, the total number of frames, i.e., plausible amino acid sequences represented by a DNA fragment is six: three for the fragment itself, and three for its reverse complement.

**Example (continued).** Consider again, the DNA sequence

\[
\text{TCTTAATCGAATCGAT}
\]

The example above shows the three frames of translation of this fragment into amino acid sequences. We now complete the full list of translations, by using the reverse complement of this fragment.

The reverse complement of the fragment is

<table>
<thead>
<tr>
<th>Sequence: TCTTAATCGAATCGAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>complement sequence: AGAATTAGCTTAGCTA</td>
</tr>
<tr>
<td>reverse complement: ATCGATTGATTAAGA</td>
</tr>
</tbody>
</table>

The three frames for the reverse complement are:

Frame 4: ATC GAT TCG ATT AAG A
Frame 5: A TCG ATT CGA TTA AGA
Frame 6: AT CGA TTC GAT TAA GA

The translation to the sequences of amino acids is:

Frame 4: ATC GAT TCG ATT AAG A
         I D S I K
Frame 5: A TCG ATT CGA TTA AGA
         S I R L R
Frame 6: AT CGA TTC GAT TAA GA
         R F D Stop

Combining the results of the two examples, here are the six amino acid sequences that may be encoded by the given DNA fragment:

S<Stop>SNR  
LNRID  
LIES  
IDSIK  
SIRLR  
RFD<Stop>
Assignment

Your assignment for this lab is to develop software solutions for five problems from the field of bioinformatics.

Problem 1: The Reverse Complement

Write a program that takes as input a single DNA sequence in nucleotide alphabet and outputs the reverse complement of this string.

Program name. Name your program icomplement.c.

Input. The input to the program is a single string consisting of letters A, T, C and G. The string will be terminated with a newline character.

The maximum length of an input string is 400.

Output. The output of the program is a single string (terminated with a new line character) representing the reverse complement of the input string.

Functions and header files. Create a header file genomics.h. You will use it, and the file genomics.c, in which you will put all function definitions, to put function definitions for all subsequent programs.

Declare (and define) the following functions.

- char compl(char c). This function takes as input a character in the nucleotide alphabet \{A,T,C,G\} and returns its complement.

- void complement(char s[], char compl[], int length). This function takes as input two strings, s, as well as initially empty string compl, and the number length, representing the length of the string s. The function puts the complement of string s (i.e., the string that consists of the complements of individual characters of s in the same order) into string compl.

- void reverse(char s[], char inv[], int length). This function takes as input two strings, s, and an initially empty string inv, as well as the number length, representing the length of the string s. The function inverts s and put the reverse of s into string inv.

Your main() function for this program must produce the reverse complement of the input string using the functions described above. (It goes without saying that your program should #include "genomics.h").

Compile your program together with genomics.c.

Example. Consider the following input string:

ATTCCATGG

The output of your program will be:

CCATGGAAT

Problem 2: Conversion of Nucleotide Sequences into Amino Acid Sequences

Your second program will apply the genetic code to convert a DNA sequence in a nucleotide alphabet into all possible amino acid sequences it represents.
Program name. Name your program dna2amino.c

Input. The input to this program is the same as the input to the icomplement.c program: a newline-terminated string of characters in a nucleotide alphabet.

Output. The output of the program is six newline-terminated strings in the amino acid alphabet. Each string represents a translation of the input DNA sequence into an amino acid sequence on one of the six possible frames. The frames are output in the order from Frame 1 (actual DNA sequence, starting at the first nucleotide) to Frame 6 (reverse complement sequence, starting at the third nucleotide).

No other output is to be generated.

In the output, the stop codons that encode nucleotide sequences TAA, TAG and TGA are to be represented using the character '#'.

Example. If the input to the icomplement.c program is

TCTTAATCGAATCGAT

then the program shall output the following:

S#SNR
LNIRI
LIES
IDSIR
SIRLR
RFD#

(see examples above for the translation).

Functions and header files. Your program shall use the same genomics.h header file and genomics.c collection of functions, as the icomplement.c program does. It will wind up using some of the same functions. In addition, more functions will be added to the genomics.h library.

Because your program needs to construct the reverse complement of the input in order to determine frames 4–6 of the output, your program will use the complement() reverse() (and compl()) functions defined in genomics.h for the icomplement.c program.

Additionally, the following functions shall be declared and defined in genomics.h.

- char geneticCode(char c1, char c2, char c3). This program takes as input three char values representing a triple of nucleotides in a DNA sequence (i.e., a codon). It outputs the 1-letter symbol representing an amino acid, encoded by the input nucleotides, according to the genetic code table. For the start codon ATG, geneticCode() shall return 'M' (note, that this is the only codon encoding Methionine as well, and therefore, there will not be an ambiguity about it). For the stop codons TAA, TAG and TGA, geneticCode() shall return '#'. For all other nucleotide triples, the output of geneticCode() is uniquely determined by the genetic code table (Table ??).

- convert2Amino(char s[], char amino[], int length, int frame). This function takes as input a string s containing a nucleotide sequence of length length, an initially empty string amino, and the conversion frame represented by the frame parameter. The function translates the input nucleotide sequence into an amino acid sequence in the specified frame. The result of the translation is stored in the amino parameter.
Problem 3: Finding Palindromes

In linguistics, a palindrome is a string (sentence of text) that spells the same when read from left-to-right and from right-to-left. "Meaningful" palindromes exist in essentially every human language. Some examples of English palindromes are:

- eye
- Anna
- tenet
- rotator
- redivider
- madam
- madam I’m Adam
- never odd or even
- murder for a jar of red rum
- rats live on no evil star
- God saw I was dog

Palindromes in genomics. In genomics, a version of palindromes plays an important role in determining the 3D structure of a DNA molecule.

A DNA palindrome is a sequence of characters in the nucleotide alphabet \{A, C, G, T\}, such that it is equal to its reverse complement.

Example. Consider a sequence

\[
\text{AGTACT}
\]

It’s complement is \text{TCATGA}. It’s reverse complement is \text{AGTACT}, i.e., the original string.

The biological significance of the palindromes in DNA sequences is illustrated in Figures ?? and ??.

Essentially, because if a palindrome is present in a DNA sequence on some strand, the DNA strand may become entangled in this place, as the nucleotides in the palindrome sequence may get "paired" with their complements in the palindrome sequence, rather than with the nucleotides on the second strand. Such entanglements, called hairpins are common in DNA sequences, and biochemists want to be able to find where they occur in the DNA.

Your assignment. You will write a program that takes as input a DNA sequence string in nucleotide alphabet, looks for palindromes in it, and reports the longest palindrome it finds.

To simplify the problem, you will be looking for only one type of palindrome in the input strings: the palindromes formed around the center of the string.

Program name. Name your program \text{palindrome.c}.

Input. The input to this program is the same as the input to \text{icomplement.c} and \text{dna2amino.c}.
Figure 2: Palindromes in DNA sequences form hairpins.

Figure 3: Palindromes in DNA sequences form hairpins. (part 2)
Output. The program shall output the following information:

- Length of the longest palindrome found.
- The palindrome itself, printed out in the following way:
  - The first line of the output, contains the entire palindrome.
  - The second line of the output contains the reverse of the second half of the palindrome, formatted so that each character is directly below its complement in the first half.

For example, let the input to the program be

AATCGATG

The longest palindrome, around the center of this sequence is ATCGAT. The output of the program will be:

Longest palindrome: 6 characters long
ATCGAT
TAG

Finding palindromes. A sequence can have palindromes anywhere. You are only responsible for detecting the longest palindrome centered at the midpoint of the input string. There are two cases to consider:

1. Input string has even length. In this case, the mid-point of the string lies between two nucleotides. You detect the palindrome by comparing the appropriate characters on both sides of the midpoint.

   For example, if the input string AATCGATT is stored in a string variable
   
   ```
   char s[8] = {'A','A','T','C','G','A','T','T'}
   ```

   the midpoint occurs between s[3] and s[4]: the fourth and the fifth character in the string.

2. Input string has odd length. In this case, the mid-point falls onto a nucleotide. Your program will omit this nucleotide and will start matching the nucleotides immediately before and after it, effectively excluding the central nucleotide from consideration.

   For example, the sequence AAT will be considered a palindrome by your program. The midpoint, the second A nucleotide is not used, but the nucleotides on both sides of it, A and T are compared to each other. By the same token, ATT, ACT and AGT are all palindromes of length 3, and ATTAT, ATAAT, ATCAT and ATGAT are palindromes of size 5.

Functional decomposition. Feel free to use functions from the genomics.h library that you have developed for icomplement.c and dna2amino.c. There is no requirement for you to create any specific functions for this program, however, you may choose to do so. If you do, declare and define these functions in the genomics.h header file.

Problem 4: GC-percentage computation.

GC-percentage. Given a DNA sequence $S = s_1 \ldots s_N$, a GC-percentage of $S$ is the percentage of characters in $S$ that are either 'G' or 'C'.

For example, is $S = ATGTTCTTA$, the GC-percentage of $S$ is 0.3 (or 30%), as $S$ contains two 'G's and one 'C' out of 10 characters.
GC content refers to the percentage of DNA bases that contain either G (guanine) or C (cytosine). DNA is the genetic material of life that encodes instructions for making building blocks of living cells—proteins. DNA is a linear polymer that contains information in the order/sequence of bases. Four bases found in DNA are G, C, T, and A. The frequency of bases in a DNA polymer varies among different organisms. GC content of entire genome (all DNA of one organism or species) can be used to distinguish two closely related species or to find a foreign gene in a genome (a gene that jumped species; for example, viruses can transfer genes from one organism/species to another).

In addition, GC content can vary between different functional regions within one genome. When analyzing a new genome, finding regions of high or low GC percentage can point to particularly interesting regions (e.g., protein coding genes, origin of replication). Most molecular biologists use GC content of small regions of DNA to choose primers (molecular tools) for DNA amplification by PCR.

**Task.** Write a program, gcp.c, that reads a DNA string in nucleotide alphabet from standard input, computes the GC-percentage of the string and prints it out.

For input we will use the same files (dna-testNN and dna-fragmentNN) as we used for the Lab 12-1 programs.

Your program shall read the string, output it (prefacing it with the text "Sequence : " (notice the two spaces). On the next line, your program shall print the text "GC-content: " followed by the computed GC-percentage number, formatted to display two (2) digits after the decimal point.

**Example.** Here is an example of how the output of your program must look.

```
$ cat dna-test04
ATGAAACCCGGGTGA
$ gcp < dna-test04
Sequence : ATGAAACCCGGGTGA
GC-content: 53.33
```

**Functions.** You shall add a new function to you genomics.c function library (see Lab 12-1 handout). As usual, the appropriate function declaration also needs to be added to the genomics.h file. The function prototype is

```c
void updateGCCount(char s[], int * gc, int * at);
```

The function takes as inputs three parameters. The first one, char s[] is the DNA string whose GC-percentage needs to be computed. The second and third parameters are references to integer variables. One, gc represents the current count of 'G' and 'C' occurrences in the observed DNA strings. The other one, at represents the current count of 'A' and 'T' occurrences in the observed DNA strings.

The function shall sweep through the contents of s and update the gc and at counts appropriately.

**Note:** This function is a bit of an overkill for the task at hand. However, in many scenarios biologists need to find a combined GC-percentage of multiple DNA strings (e.g., DNA strings representing two different chromosomes of the same organism). updateGCCount() is handy in such situations. (Also, keeping both GC- and AT- counts is unnecessary, but convenient on occasion).

You must write the int main() of your GC-percentage computation program in a way that uses (calls) updateGCCount().

**Problem 5: Consensus Sequence.**

This is the only program in this lab that will take a different type of input.
**Definition.** Given a set of DNA strings of the same length in the nucleotide alphabet (usually representing the same DNA fragment for different related species/organisms), a *consensus sequence* is a string in a nucleotide alphabet that has at each position the nucleotide that shows in the plurality of DNA strings at that position.

**Example.** Consider the following collection of three DNA strings:

GTTAC  
GATAA  
GAATC

The string **GATAC** is the consensus sequence for this set of strings. **G** appears in all three strings in position 1, **A** appears twice in position 2, **T** appears twice in position 3, **A** appears twice in position 4, and **C** appears twice in position 5.

Some sets of strings can have multiple consensus sequences. E.g., the set of four strings:

TTGCA  
TTGGA  
TTGCT  
TTGGT

has four consensus sequences: **TTGCA**, **TTGCT**, **TTGGCT** and **TTGGT** - in the last two positions we have even number of **G** and **C**, and **A** and **T** nucleotides respectively.

**Task.** Your task is to write a program, `consensus.c`, which reads 10 nucleotide sequences from the standard input (all sequences will have the same length) and outputs a consensus sequence for these 10 sequences.

In case when there are multiple possible consensus sequences, your program shall report *any one* consensus sequence.

**Example.** — Here is an example of running this program.

$ cat seq-test02  
ATCGATCGAT  
ATCGATCGAT  
ATCGATCGAT  
ATCGATGGAT  
ATCGATGGAT  
ATGGATGGAA  
ATGGATAGAA  
ATGGATAGAA  
ATGGATTGAA  
ATGGATTGAA  

$ consensus <seq-test02  
ATCGATCGAT

**Notes.** Feel free to implement this program in any way you find convenient. There are no required functions for this program. You are allowed to define functions that help you simplify your `int main()`, but, please, declare and define them directly in the `consensus.c` file.
Submission.

Files to submit. Each pair submits one set of files from one account. The following files are mandatory:

- team.txt, icomplement.c, dna2amino.c, palindrome.c, genomics.h, genomics.c, gcp.c, consensus.c

If you have developed other files (extra .h file, for example), submit them as well.

The team.txt file shall contain the name of the team and the names of all team members in each pair, and the Cal Poly IDs of each. E.g., if I were on the team with Dr. John Bellardo, my team.txt file would be:

Go, Poly!
John Bellardo, bellardo
Alex Dekhtyar, dekhtyar

Submit team.txt as soon as you form your group.

Files can be submitted one-by-one, or all-at-once.

Submission procedure. You will be using handin program to submit your work. The procedure is as follows. Ssh to unix1, unix2, unix3, or unix4. The handin command is:

> handin dekhtyar lab10 <files>

Testing

Tests are available on the course web page. Please note, that test input files for the icompliment.c, dna2amino.c, palindrome.c and gcp.c are the same. While some data is "toy", to help you debug your programs, a portion of the test cases will contain real and long(ish) DNA sequences that may present actual interest to the students of bioinformatics courses.