Bioinformatics Programming

Some Perl reminders that may be helpful for writing programs for the following problems.

- An array holds a list of objects identified by its index in the array. For example, the following code creates an array called $myArray, adds three strings to it, and prints each one on a new line.

```
    my @myArray;
    push(@myArray, “C”);
    push(@myArray, “G”);
    push(@myArray, “T”);
    foreach $item (@myArray) {
        print “$item¥n”;
    }
```

- The underbar variables (@_ and $_) are special. $_ is automatically set if a variable is not specified for the foreach loop. For example, if $item was not specified above, we could simply print $_ inside the loop.

- A two-dimensional array can be created by adding arrays to arrays, as in the following.

```
    my @myMatrix; // create the global array
    for ($i = 0; $i < $n; $i++) {  // for each row
        my @newRow = ();      // create a new row
        for ($j = 0; $j < $n; $j++) {  // for each item in the row
            push(@newRow, $i*10+$j);  // add a number to the row
        }
        push(@myMatrix, [@newRow]); // note the square brackets!
    }
    foreach (@myMatrix) {    // for each row in the matrix
        my @newRow = @{$_};  // get the row as an array
        foreach (@newRow) {  // for each item in the row
            print $_,”¥t”;    // print the item, followed by a tab
        }
        print “¥n”;          // print a newline
    }
```

- Arguments to a program are automatically defined in Perl as `$ARGV[0]`, `$ARGV[1]`, etc., corresponding to the first argument, second argument, etc. So you can use these variables directly from the beginning of your code.

- The contents of the matrix created above could also be accessed directly using index numbers. For example, instead of using foreach, we could use a for loop as in the following:

```
for ($i = 0; $i < $n; $i++)  {
   for ($j = 0; $j < $n; $j++) {
      print $myMatrix[$i][$j],"¥t";
   }
   print “¥n”;
}
```

- You can read in a file given its filename, say `$filename`, in any of the following ways:

```
open(FILE, $filename);
my @contents = <FILE>;
// the entire contents of the file is now in the array @contents,
// where each item in the array corresponds to a line
// (including the newline character at the end of the line)
```

```
// the following form is more memory-efficient
// since it avoids putting the entire contents of the file in memory
open(FILE, $filename);
while (<FILE>) {  // recourses through each line in the file
   my $line = $_; // this is the next line in the file
   chomp($line); // removes the newline at the end of the line
}
```

- The `index($str,$substr,$pos)` function in Perl can be used to return the position of the first occurrence of a string (i.e. `$substr`) within the string `$str` at or after position `$pos`. If a third argument is not provided, the search starts from the beginning of the string. The return value is based at 0, and if the substring is not found, it returns -1.

- The `length($str)` function returns the number of characters in `$str`.

- The `substr($expr,$offset,$len)` extracts a substring out of `$expr` and returns it. The first character is at offset 0. If `$offset` is negative, it starts `$offset` positions from the end of the string. If `$len` is omitted, it returns everything to the end of `$expr`.

- To compare numbers to see if they are equal, use “==”, to compare strings to see if they are equal, use “eq.”

Problem 1

Part 1:

Let's think of an algorithm for the following problem.

> **Input**: a number
>
> **Output**: the Fibonacci number corresponding to it
>
> The Fibonacci number is defined by the following recurrence relation:

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n > 1. \end{cases}$$

Part 2:

Now write the program for computing the Fibonacci number for a given input value, in Perl. Feel free to make a copy of one of the programs in the notes from yesterday and modify it!

Part 3:

What is the running time of your algorithm?

Problem 2

Now let's try a bioinformatics problem. The GC content of a sequence is defined by the following formula, where each capital letter is the number of occurrences of that letter.

$$\frac{G + C}{A + T + G + C} \times 100$$

> **Input**: a DNA sequence in FASTA format
>
> **Output**: its GC content.

Test your program with an actual DNA sequence from GenBank.

Problem 3

Ok, finally for a more challenging problem.

> **Input**: two amino acid sequences in FASTA format
>
> **Output**: its alignment using the dynamic programming algorithm taught yesterday.

Hint: use your own pre-defined symbols, numbers, or strings for up-arrow, left-arrow, etc.