# BIOL591: Introduction to Bioinformatics
# Microarray Analysis Case Study: The Program

**Outline:**

I. Goal of program
II. Overview of program shell
III. Outline of your additions to the shell
IV. Write the subroutines
V. Sort the training set and print it or write it to disk

## I. Goal of program

Our goal now is to write a program in Perl that will recreate the calculations Golub et al (1999) did in order to produce Fig. 2 of their paper. Why do this? To find what genes are most predictive of the ALL/AML class distinction? They already did this for us. To verify their answer? Commendable, but that's not our official job. To learn how their method works? Excel is a better tool for that task.

I can think of three better reasons to write this program. First, with the program in hand, you will be able to go beyond the Excel program we did in class, seeing how Golub et al ***used*** the class predictor to work on data from unknown patients (a task for later). Second, it gives you an opportunity to reinforce what you learned with the Excel program by doing the same steps in a different environment. Finally, this is your chance to write the logic of a program almost from scratch. I've given you just the shell of a program, one that takes care of the behind-the-scenes dirty work (like reading in data) but leaves for you to choose what statistical operations should take place when.

The starting point is available on the web, from today's calendar entry or from the **Program** section of the web page for this unit (click on Class_predictor.pl).

## II. Overview of program shell

Download Class_predictor.pl and examine the most comprehensible parts: the primary documentation, the **Files** section and the **Main Program**. When you reach the **Main Program**, you'll find that there's nothing much there except two holes for you to fill in. Run the program to confirm that nothing useful is produced.

Well,... the program is not ***totally*** devoid of content. The **Main Program** already calls a subroutine that sounds like it has something to do with reading in the data, and scrolling down to the **Subroutine** section, you'll find that the subroutine actually exists and is stuffed with statements that seem reasonable. But what exactly is read in and in what form?

**SQ1. Have the program print out the data from the first line of the training set (open up the training set file so you know what to expect). The most straightforward way to do this is through a loop.**

You're given a suggestion in the **Main Program** to make good use of a subroutine with the provocative name, *Get_data_from_line*. Take a look at that subroutine (the documentation may be all you need to see).

**SQ2. What data does it get? Have the program print out the values held in each key variable and compare it to the training set.**

## III. Outline of your additions to the shell

Think back on what you did in Excel to calculate the measure of correlation for each gene, i.e. a measure of the degree to which expression values for a gene fit the distinction between ALL and AML.

**SQ3. Translate the sequence of steps you took into a list of subroutine calls and place these calls into the appropriate place in the Main Program. Your procedure should be designed to produce a measure of correlation for each gene in the training set.**

Hint 1: A loop seems unavoidable

Hint 2: Don't try to put any equations into the Main Program. Leave that sort of thing for the subroutines you invent.

Hint 3: It's not likely that you're going to be able to come up with the ultimate **Main Program** immediately from scratch. Still, get something down, realizing that you'll probably change it before the program is through. Don't get stuck at this point. Jotting down a few feeble lines and moving on is better than staring miserably at a blank screen. Your thoughts will no doubt evolve as you go through the problem.

Hint 4: You might be tempted to write subroutine calls like:

```
Calculate_mean_of_ALL_patients ( );
Calculate_mean_of_ALL_patients ( );
```

but this this is both inefficient and prone to error, because whenever you correct an error in the first subroutine, you'll probably have to remember to correct it in the second one as well. It's better to write a single subroutine that can act on any set of patients, like:

```
mean_ALL = Calculate_mean (@ALL_values);
mean_AML = Calculate_mean (@AML_values);
```

Now you need only get one subroutine to work right.

**SQ4. Perl will object if you call a subroutine that doesn't exist. For now, put in a stub in the Subroutine section for each subroutine you invented, just a header in the format shown below (for a hypothetical subroutine named Calculate_squeegie):**

```
##### CALCULATE_SQUEEGIE     (list of arguments, if any)
#         Documentation, or fill this in later
sub Calculate_squeegie {
}
```

**SQ5. Go through your new Main Program, noting each variable that you invented (there will surely be some). Declare and describe each new variable in the Variable section of the program.**

If you've done all this properly, your program should run just as well as before, i.e. with no errors but no useful output either. Writing an outline of this type gives you a framework in which to work. You can fill in each subroutine, make sure that it works, and make your program functional one step at a time.

## IV. Write the subroutines

Teach the program to do just one subroutine at a time and test the new subroutine before moving on to the next. You know how to do things like calculate means and standard deviations, so the task is merely a question of translating what you know into Perl. If you ***don't*** remember the formula, then here are some useful quantities:

**mean** = (sum of each value) / (number of values summed)

**variance** = (sum of (each value – mean)$^2$ ) / (number of values – 1)

**standard deviation** = Square root of variance

**measure of correlation** = [look it up in Golub et al (1999)]

Hint 1: Test each subroutine in two ways. First, if possible, try it out under simple conditions where you know the answer. For example:

```
print $LF, "mean: ", Calculate_mean ( (1 2 3 4 5) ), $LF;
```

Second, try it with actual data and compare the answer with that calculated by Excel.

Hint 2: Avoid the following common errors in writing a subroutine:

a. Forgetting to grab the argument (if the subroutine is called with one):

```
mean_AML = Calculate_mean (@AML_values);
...
sub Calculate_mean {
   my (@values) = @_;             ← remember something like this
```

Failing to do this (or failing to initialize a variable by some other means) may lead to the following error:

```
Use of uninitialized value in addition (+) at...
```

b. Forgetting to declare new variables with `my`. Failing to do this may lead to the following error:

```
Global symbol "$sum" requires explicit package name at...
```

c. Forgetting to put a semicolon at the end of a statement. Failing to do this will lead to an unpredictable error. In such cases, the given line number may prove useful.

d. Forgetting to return a value at the end of the subroutine, if the subroutine is called as part of an assignment statement. For example:

```
mean_AML = Calculate_mean (@AML_values);
...
sub Calculate_mean {
   ...
   return sum/n;             ← remember something like this
```

Failing to do this won't immediately cause an error, but may lead to weird errors later.

**SQ6. Write each subroutine called in your Main Program, one at a time, testing each.**

## V. Sort the training set and print it or write it to disk

There are at least two useful directions in which this program might proceed: (1) Save the correlation measures and expression data for the best genes (however you define them) for use later in a program designed to assess expression data from patients with unknown types of leukemia, and (2) Save correlation measures and expression data for ***all*** the genes for import into Excel (to allow you to draw the curve shown in Figure 2 of the paper). Either way, it is first necessary to sort the data by correlation value.

Sorting is an obscure process in Perl. Fortunately, you have been provided with a subroutine, `Sort_training_set_by_correlation`, which takes care of sorting for you, so long as you know how to call it.

**SQ7. Examine `Sort_training_set_by_correlation` and then put in your main program a proper call of the subroutine.**

Now take a look at the currently ineffectual subroutine `Write_best_genes`. You need to alter it so that it actually does write out the genes best able to predict the ALL/AML distinction according to the criteria proposed by Golub et al. Output is seldom as simple matter, but the hard part has been done for you in the subroutine `Print_gene_info`.

**SQ8. Examine `Print_gene_info` and then modify `Write_best_genes` so that it uses the subroutine to print the most predictive genes.**

Hint: You'll need two loops for this, the first to print information about those genes that best fit data from ALL patients and the second to print information about those genes that best fit data from AML patients.

**SQ9. Write a new subroutine modeled after `Write_best_genes` that saves to file all genes of the training set (now sorted and given a correlation measure).**

Hint: You'll need to define the name of the output file (in the **Files** section of the program) and open the output file (done for you in a commented-out line in `Write_best_genes`.