CISC106 Fall 2009 Lab03

- Review the code examples from class.
- Some programs below are associated with a question. **Answer the questions** using comments below your code in the m-file.
- The office hours of the TAs and the instructor are on the class website. Visit us!
- **NOTE:** Every function comment section should contain, at a minimum, **three examples** of the function being called and the result of evaluating the call. These examples must include boundary conditions (as discussed in class). Your test files must cover **at least** these exact examples (otherwise, why did you choose them?) and possibly more. Testing is important.

Problems

1. files: fact.m, factTest.m

   Write a recursive MATLAB function **fact** that takes a positive integer `n` as argument, and returns `fact(n)` as defined:

   $$\text{fact(n)} = \begin{cases} \text{n, if } n <= 2 \\ \text{the product of n and fact(n-1) otherwise.} \end{cases}$$

   Thus `fact(4)` evaluates to 24.

   After you write fact, write a test function that tests it. Demonstrate the test function in a diary file.

2. files: expt.m, exptTest.m

   Review your notes from class where we discussed what every recursive function needs to work correctly. Now write a recursive function to calculate the positive integer exponent of an integer, according to the following definition:

   $$expt(base, exponent) = \{ \begin{array}{l} base,\ if\ exponent\ is\ one; \\ base\ *\ expt(base,\ exponent - 1)\ otherwise. \end{array} \quad (1)$$

   Thus `expt(2, 4)` evaluates to 16.

   Hint: When the function calls itself, the base should stay the same every call, but the exponent should decrease (why?).

   After you write expt, write a test function that tests it. Demonstrate the test function in a diary file.

3. files: f.m, showMaxExamples.m

   Matlab offers the ability to return multiple values from a function. Create a function file that contains something like this:

```
function [a b] = f()
  a = 4;
  b = 5;
```

Now in a diary try:

```
x = f()
```

and then

```
[x y] = f()
```

Note that I'm leaving off the semi-colons (why?).

Now look up the built-in `max` function. Call `max` with the following examples and examine the output:

```
>> max([1 7 5])

>> max([1 7 5], [6 8 4])

>> max([1 7 5; 6 8 4])
```

Write a function **showMaxExamples** that prints the max function examples and the answers to them. Create a comment line (or multiple lines) in your function directly above each call to the max function, and **Explain** what a call to max evaluates to for the examples listed above.

Demonstrate execution of your function in a diary file.

4. M-files: sumIntsTest.m, sumInts.m, sumOddIntsTest.m, sumIntSquaresTest.m, sumOddIntSquaresTest.m, sumOddInts.m, sumIntSquares.m, and sumOddIntSquares.m

   diary files: one large diary showing all test files run.

   MATLAB provides a function for summing the elements of a vector, but the problem is a good demonstration of traversing (walking through the elements of) a vector with a loop.

   Write a function sumIntsTest that will test a function called sumInts (duh).

   Enter your lab03 directory (how? Did you remember to make it first?). Copy the file sumInts.m from the class website's lab directory by using a Unix command (more practice!) in a shell (your xterm on Strauss is a shell).

   If you don't remember how to use the Unix copy command, look back at a previous lab now (using the command will be on the exams). Here is the directory to copy from:

   ```
   /www/htdocs/CIS/106/jatlas/09F/labs/lab03
   ```

   Now run the code by running the test function that you already wrote.

   Now make three new unit test function files (you should always write the test file first! (why?)) for the functions sumOddInts, sumIntSquares, and sumOddIntSquares described below. Then write the functions themselves, and run your tests in a diary file.

   - sumOddInts will use a for loop to sum all the odd integers between two parameters called "start" and "finish", e.g. `sumOddInts(5,9) -> 21`

- sumIntSquares will use a for loop to sum the squares of the integers from 1 to "finish".
- sumOddIntSquares will use a for loop to sum the odd squares of integers from "start" to "finish", i.e. 16 will be skipped but 25 would be added to the sum.

5. files: sumOddInts2Test.m, sumOddInts2.m

   The function sumOddInts can be correctly written many ways. One common way is to test a number, and add it to the sum if it is odd; another is to start at an odd number, use an interval of 2, and thereby only generate numbers that are odd. Rewrite the function in a new file, sumOddInts2.m, in one of these two ways (but not the same way you did before) and create a new version of your previous unit test file to test the new name.

   Type and demonstrate your function in a diary file.

6. Evaluate the following in the interpreter:

```
>> a = 'asdf'
>> b = 'wer'
>> c = [a;b]
>> c = [a;a]
```

   Do you understand the error? Read section 6.2 through 6.2.4 of your text. Then show in a diary file how to correctly create the matrix that caused the error above.

7. files: myFigurePlot.png

   Evaluate the following in the interpreter:

```
>> a = [0 1 2 3 4]
>> b = [1 4 2 3 7]
>> plot(a,b)
```

   Now execute the command **clf** to clear the currently plotted figure. Now look up the built-in `plot` function and review section 2.11 of your text. Find out how to plot two different series of data **using a single call to the plot function** and then save the figure as a picture with the Matlab print command:

```
> print -dpng myFigurePlot.png
```

   You can look at your image file on Strauss using a web browser to be sure it is correct. Submit a diary file containing all of the commands you executed to create and save your figure, and also submit your myFigurePlot.png.

This assignment is due on Sakai at midnight on October 1st. Please bring a paper copy to class on Friday October 2nd. Be sure that you have a printed copy of your 15 function M-files, one image file, and 7 diary files demonstrating your testing. All must be stapled together, with your name, lab section, and TA name on the top page.

Be sure that you upload a copy of all the MATLAB function, image, and diary files to Sakai. Then, click submit ONLY ONCE.

On the first page of every printed copy for this course, your name, lab section, and TA's name must appear.