Program Documentation and Style Guidelines

by Dr. Usha Chandra

What are programming guidelines?

The purpose of these programming guidelines is to promote the use of good programming style and documentation. Useful programs have much longer lifetimes than the classroom programming assignments. Real life programs have to be modified and updated quite often during their lifetime unlike the programs that you would be developing. It is very likely that the analyst who maintains these programs may not be the one who developed them and hence good style and documentation are very essential for another programmer to understand and work with your programs. The following programming guidelines were created to make your programs simple, consistent and easy to read and deal primarily with the coding phase of the software life cycle. However, coding is the easiest and least time consuming phase of the software life cycle of real world problems. A thorough problem analysis and design should precede coding/implementation phase. Simply stated, the more time you spend on the problem analysis and design phase the less frustrated you will be during implementation and testing phase.

Program Header: Every program that you submit in the lab or in the lecture as programming exercise should have a program header as given below:

```
% Script file: geometry
% Purpose:
%   To calculate area and perimeter of simple shapes.
%
% Record of revisions:
%     Programmer    :   John Tooker
%     Date Due      :   9/8/09
%     Date submitted:
%     References    :   www.mathworks.org
%     Help Received :   Identify all help received and from whom and what
%                       help.
%                       If no help was received, then note that here
% Variables Used/data dictionary:
%   width     - scalar, The width of the rectangle in inches
%   length    - scalar, The length of the rectangle in inches
%   area      - scalar, The area of the rectangle in square inches
%   perimeter - scalar, The perimeter of the rectangle in inches
%
% Assumptions:
%   For example,
%   For this program to generate correct results, the vectors A, B and C,
% should be of the same size.
```

What are the program documentation guidelines?

1. All functions should begin with a preface of comments as given below: The writing of these expressions should be clear and concise. The goal is to communicate to another programmer two things:
    - What must be true in order for that programmer to use the function; and
    - What work the function will accomplish.

Not such a good design:

```
function [diameter] = find_diameter( radius)
%  Purpose:    computes the diameter of a  circle given the radius
%  precondition: radius >= 0
%  postcondition: returns diameter, if radius > 0
   Diameter = 2 * radius;
end
```

The programmer who calls a function is responsible for ensuring that the precondition is valid. But if you do not specify a precondition, all bets are off and you are responsible. When you write a function, you should make every effort to detect when a precondition has been violated. If you detect that a precondition has been violated, then print an error message and halt the program.

a good design:

% Purpose: computes the diameter of a circle given the radius
% precondition: radius >= 0
% postcondition: returns diameter, if radius > 0, **otherwise prints an error message**

When you write a function, you should make every effort to detect when a precondition has been violated. If you detect that a precondition has been violated, then print an error message and halt the program rather than causing a disaster as a system or program crash.

2. Logical sections of the code should be commented. Such functional comments should be easily distinguishable from the rest of the code. **Please do not comment each and every MATLAB statement in your program**. Major loops and selection statements should be commented. Comments before the code should be indented at the same level. Comments within selection/loop statements should be indented consistently using a tab or 3 spcaes or whatever you are comfortable with. For example,

```
%make sure the radius of a circle is positive
if (radiusCircle> 0)
    diameterCircle = 2.0 *radiusCircle;
    areaCircle = pi * radiusCircle * radiusCircle;
    circumferenceCircle = 2 * pi * radiusCircle;
```

3. Create a data dictionary/variables used for each program to make program maintenance easier.

What are the Style Guidelines?

Variable Names:

1. Descriptive and meaningful identifier names should be used. `radius, diameter,` and `circumference` are more descriptive than `r, d,` and `c`.
2. Declare global variables in all capital letters to make them easy to distinguish from local variables.
3. MATLAB is case sensitive. We recommend that variable names and function names begin with a capital letter. In MATLAB, you cannot get into trouble if you use this convention since so many common variable names are predefined constants or functions in Matlab, e.g., min, max, pi, etc. Also, mixing up cases results in different occurrences of variables.

```
radius = 2.4;  % radius of what  - a ball, a globe, earth, moon
circleRadius = 2.4;  % much better puts it in context
```

4. Variables should be initialized at the appropriate place.

Code Segments:

5. Use a semicolon at the end of all MATLAB assignment statements to suppress echoing of assigned values in the command window. If you need to examine the results of a statement during program debugging, you may remove the semicolon and replace them before submitting your program for grading.
6. Use parenthesis as necessary to make your equations clear and easy to understand.
7. Use blank lines between logical sections of the code to enhance program readability.
8. Consistent indentation conventions (like 3 spaces or a tab) are to be used for nested statements in a block.
9. Leave blank lines between functions.
10. Align and indent the body of the function.
11. Do not write functions that are more than 25 lines.
12. The indentation in your program should reflect the organization of your code.

```
if (x < 0)
      y = abs(x);
else
      y = -x;
end
```

13. Always indent code blocks in **`while`** and **for** constructs to make them more readable.

Files and Input/Output:

14. Save ASCII data files with a "dat" file extension to distinguish them from the M-files that have a file extension of .m
15. The diagnostic and prompting messages should be self-explanatory.

Not such a good prompt:

```
Enter an integer
```

a good prompt:

```
Enter the radius of a circle in inches:
```

16. Format all the output statements so that they can be easily understood.
17. Always include the appropriate units with any values that you read or write in your program.