C PROGRAMMING STANDARDS

COMPUTER SCIENCE 359 – FALL 2012

1. INTRODUCTION

Why do we need programming standards? This document is contains C programming standards, but this information from the official Java website is still very much applicable:

> Code conventions are important to programmers for a number of reasons:
>
> - 80% of the lifetime cost of a piece of software goes to maintenance.
> - Hardly any software is maintained for its whole life by the original author.
> - Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
> - If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

Further:

- You gain experience using programming standards representative of those in the operational Air Force and other professional environments.
- Since your work will be graded by someone other than yourself, it is imperative that you do all that you can to ensure your code is understandable. If your instructor cannot understand it, it will not be graded as correct.
- You avoid penalties on each programming exercise for not adhering to these standards!

2. SOURCE FILE (*.C) STRUCTURE

2.1. The elements of your C source files should be arranged in the following order:

a. file header

b. #include preprocessor directives

c. main function (if applicable)

d. other function definitions

3. HEADER FILE (*.H)STRUCTURE

3.1. The elements of your C header files should be arranged in the following order:

e. file header

f. #include preprocessor directives

g. #define preprocessor directives

h. global variables (if absolutely necessary; avoid global variables if at all possible)

i. type definitions

j. function prototypes

4. FILE HEADER

4.1. Each C source code file shall have a descriptive comment at the beginning (file header):

4.1.1. Each file header comment will contain at least the following information:

```
/**
 *   File: File_Name_Here
 * Author: Your Name Here
 *
 * Description: Description of contents of file here.
 */
```

4.1.2. The main file header comment will also contain your detailed documentation statement:

```
/**
 *   File: File_Name_Here
 * Author: Your Name Here
 *
 * Description: Description of File Contents Here
 *
 * Documentation Statement: Documentation Statement Here
 */
```

5. FUNCTION HEADER

5.1. The C programming language does not have the exception detection and handling functionality of Java which means more of the burden of writing reliable code falls upon the programmer. Thus, it is important that each function clearly state required *pre-conditions* and guaranteed *post-conditions*. These are in essence a contract between functions/programmers.

5.1.1. Pre-conditions are requirements that must be satisfied *before* a function is called. If the pre-conditions are not satisfied, the function's behavior is not guaranteed.

5.1.2. Post-conditions are conditions that are guaranteed to be true after a function executes *if the pre-conditions were satisfied* when the function was called.

5.2. Each function header comment will contain at least the following information:

```
/**
 * Description: Description of function here.
 *
 * Input: Description of parameters here.
 *
 * Result: Description of results/return value here.
 *
 * PRE: Description of required pre-conditions here.
 *
 * POST: Description of guaranteed post-conditions here.
 */
```

6. File, Function, Variable, and Constant Names

6.1. C programs historically do not use the same mixture of upper and lower case within an identifier as Java programs. Most identifiers are all upper-case or all lower-case with underscores used as delimiters. Although this is mostly due to historical reasons, we will adhere to this convention.

6.2. File names will be all lower-case letters and single-word when possible. When necessary, multiple words in a file name will be delimited by an underscore.

6.2.1. Good Examples: main.c, user.h, user.c, linked_list.h, linked_list.c

6.2.2. Bad Examples: Main.c, USER.H, USER.C, LinkedList.h, linkedlist.c

6.3. Function names will be all lower-case letters and multiple words will be delimited by an underscore. In general, function names will be verbs.

6.3.1. Good Examples: load_tractor, is_tractor, locate_tractor

6.3.2. Bad Examples: loadTractor, LoadTractor, IWISHIHADATRACTOR

6.4. Variable names will be all lower-case letters and multiple words will be delimited by an underscore. In general, variable names will be nouns.

6.4.1. Good Examples: count, largest_tractor, num_tractors

6.4.2. Bad Examples: Tractor, TRACTOR, Tractor_Count

6.4.3. Exceptions to this rule can be made for standard acronyms such as GPA.

6.5. Constant names will be all upper-case and multiple words in a constant name will be delimited by an underscore.

6.5.1. Good Examples: MAXIMUM, NUM_TRACTORS

6.5.2. Bad Examples: Maximum, numTractors, NUMTRACTORSINIOWAANDMISSOURI

7. GENERAL

7.1. Indentation and Braces

7.1.1. All code will be properly indented to clearly show the logical structure of the code.

7.1.2. Opening braces can be on the same line as the block being opened or on a new line:

```
for( int i = 0; i < N; i++ ) {
  System.out.println( i );
}
```

or

```
for( int i = 0; i < N; i++ )
{
  System.out.println( i );
}
```

You may choose either style, but you must be consistent throughout your code.

7.2. Commenting and Whitespace

7.2.1. In addition to class and method header comments, inline comments should be included in your code where necessary.

7.2.1.1. Variable declarations should have a truly descriptive name or a comment explaining the purpose of the variable.

7.2.1.1.1. Good Examples:

```
private int count_tractors, tractor_weight;
private int count; // Count of tractors processed so far.
private int weight; // Weight of current tractor.
```

7.2.1.1.2. Bad Examples:

```
private int count; // Without comment, what is being counted?
private int weight; // Without comment, weight of what?
```

7.2.2. Particularly complex code segments should have comments. These comments may need to explain both what is being done and why it is being done at this point in the code.

7.2.3. Whitespace (spaces and blank lines) should be used consistently to enhance the readability of your code.

7.2.4. Especially long lines of code should be broken and indented to be easily readable. No single line of code should exceed the width of a standard laptop screen, generally about 100 characters.

7.3. Other Issues

7.3.1. Functions should be logically cohesive. In other words, all code in a given function should work to perform a single, narrowly focused task.

7.3.2. Use of “magic” numbers should be avoided and defined constants used instead. For example, define and use a constant named ALPHABET_SIZE instead of the number 26.