SCHEME PROGRAMMING STANDARDS
COMPUTER SCIENCE 359 – FALL 2012

1. INTRODUCTION

Why do we need programming standards? This document contains Scheme 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 STRUCTURE

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

a. file header

b. main function (if applicable)

c. other function definitions

d. testing function calls

3. FILE HEADER

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

3.1.1. The main 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.
;
; Documentation: Documentation statement here.
```

3.1.2. If your project contains multiple files, each file will have the above information in a header comment except the documentation statement.

4. FUNCTION HEADER

4.1. The Scheme 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.

4.1.1. 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.

4.1.2. 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.

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

```
; Function: (function-name arg1 arg2 … argN)
; Description: Description of function and arguments here.
; PRE: Description of required pre-conditions here.
; POST: Description of guaranteed post-conditions here.
; Use: Example showing proper use of the function.
```

4.3. An example function header is provided here:

```
; Function: (count-atoms lst)
; Description: Counts all atoms in a given list, including
; atoms in nested lists.
; PRE: The argument lst is a valid Scheme list.
; POST: The value returned is the number of atoms in the list.
; USE: (count-atoms '(1 2 (3 4) 5 (6 7 8))) ; expected result is 8
```

5. FUNCTION AND ARGUMENT NAMES

5.1. Scheme programs historically do not use the same mixture of upper and lower case within an identifier as Java programs. Most identifiers are all lower-case with dashes used as delimiters. Since this is mostly due to historical reasons, we will also use camel-case.

5.2. Function names will begin with a lower-case letter and multiple words will be delimited by a dash or upper-case letters.

5.2.1. Good Examples: `count-atoms`, `countAtoms`

5.2.2. Bad Examples: `CountAtoms`, `COUNT_ATOMS`

5.3. Predicate functions (those that return only a true or false value) will end with a `?`.

5.3.1. Examples: `operator?`, `isEven?`, `isEmpty?`

5.3.2. It is not mandatory, but such function names often begin with the word "is", as shown.

5.4. Argument names will begin with a lower-case letter and multiple words will be delimited by a dash or upper-case letters.

5.4.1. Good Examples: `lst`, `character-list`, `characterList`

5.4.2. Bad Examples: `LST`, `CharList`, `Char_List`

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

6. GENERAL

6.1. Indentation

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

6.1.2. The Racket environment does an excellent job indenting Scheme code. Under the Racket menu are Re-indent and Re-indent All options. Be sure to use these before submitting your code.

6.2. Commenting and Whitespace

6.2.1. In addition to file and function header comments, inline comments should be included in your code where necessary.

6.2.2. 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.

6.2.3. Whitespace (spaces and blank lines) should be used consistently to enhance the readability of your code. Do not write entire functions all on one line unless they are very, very short.

6.2.4. 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.

6.3. Other Issues

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

6.3.2. Use helper functions to decrease the size of functions.