University of North Carolina at Greensboro
CSC 330: Advanced Data Structures
Prof. Stephen R. Tate

Handout 2
August 26, 2009

CSC 330 C++ Coding Style Rules

Major Rules

Each violation of these rules will count 15 points off your final grade on the assignment. That means that if you have 4 global variables in your program, I'll immediately knock of *60 points*.

1. *No* global variables.
2. *All* data elements in a class should be private.
3. Never put executable code in a `.h` header file unless it either (a) is in a template class or (b) is a very short function that could benefit significantly by "inlining".
4. Never `#include` any file *except* a `.h` file.
5. Required comments at the top of every file include: your name, a brief (one line) description of what is in the file, and the last date modified.

Other Rules

These additional rules are not as severe as the rules above, but violations will still result in some points being taken off.

6. Proper indentation is an absolute must! Use at least 2 characters for each indentation level, and at most 8 (I think 4 gives a nice indented structure, although I leave this up to your personal taste).
7. Lines of source code should be at most 80 characters long — lines should not "wrap around" on printouts.
8. Use whitespace and blank lines to increase readability.
9. Use meaningful but concise names for classes, methods, and variables. Class names and variables should be nouns and the beginning of new words in a multi-word name should start with a capital letter (Examples: `tree`, `binaryTree`, `binarySearchTree`, `rightChild`, `vertexList`, etc.). Method and function names should be verbs, start

with a lower case letter, and start subsequent words with upper case letters (Examples: `setData()`, `isEmpty()`, and `getNext()`). Local variables to a function should also have meaningful names, with an exception being for simple loop variables (`i` is ok for an integer loop, `cp` for a character pointer, etc.). Note that names from the STL or in the book don't always strictly follow these guidelines, but in your own code please try to be consistent in following these rules.

10. Use modular, multi-file construction with a `Makefile`. Files to implement a class `myClass` should be called `myClass.h` and `myClass.cpp`.

11. Use constructors and destructors to make sure objects are initialized properly and dynamic storage is returned properly (and be careful to include copy constructors and assignment overloads where necessary).

12. Avoid the C preprocessor — use `const` and `inline` as provided by C++ instead.

13. Use operator overloading with care — using overloading for well-understood operations (like `++` for iterators) is fine, but if you get "creative" in using operator overloading it just makes your code hard to understand.

14. Variable initialization/declaration: Declare variables as close to their first use a possible, and initialize them in the declaration. Loop variables that are only useful inside the loop should be declared inside the `for` statement. Note that the book does not do this — they use the older "C-style" declarations at the top of a function/block. This is really "old-school" and should be avoided.

15. Large objects can be passed as arguments as a `const` reference — this avoids making a local copy, and still protects the integrity of the data.

16. Use `const` wherever appropriate — this helps the compiler catch mistakes before they happen.

17. Use parentheses for grouping expressions, especially in cases where precedence is not obvious.

18. Put comments before each function or method, and at the beginning of each source file. Your comments should be in "Doxygen style", which allows a description of your class to be extracted and formated in HTML. The example code for these style guidelines shows how these comments should look.

19. Comment unclear parts of the code.