

C and C++ Code Preparation

1. Properly format your code:
 - a. Your code should follow a consistent standard (e.g., brace placement, spacing, etc.)
 - b. Be sure to follow naming the following conventions
 1. Variable and method/function name are lower case for the first word and have the first letter in upper case for all other words (e.g., childName). **All** variables must have associated explanatory comments at their declaration.
 2. Class names have the first letter in upper case for all words (e.g., DatabaseAccess)
 3. Constants are all uppercase.
 4. Justify all numeric values with comments
 5. Do not use underscores.
 - c. Do not use tabs.
 - d. Always use braces on `for`, `while`, `if`, etc. (We didn't do this in the book for space purposes, but you need to.)
 - e. Add the following to the beginning of all of your source files:

```
/*  
 *  
 * Author:      Jeff Donahoo  
 * Assignment:  Homework 1, Problem 3  
 * Class:       4321, Fall 2001  
 * Date:        1/2/2001  
 *  
 * This program does stuff.  
 *  
 */
```
 - f. Add the following to each function, including `main()`

```
/*  
 * myFunction:  Function to do stuff  
 *  
 * param1:     The incoming parameter for stuff  
 *  
 * returns:    Number of things on success; -1 otherwise  
 *  
 */
```
 - g. Liberally comment.
2. Include only necessary files. Provide a comment justifying each inclusion. Do not include `.c` files!!
3. Compile your program with the following options `-Wall -ansi -pedantic -std=c99`. Fix any errors and warnings issued by the compiler.
4. Test your code with `valgrind`. Eliminate all self-inflicted problems identified by `valgrind`. Provide justification in your code comments for any problems outside your control such as the standard library leaking (see <http://cs.baylor.edu/~donahoo/tools/valgrind>).
5. Review a hardcopy of your code, correcting questionable code. Consider the following questions:
 - a. Can all of the numeric constants in the code be justified? `char x[37]` typically cannot be justified. `char x[BULENGTH]` can.
 - b. Is the flow of the code easily understandable? Are variable and function names meaningful? Would you want to be given this code for maintenance and modification? If no, why not? How can it be fixed?
 - c. Is the code properly designed and commented?
 - d. Will the program properly terminate in all cases?
 - e. Try all of the timing "What Ifs" that you can think of. What if the other side sends half the expected message, waits for a few seconds, and then sends the rest?
 - f. Does this program leak memory?
6. Check your comments for spelling and grammatical errors!
7. Staple your hard copy.