Coding standards - CS240/CS301

1. Good programmers write beautiful code. Pay attention to readability. You want me to be able to read and understand your code, and you want to be able to read and understand it, too.

    1.1. Choose meaningful names for quantities such as variables, constants, classes and methods.

    1.2. Choose an indentation style and stick with it. Xemacs understands how to indent a program. Take advantage of its features. Hitting the tab key at the beginning of a line indents it properly. Typing "Esc x indent-buffer" reindents the entire file.

    1.3. Separate your code into logically related chunks using blank lines.

    1.4. Put **two** blank lines between method definitions.

2. Good programmers write understandable code. Documentation is vital. It should be written first, before you begin coding.

    2.1. If the name of a data member or local variable is not completely self-explanatory, write a comment that describes what it will be used for. (This is not necessary for loop control variables.)

    2.2. Every method or function you write should begin with a comment that describes the following:

    > the purpose of the method,
    >
    > what the method expects when it is called (preconditions),
    >
    > what the programmer can expect when the method terminates (postconditions),
    >
    > what each parameter is used for, and
    >
    > what the method returns, if anything.

    When documenting class methods, these comments should appear both in the header file where the prototypes are declared and in the implementation file where they are defined. (The header file for a class is usually available for inspection by the authors of client programs; the implementation file may not be, but you want to be able to easily refer to the description of the method while you are writing or modifying its definition.)

    2.3. Complex methods should contain internal documentation. If there is a section of your code that you spent more than thirty seconds thinking about, it needs a comment. (I'm exaggerating a little bit, but not much. You'll see what I mean ten years from now when you try to reuse a bit of code you wrote in one of your courses. Don't laugh. If you stay in this business long enough, it's inevitable.) The purpose of loops and conditions are frequently good candidates for internal comments. Design decisions (why one algorithm was chosen over another, why a certain data structure is appropriate, why a file format or input format was chosen, etc.) should also be explained in the code.

2.4. Every class you write should begin with a comment that describes what the class represents and/or does. Every class data field should have an explanatory comment. As stated in 2.2, every method or function also needs to be commented.

2.5. Comments on methods and classes should NOT reveal implementation details. They should provide a general description of what the methods and classes do or represent. Private data fields and methods should not be referred to in comments.

3. Good programmers write maintainable code.

3.1. Use defined constants for widely used values that may be subject to change, or where using a meaningful name would improve the understandability of code when used in place of a cryptic literal value.

3.2. Parallel arrays are seldom the easiest or safest way to deal with several pieces of data of different types that are logically related. Learn how to construct simple structs or classes for data aggregation purposes. You can't succeed in computing if you are afraid of the power that your tools provide you.

3.3. Big design decisions should be hidden inside classes. Thus, if the design decision needs to be reconsidered, only the class that hides the decision needs to be rewritten. Think carefully about what public methods your classes provide. Well-hidden implementations are easy to modify. Public interfaces are difficult to modify.

4. Good programmers write testable code.

4.1. Determine the interface your class presents to client programs first. In C++, you can write the header file at this point.

4.2. Write your test driver next. If you understand how to test your class, you understand how it needs to behave.

4.3. Now, write your class implementation and test it.

4.4. Regression test after changes. Changes sometimes have unintended side effects. Testing the entire behavior of your program after a change catches these problems. (A cautionary tale: Version 4.10 of Kawa, a Java IDE, had a broken "Save as" capability. This was clearly a result of failure to do adequate regression testing on the other bugs that v4.10 was intended to fix.)

5. Good programmers write well-designed, object-oriented code.

   5.1. Classes should represent a single construct or object. Having several small classes where each class represents a logical entity is better design than having a single class that combines unrelated functionality together.

   5.2. Methods should perform a single action. If more than one action needs to be performed, several simple methods should be created, where each method performs a single action, and then all of those methods should be called sequentially. For example, if you want to sort an array and then print it, two methods should be created (one for sorting and one for printing).

   5.3. While accessor methods are appropriate, methods should not return references or pointers to private data, nor should they return copies of collections of private data. If manipulation of the private data is needed, the class should have a new method created that manipulates the private data and then returns the required result.

Now, some specifics about the assignments you will be doing for our course:

6. Every programming assignment should include a README file containing your name and a description of the work. Submissions without a README file will not be graded.

7. Most professors prefer not to receive programming projects by email. You will receive specific directions in the assignment on how to turn in your work. Unless explicitly allowed, assignments submitted by email will be ignored.

8. Putting the `main` function in a class definition file will result in lost points.

9. You need to have comments in each header file explaining what the class does. Each datum and method should have a brief explanation. In particular, for methods you should explain what the arguments to the method are, what the function does, and what the function returns if it returns something. Method descriptions may alternatively go in the class implementation file (.cpp/.cc/.c) immediately after the method header but before the code that implements the header.

   Not commenting will cause you to lose points on assignments. You should comment anything that is not particularly obvious at a high level, meaning explaining what code does logically not how it's done.

10. All files must have your name(s) at the top.

11. Do not include .cpp/.cc/.c files using the `#include` directive.

12. You may not get solutions to either the programming assignments or to the homework assignments from the web. That is a violation of the honor code. You may, however, use the web as a reference guide; for example, you can use it to find out information about what C++ functions are included in particular libraries or what a debugger message means.