CS 138: Coding Conventions

January 27, 2010

1 Purpose

When working on a large project, consistency is important. A decent coding convention helps make code clearer and ultimately makes code easier to debug and maintain. More importantly, it improves the readability for *other* people, who in this case are going to be the TAs grading you. When in doubt, easily readable code tends to be favored over spaghetti.

The coding convention presented here will allow you to uniquely identify global functions, member functions, class names, local variables/function arguments and global variables at a glance. Furthermore, these coding conventions will provide some protection from common mistakes that lead to tricky run-time errors. These are strong suggestions, but only suggestions - if you have an insurmountable personal bias about conventions you already use, feel free, but make sure it's clear. (16 letter Hungarian prefixes or writing all of your code on one line, for example, are bad ideas.)

2 Naming Conventions

Classes

Capitalize the first letter of the class name as well as the first letter of words in the class name.

Examples: `HelloWorld, MyClass`

Local Variables/Functions

Capitalize the first letter of the words in the variable name *except* the first letter of the variable name.

Examples: `helloWorld, myLocalVariable`

Member Variables

Same style as local variables, except an underscore is prepended to the variable name.

Examples: `_helloWorld, _myMemberVariable`

Name Length

Opt for longer and more descriptive names, rather than short and ambiguous ones (except for loop variables, where "i" works just fine). For instance, `totalCost` is a better name than `tc`.

3 Formatting Conventions

Brace Placement

The opening brace can appear either on a new line below the if, loop construct or function name or on the same line. The ending brace should be on a line by itself (example below).

Indenting

Indent the code between two braces with a tab. Remember that when indentation is done well, brace position becomes implied and code becomes much more readable.

```
if( <condition> ) {
    while( true )
    {
        ...
    }
}
```

Comparison Equal (==)

Have the function or constant on the left side of the `==`, as this will lead to a compile-time error if you accidently type `=` instead of `==`. For instance, use `10 == i` instead of `i == 10`.

4 Java Language Conventions

Imports

Err on the side of caution when importing - if you only need one class from a package, `import` only that class instead of the entire package.

Constants

Define interfaces in which you define constants and nothing else - this will give you an easy way to centralize your constants into a single location.

Packages

Organize your code into packages. For clarity, all related classes should be grouped together in a package. This allows for good software design practices such as encapsulation because you can protect access to methods and member variables appropriately.