Code conventions

Code conventions are essential especially in developing software with multiple developers working on it at the same time. One of the reasons for this is because each person has a different style of coding, just like a language, where different people have different writing styles.

Having this form of consistency does bring some advantages to the table:

* Any programmer will be able to get into the code that is being developed and would have a clear understanding on what is going on in the code
* Less mistakes can be made because we have consistency
* Should the project be developed further in the future, by ensuring code consistency, one can get a good grasp on the code easily.

The following conventions have been defined for this project:

* Header files
  + Avoid the use of inline functions as they are quite vague at times
  + The order of parameters in the function should be input parameters, followed by output parameters
  + Avoid the inclusion of namespaces in the header files
  + Header file format should be .h
  + Include the C and C++ libraries first before including user defined header files
* Code files (CPP files)
  + Avoid the use of static and global variables as much as possible
  + The use of classes is encouraged over structures. Data variables should be made private.
  + Avoid operator overloading
  + Functions used should be short and not be overly complicated
  + Avoid the use of exceptions
  + When doing type casting, use static\_cast
  + When doing sizeof, run it on the variable itself, not on it’s type
  + Be careful when using a C++11 language extension, make sure the compiler will be able to compile the application
  + When coding, it is good to compile and test as every function is developed
  + Avoid the use of unstructured programming
* Naming variables
  + Avoid the use of abbreviations, unless it is a common one like max for maximum and min for minimum (for example)
  + No underscores in variable names instead separate words with the use of capitalisation of the first character of each word (like myVariableName).
  + Apply the same rule when preparing function names and class name, as per variable names
* Commenting
  + It is essential to write the comment that describes what the file is at the top of the file
  + For each function in the header file, comment to tell what the function’s purpose is
  + When declaring classes, comment on the purpose of the class
  + Comments are generally not necessary for variable names as they are quite self explanatory, however if necessary then it is OK to have some comments to describe the purpose of a variable
  + In the implementation, have comments that tell what the implementation code is trying to do in logical positions. It is not necessary to have comments on every line
  + Try not to fit long comments on a single line. Instead split them up into multiple lines
  + Comments should have clear punctuation and grammar and should be meaningful
* Formatting
  + Code should be clearly indented
  + Use tabs instead of spaces for indentation
  + Else keyword should be on a new line
  + Do not have spaces around a period or arrow for pointers
  + When writing blocks that make use of the curly bracket( {} ), ensure the open and closing brackets are on their own line, for example:

int main ()

{

cout << endl << “Hello” << endl;

return 0;

}

By maintaining to these conventions, we can make the development process a smoother experience.

<http://google-styleguide.googlecode.com/svn-history/r97/trunk/cppguide.xml>

http://www.possibility.com/Cpp/CppCodingStandard.html