Team Rancor Coding Standards

Naming

In general: names should be descriptive without being too long/ redundant. No abbreviations should be used unless it is done sparingly.

Functions:

For function names, use camel case starting with a lower case letter.

ex.

```
void exampleFunction();
```

For class accessors, use `getVar()` and `setVar()` names.

Variables:

For variable names, use camel case. Single worded variable names should not be capitalized.

ex.

```
int exampleVariable;

int example;
```

Avoid using single lettered variables unless they are being used for iteration, etc.

Names for constants should be in all capital letters, separate individual words in the name with an underscore.

ex.

```
const int EXAMPLE_CONSTANT = 0;
```

Member variables should be prefixed with a lower case ‘m’ followed by the name.

ex.

```
int mExampleVar;
```

Static variables should be prefixed with a lower case ‘s’ followed by the name.

ex.

```
static int sExampleVar;
```

Classes:

Class names should use camel case beginning with a capital letter.

ex.

```
class ExampleClass
{
…
}
```

Files:

File names should begin with your initials (capitalized) and the name of the file.

ex.

```
ASExampleFile.h
```

Commenting

In general, when describing a portion of code using a comment put the comment before the portion of code being described.

Single line comments:

ex.

```
//this is a comment (duh)
```

Multi-line comments:

ex.

```
/*
This is also a comment.
(Also duh)
*/
```

Code Style

Tabs vs. Spaces:

Uses tabs or spaces at your own discretion, however be consistent with whatever style you choose.

Whitespace:

In general, no more than 1-2 lines of whitespace should be in between lines of code

Brackets:

Curly brackets ({ }) should be placed one line after methods.

ex.

```
exampleMethod()
{
…
}
```

Error Handling

To handle errors we will use assertions by using the <cassert> (assert.h) library.

The assert function is defined as follows:

```
void assert(int expression);
```

If the argument *expression* is false, a message is written to standard error and the program will terminate.

ex.

Assertion failed: *expression*, file *filename*, line *line number*

The macro is disable if NDEBUG is defined, which allows asserts calls to be used as many times in debugging and then disabled in the main (production) version

ex.

```
#define NDEBUG
```