C++ Programming Style Guide

Adapted from the Cedarville University C++ Programming Style Guide and Dr. Mark Van Gorp's Style Guide.

In General:

- Good code is straightforward and understandable. Clever solutions are great, clever solutions at the expense of understandability are not.

- Good code is written in such a way that it is understandable to someone who has not seen it before.

- Good code has a consistent style.

Layout and Comments:

1. The top of *every* file should have the following comment:

   ```
   /**
   * [Author Name]
   * [File Name]
   **/
   ```

   where [Author Name] is replaced with your name and [File Name] is the full file name (such as project2.cpp).

2. An appropriate amount of comments should be used throughout your code. Although it is most common for programmers to provide too few comments, over-commenting can also negatively impact the readability of the code. It is important to comment on any sections of code whose function is not obvious (to another person). For algorithmic-type code which follows a sequence of steps, it may be appropriate to summarize the algorithm at the beginning of the section and then highlight each of the major steps throughout the code. Comments should be indented at the same level as the code they describe.

3. Once you begin writing your own functions/methods…

- *Most* function declarations should have above them a Javadoc style comment which at minimum gives an overview of what the function does, describes each parameter, describes the return value (if the function is not void), and any pre-conditions. A programmer should be able to use this comment along with the function declaration to be able to use the function knowing nothing about the implementation. This is optional for constructors, destructors, and basic mutators and accessors. The general format to be used is:

```
/**
* A basic description of the function here. Use multiple
* lines if necessary.
*
* @param  firstParam  Description of the first parameter.
* @param  secondParam  Description of the second parameter.
* @return  Description of the returned value.
* @pre  What must be true for the function to run
*       correctly.
**/
int someFunc(int firstParam, int secondParam);
```

For example:

```
/**
* storeData writes the array of students to file. The file
* itself is hardcoded in as students.txt.
* @param  students An array of Student objects.
* @return  1 if file is written successfully, -1 otherwise.
* @param  numStudents Number of students in the array.
* @pre  students[] must be filled with data and numStudents
*       must contain the number of students in the array.
*/
int storeData(Student students[], int numStudents);
```

- All functions should have a return type. For functions not returning a value, use *void* type.

- When including standalone functions in your application file with `main()`, function declarations should always be *above* `main()` and function definitions should always be *below* `main()`.

4. Once you begin writing your own classes…

- If you haven't yet learned to split classes into a separate file, the class definition goes in your application file *above* `main()` and the member function definitions go *below* `main()`.

- Class member function declarations should also be documented with Javadoc style comments as instructed above.

- Once you've learned to move classes to a separate file do so in a logical manner. In general, only one class should be per file, with the class interface in a header file (.h suffix) and the class implementation in a source file (.cpp suffix).

Naming Conventions:

1. Use descriptive names for variables, functions, and objects in your code, such as `xPosition`, `distanceToGo`, or `findLargest()`. The only exception to this may be loop iteration variables, where no descriptive name makes sense. In this case, use the lower case letters beginning with *i* (i, j, k, etc).

2. For variable and function/method names use camel case. The first word should be lowercase, and subsequent words should be capitalized. For example, `upperLimit`, `averageValue`, `makeConnection()`, `addBody()`. Whenever possible, function/method names should be verbs: `draw()`, `getX()`, `setPosition()`.

3. For constants, capitalize all letters in the name, and separate words in the name using an underscore, e.g., `PI`, `MAX_ARRAY_SIZE`. Any numeric constants needed in your code (other than very simple ones like –1, 0, and 1) should be replaced by a named constant.

4. For classes use camel case but with the initial letter capitalized: `DigitalClock`, `BankAccount`, `Shape`.

Statements:

1. Only one statement is allowed per line, and each line of code will be no more than 80 characters in length (to prevent line-wrap). If a statement requires more than one line, subsequent lines will be indented to make it obvious that the statement extends over multiple lines, as shown below.

```
int myFunction (int variableA, int variableB, int variableC,
                int variableD, int variableE, int variableF );
```

2. Braces can be done in one of two styles: the opening brace can be put on the end of the line defining the block, or it can be on a separate line by itself, as shown below. Code inside the braces will always be indented.

```
for (i = 0; i <= maxSize; i++) {
   …
   …
}
```

```
for (i = 0; i <= maxSize; i++)
{
     …
}
```

3. Make ample use of horizontal white space. Always include spaces after commas, and between operands and operators in expressions. Avoid using tabs to create white space.

4. Use vertical white space (i.e. blank lines) to separate your code into “paragraphs” of logically connected statements.

5. Use parentheses liberally for clarity.

Classes:

1. Member variables should always be private, with public or protected accessors and mutators [e.g., `getX()`, `setX()`] provided.

2. Interface files (.h) should always be protected from multiple inclusion using `#ifndef`.

3. A destructor should be included with all classes that use the heap.