1 Introduction

This document describes the coding standard to be used by all program submissions for ECE3090. Consistency of coding style for all students leads to neater and easier to read code, and ease of grading for the instructor and teaching assistant. Submission of code that does not adhere to this standard results in a 10% penalty on grading.

2 Standards

2.1 General

1. No line of code will be longer than 80 characters in length, to prevent lines wrapping in the emacs or vi editors. For both of these linux text editing tools, the default is a window that is exactly 80 characters wide, so if none of the lines wrap when editing in emacs or vi this requirement is met.

2. Each C++ statement will be on a separate line. The only exception is when an if, else, for or while statement has a single statement sub-block these can be on the same line.

Examples:

```cpp
int i = 0 // Right
i = 10; j = 20; // Wrong. Two statements same line
Sub1(k); Sub2(k); // Wrong. Two statements same line
if (done) break; // Right. If statement with single statement sub-block
```

3. Each variable declaration will be on a separate line.

Examples:

```cpp
int c, d; // Wrong. c and d defined on same line.
int a;
int b; // Right. a and b on different lines
```

4. Excepting when used in a switch statement, the open and close curly braces ({ and }) are always on a separate line.

Examples:

```cpp
for (int i = 0; i < MAX_COUNT; ++i)
{
    // Right. Open brace on separate line
    sum += i;
    prod *= i;
} // Right. Close brace on separate line
```
for (int i = 0; i < 10; ++i) { // Wrong. Open brace on same line
    sum += i;
    prod *= i; } // Wrong. Close brace on same line

5. The C++ goto statement is not to be used.

2.2 Commenting

In general, comments should be use liberally throughout the program to increase readability. Specifically:

1. Every variable declaration should have a short, one or two line comment describing the purpose of
   the variable, unless it is a short–lived local variable whose use is obvious from the context. The short
   comment should be on the same line as the variable declaration, unless it is too long, in which case it
   should be on the preceding lines.

   Example:
   
   int averageGrade; // Computed average grade for this project
   // Counts the total number of students completing the project, but
   // does not include those not turning in the project.
   int projectCount = 0;

2. Every function should be preceded by a detailed comment block describing what the function does,
   what the formal parameters are, and what the return value is (if any).

3. Every class declaration should be preceded by a comment block describing what the class is to be
   used for.

4. Unless obvious from context, each if statement should include a one–line comment on the open curly
   brace following describing the TRUE condition and the FALSE condition.

   Example:

   if (iter == students.end())
       { // Student not found, add him
           students.push_back(thisStudent);
       }
   else
       { // Student exists, modify existing data
           iter->grade += thisGrade;
       }

2.3 Naming Conventions

1. **Variable Names.** All variables, including global variables, local variables, and member variables
   in classes will start with a lower case letter, and consist of only alphabetic characters and numeric
   digits. Capital letters are to be used when appropriate between words in a variable name for increased
   readability. The variable name should be descriptive of the use of the variable, excepting for temporary
   local variables where the function is obvious from the context (for example loop index variables are
   commonly a single letter such as i).
Examples:

```c++
int i;
int nextIndexValue;
int sum1;
int loopCount10;
```

2. **Subroutine Names.** All subroutine names, including global routines and member functions in classes, will start with an upper case letter, and consist of only alphabetic characters and numeric digits (although digits should be rarely needed). As in variable names, upper case letters are to be used between words as needed to increase readability.

Examples:

```c++
int ComputeNextIterator()
int Calculate()
int TransmitPacket()
int Dummy()
```

3. **Defined Constants.** All defined constants will be all upper case letters or numeric digits, with the underscore character separating words.

Examples:

```c++
typedef enum { PACKET_RX, PACKET_FIRST_BIT_RX, PACKET_TX}
#define NUMBER_ELEMENTS 10
const int LOOP_COUNT = 100
```

4. **Defined Types.** All user defined types will end start with an upper case letter, consist of upper and lower case letters only, and end in _t_.

Examples:

```c++
typedef double Time_t; // Simulation time
typedef unsigned long SimulatorUid_t; // Unique ID for each event
typedef unsigned long Event_t; // Idenifies events in handler
```

5. **Class Names.** Class names will start with an upper case letter, consist of only alphabetic characters, and include capital letters as needed to increase readability.

Examples:

```c++
class DropTailQueue {

class Ferrari {
```
2.4 Statement Formatting

1. **Continuation statements.** Frequently a single statement is too long to fit within a single 80 column line. In this case, the statement is simply continued on the next one or more lines. Each continuation line must be indented at least four character positions, and more as necessary to increase readability.

   Examples:

   ```
   longVariableName = (anotherLongName * shorterName) + (loopIndex2 * i) + (k * j); // Correct, indented for neatness
   ```

   ```
   for (LongTypeName longLoopIndexName = aLongExpression;
       longLoopIndexName < MAX_VALUE;
       longLoopIndexName++) // Wrong, continuations not indented far enough
   ```

   ```
   for (LongTypeName longLoopIndexName = aLongExpression;
       longLoopIndexName < MAX_VALUE;
       longLoopIndexName++) // Right, indented at least 4 columns
   ```

2. **IF Statements.** The open curly brace following an IF statement must be on the following line, indented by two positions. The subsequent lines must indented an additional two positions. IF statements with only one statement in either the TRUE or FALSE sub–blocks may omit the curly braces. The ELSE statement (if present) must be on a line by itself.

   Examples:

   ```
   if (someCondition)
   {
     // Describe TRUE condition here
     i = k;
     k = i + 2;
   } // Right, curly block indented two spaces, statements two more
   ```

   ```
   if (someCondition)
   {
     // Describe TRUE condition here
     i = k;
     k = i + 2;
   }
   ```

   ```
   else // Right, ELSE statement on separate line, same indent as IF
   {
     // Describe FALSE condition here
     i = k * 2;
     k = i + 4;
   } // Right, closing curly brace lined up with open brace
   ```

   ```
   if (someCondition) // Describe TRUE condition here
   i = k; // Right, single line block need not have curly braces
   ```

   ```
   if (someCondition) i = k; // Right, single stmt may be on same line
   ```
3. **FOR Statements.** The open brace following a `for` statement is indented two columns from the `for` statement itself. Each statement in the sub–block is indented two columns from the curly brace. If the sub–block is a single statement, the curly braces can be omitted and the statement indented two columns, or optionally appear on the same line as the `for` statement.

Example:

```java
for (int i = 0; i < MAX_COUNT; ++i)
{
    // Curly brace indented two characters
    sum += i; // Statements indented another two characters
    prod *= i;
} // Close brace on same column as open brace
```

```java
for (int i = 0; i < MAX_COUNT; ++i) Sub1(i); // Right, single statement
```

4. **WHILE Statements.** While statements are formatted similarly to IF statements, with curly braces indented two columns on separate lines, and the inner statements indented two more columns. If the sub–block has only a single line, the curly braces can be omitted, and the statement may appear on the same line as the `WHILE` statement.

Examples:

```java
while (someCondition)
{
    i = k; // Right, statements indented two columns from open brace
    k = i + 2;
} // Right, close brace lines up with open brace
```

```java
while (someCondition) i = i + 2; // Right, single stmt on same line
```

5. **SWITCH Statements.** The open curly brace for a switch statement will be on the same line as the switch statement itself. Each case statement following is indented two columns from the switch statement. Each statement in the case block is indented two column from the case statement. The closing curly brace is on a separate line by itself, indented two columns from the switch statement.

Example:

```java
switch(someCondition) {
    case 0 : // Right, open brace on same line as switch
        i = k; // Right, case indented two columns from switch
        k = i + 2;
        break;
    case 1 : // Right, case indented two columns from switch
        i = k + 2; // Right, statements indented two columns from case
        k = i + 4;
        break;
} // Right, close brace lines up with case statements
```
6. **Functions.** Since C and C++ do not allow nested functions, all functions start with no indentation at column 0. The open curly brace is on a line by itself immediately following the function header and formal parameters, also in column 0. Any local variable declarations immediately following the open curly brace also start at column 0. One blank line follows the initial local variable declarations (if any). The statements in the function body are indented two columns from the curly brace. Any variable declarations after the start of the statements are indented at the same level as the preceding statement. The closing brace is at column 0.

Example:

```c
void Function1(int arg1, double arg2)
{
    // Right, curly brace at column 0
    int local1 = 0; // Right, local variable at column 0
    int local2;
    local2 = local1 + arg1 + arg2; // Right, indented two columns
    int local3; // Right, variable at same level
    local3 = Function2(local2);
    if (someCondition)
    {
        local3 = 0;
        local2 = local1;
        int local4 = local1 + 1; // Right, variable at same level
        Function3(local4);
    }
} // Right, close brace at column 0
```

7. **Expressions.** Spaces should be used liberally in expressions to increase readability. One space before and after each operator, excepting the increment and decrement operators, leads to easy–to–read expressions. Continued expressions should be indented as far as needed for neatness and readability.

Examples:

```
i = k * 2 + 3 / var1++; // Right, spacing separating terms

i = k*2+2/var1++; // Wrong, crowded together and hard to read

someLongVariableName = anotherLongVariableName * shorterName + anotherName; // Right, indented to line up
```