C++ Style Requirements
Li Liang
CS 162: Introduction to Computer Science II

This document describes the style requirements for programming in C++ in these sections of CS162.

1. Use the following conventions for all variable names:
   a) *Always* begin with a *lower case* letter. Remember also that an identifier cannot begin with a digit
   b) An identifier must consist of letters, digits, or underscores only.
   c) Multiple word variables should have each word *other than the first* start with a *capital* letter. Alternatively, each word should be separated from the next by an underscore (in which case the whole word will be in lower case.) This latter convention is no longer fashionable, however, and I would discourage its use.
   d) Use meaningful words that represent the function of the variable
   e) You cannot use a C++ reserved word as an identifier

2. Place all variable definitions/declarations before the executable statements of the main() program or each function. This means that variable declarations should not be scattered throughout your code! C++ allows otherwise...so take this as a style requirement.

3. Place only one statement per line. C++ allows multiple statements to appear on the same line; however, for style please limit yourself to only one per line.

4. A blank space is required between words in a program line. Always leave a blank space after a comma. Always leave a blank space before and after the following operators: * / + - = << >>.

5. Indent each line of the program except for the curly braces that mark the beginning and end of the main program. All lines between the {} are to be indented three or four spaces consistently. Please note that use of proper indentation is an essential requirement of this class.

6. Use blank lines between *sections* of the program. For example, there should always be a blank line between the compiler directives and the rest of the program.

7. Do *NOT* use a blank line between every line of code; this reduces program readability, instead of enhancing it!

8. When choosing between an if/else control statement and using the conditional operator (?:), always choose the if/else structure. This is required to assist debugging and readability.

9. Indentation for if/else control statements:

```
if (conditional expression) {
     statement;
     statement;
} else {
     statement;
     statement;
}
```

Alternatively, you may use the following:

```
if (conditional expression)
{
     statement;
     statement;
}
else
{
     statement;
     statement;
}
```

Nothing else is acceptable.

10. Indentation for Switch control statements:

```
switch (selector expression) {
     case label1 :
          statement;
          statement;
          break;
     case label2 :
          statement;
          statement;
          break;
     default :
          statement;
          break;
}
```

a) Note: For style always use the default label.

b) Place only one case label per line

c) Always place a break after the last case label -- even though it is syntactically unnecessary. If you decide to add more labels later on ... you just might forget to add the break at that time.

11. Under no circumstances are goto's allowed in this class!

12. Unless you have a good reason for doing otherwise, use local variables (i.e. those declared *within* main or some other function) in preference to global variables (i.e. those declared outside of main or any other function). Another way to say this is: avoid declaring variables at file scope unless absolutely necessary.

13. Indent loops as follows:

```
for (i = 1; i <= some_max; ++i) {
     statement;
     statement;
}

i = 0;
while (i <= some_max) {
     statement;
     statement;
}

do {
     statement;
     statement;
} while (i <= some_max);
```

a) Avoid changing the loop control variable inside of the body of a “for” loop!

14. When writing your own functions, use the following style guidelines:

a) Always use a return at the end of a function; if the function is not supposed to return a value then end the function with:

```
return;
```

15. Functions that do not have any parameters should omit the term "void" inside the parentheses of the function declaration. The use of the reserved word void in this context is no longer fashionable, and should be avoided.

16. Functions that do not return any value to the calling routine must be declared with the "void" return data type:

```
void function_name(parameters);
```

17. Unless you have good reasons otherwise, do not use global variables (those declared at file scope) to pass data to or from functions. Instead, use formal parameters to input the data and output the result. And, use local variables to assist with any intermediate calculations.