C++ Style Guidelines
CS 2308
Molly O’Neil
(adapted from J. Slomka and R. Priebe)

In addition to working correctly, it is important that programs are both readable and maintainable by others. Studies have shown that for large software projects, the cost of writing a program is much less than the cost of maintaining the code over its lifespan. Therefore, it is a matter of economic importance to write clear, consistent code that can easily be understood and maintained by others.

There is no one ‘correct’ style for C++, and therefore consistency in style is more important than adherence to any one particular style. Most large programming groups will have a style manual that must be followed by all programmers.

Below are the style conventions to be used in my courses. In some cases, I present options for students to use; in others, I specify a precise, required style.

Once again, the style you choose to use must above all else be CONSISTENT!

File Layout & Documentation

Header Comments

- At the top of each source code file, place a block of comments listing the name of the file, your name, the course and section number, the assignment number, and a brief description of the included code’s purpose. Here’s an example:

  /*******************************************************************/
  * Student.cpp
  * *
  * Molly A. O’Neil
  * 15 Aug 2015
  * CS 2308-002 – Project #1
  * *
  * A representation of a student
  *******************************************************************/

- #include statements must immediately follow header comments

Function Comments

- At the beginning of each function, place a block of comments listing the function name, the meaning of each parameter and any constraints on its value, a description of the return value (if any), and a brief description of the function’s purpose.
getSubjectGPA: calculates and returns the student’s grade point average across all coursework in specified subject.

subject: the abbreviation for the subject, e.g. “CS”

returns the GPA as a float

```c
float getSubjectGPA(string subject) {
    ...
}
```

- Comments should be used within function bodies to describe what conceptual sections of the code are doing. Do NOT comment every line of code.

Source code lines should be no longer than 80 characters. (*Historical trivia question:* This 80-character maximum code line limit is a standard you will see in nearly every style guide. Where does it come from originally?)

**Whitespace, Indentation, & Spacing**

**Indentation**

- **DO NOT USE TABS!!**
  
  All indentation should be done with spaces, not tabs
  
  - You may use 2-4 spaces as a standard indentation
  - You must be consistent in your choice of how many spaces == 1 “tab”
  - Some editors (e.g., VIM) can alias the ‘TAB’ key on your keyboard to a fixed number of spaces so you don’t have to worry about it. If you use VIM, you can add the following to your .vimrc to do this:
    ```
    set tabstop=2
    set shiftwidth=2
    set expandtab
    ```
  - Indent all lines within a block by the same amount, e.g.:

    ```c
    if(x) {
        if(y) {
            while(z) {
                // more code here
            }
            doSomething();
        }
        doSomethingElse(y);
    } else {
        y = -1;
        doSomethingElse(y);
    }
    x = 10;
    ```
Braces

- I prefer the following style for braces:
  
  ```
  if(a < b) {
    cout << "Hello" << endl;
  }
  ```
  
  However, Gaddis uses and therefore I accept:
  
  ```
  if(a < b) {
    cout << "Goodbye" << endl;
  }
  ```
  
  Whichever braces style you choose, you **must be consistent**!

Whitespace

- Use blank lines to separate sections (#includes, functions, variable declarations from code, logical sections within blocks) of your code

- A space must be placed on each side of a binary operator, e.g.:
  ```
  sum = operandA +_operandB;
  ```

- A space should not be placed between an array name its index, e.g.:
  ```
  foo[x]
  ```
  Not:
  ```
  foo [x]
  ```

- I prefer no spaces around parentheses, e.g., in if() statements:
  ```
  if(x <= y)
  ```
  However, many programmers prefer and I will accept:
  ```
  if ( x <= y )
  ```

Identifier & Function Naming

Names

- Variable/constant names must be meaningful and descriptive
  Exceptions:
  - Loop index variables may be single letters (e.g., i, j, k)
  - Generic or temporary variables may be named, e.g. for pointers, p1, p2, etc.

Capitalization

- Constants must be named in ALL_CAPS with multiple words separated by underscores, e.g.:
  ```
  const double SALES_TAX_RATE = 0.0825;
  ```
Variable names should begin with a lower-case letter. If a variable name is multiple words, I strongly prefer camelCase (following words should be capitalized, no underscores), e.g.:

```
lengthInInches
```

However, I will also accept C-style all-lowercase letters with underscores between words, e.g.:

```
length_in_inches
```

Whichever capitalization style you choose, you must be consistent!

Function names must begin with a lower-case letter and must be in camelCase, e.g.:

```
calculateAverage()
```

Abstract data type (class and struct) names must begin with a capital letter, contain no underscores or punctuation, and use capital letters for the start of each new word, e.g.:

```
struct Pixel
class PageWriter
```

Variables

Few numeric literals should appear in your program. Use named constants, including for array sizes.

Do not use global variables other than constants!

Remember: BE CONSISTENT!