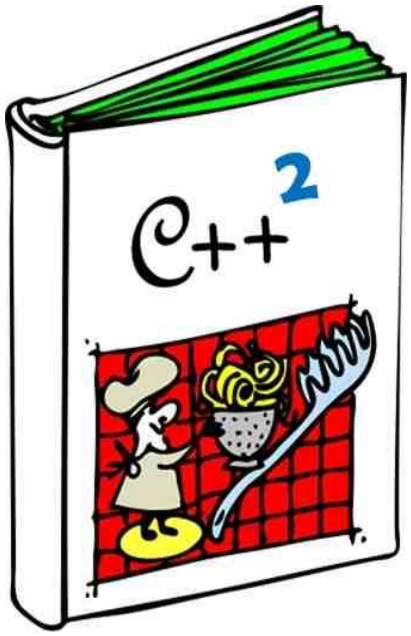
C++ AMP CONTEST

Coding Style Guidelines

It pays to be obvious, especially if you have a reputation for subtlety.

Isaac Asimov

We would like to clarify from the start that this is first and foremost a contest focused on programming prowess and not on unrelenting adherence to some particular definition of "beautiful" code. However, we do value the time of the referees and their general happiness and wellbeing, which makes it necessary to lay out some coding style rules. Whilst completely refusing to adhere to them cannot make you lose the contest if you have a much better solution than all others, some points will be lost if the submitted code makes no effort to at least abide by the following general pointers. Most of what we suggest is in line (up to being identical where adequate) with Bjarne Stroustrup's recommendations presented in [1], which are partially reproduced in [2]. Alternatively, the rules presented in [3] are equally palatable, as are minor variations of these two (not hugely dichotomous) approaches.



General aspects

1) Avoid functions or classes that contain more than 200 logical source lines of code;

2) Try, to the best of your ability, to ensure that any single function or class fits on a screen and serves a single logical purpose.

Preprocessor related aspects

1) Avoid, to the best of your ability, macros; use them solely for source control;

2) **#include** directives should precede all non-preprocessor declarations;

3) Place only **const** variable definitions and inline template function definitions in header files.

Naming and layout related aspects

1) Use indentation and maintain its consistency within the same source file / project;

2) Place one statement per line:

```
float pi=3.14f; c=2*pi*r; a(pi,r);//not OK
float pi = 3.14f; //OK
c = 2 * pi * r; //OK
area(pi, r);//OK
```

3) Use descriptive names for identifiers, which may contain common abbreviations and acronyms;

4) Prefer the **number_of_elements** style of notation when using multiple words;

5) Hungarian notation shall not be used (please!);

6) Only type, template and namespace names start with a capital letter; be parsimonious in your naming (no excessively long names);

7) Underscores shall not be used as the opening character of an identifier;

8) Differentiate identifiers by more than:

- a mixture of case;
- presence/absence of the underscore character;
- interchanging the letter O with the number 0 or the letter D;
- interchanging the letter I with the number 1 or the letter l;
- interchanging the letter S with the number 5;
- interchanging the letter Z with the number 2;
- interchanging the letter n with the letter h (or any other similar exchange.

9) Avoid using all capital letters and underscores for an identifier.

Function and expression related aspects

1) Identifiers in an inner-scope should not be identical to identifiers in an outer scope;

2) Try to include declarations in the smallest possible scope;

3) Magical constants shall not be used (please!).

Following the simple guidelines enumerated above will make your code easier to parse, thus ensuring you will achieve the highest score possible in the subjective evaluation of code quality, and ensuring that once its published other interested people will be able to learn from it, as opposed to being completely discouraged by the difficulty of figuring out even the simplest aspects. We thank you in advance for making everybody's life easier by making your code reader-friendly.

References

[1] B. Stroustrup, *Programming: Principles and Practice Using C++*, 1st ed. Addison-Wesley Professional, 2008.

[2] "Stroustrup: C++ Style and Technique FAQ." [Online]. Available: http://www2.research.att.com/~bs/bs_faq2.html#whitespace.

[3] "Qt_Coding_Style | Qt Wiki | Qt Developer Network." [Online]. Available: http://qt-project.org/wiki/Qt_Coding_Style.