Object-Oriented Programming

Coding Style

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Code

- Developers *read code* much more than they *write code*

- In the long run, there is no "your code" and "my code"

- Often, the code is the only documentation of the design

- Common coding style/coding standard is necessary

Essential Parts of Coding Style

- Indentation
- Newlines
- Whitespace
- Comments
- Naming conventions

Other Parts of Coding Style

- Header and class/method/function comments

- Libraries used, e.g., C++ standard libraries + Boost

- Preferred language statements, e.g., `switch` vs nested-if statement

Characteristics of a Coding Style

- Consistency
- Scalability
- Maintainability

Inconsistency

```
auto value=*begin; // pivot value is the first element

auto left = begin;
auto right= std::prev(end);
while (std::distance(left,right)>0) {

        // move left-to-right while value is greater than elements
        while(value>= *left && std::distance(left, end) > 0){
           if (std::next(left) == end)
                    break;

                left=std::next(left);
          }

        // move right-to-left while value is less than elements
        while (value <*right)
        {
                right= std::prev(right);
        }

// exchange so elements less than value are on the left
// and elements greater than value are on the right
        std::swap(*left, *right);
}


std::swap(*begin,*right); // exchange pivot and final location

// post-condition
assert(std::all_of(begin, right, [right](int n){ return n <= *right; }));
assert(std::all_of(right, end,   [right](int n){ return n >= *right; }));
```

Scalability Problems

```
auto value=*begin; // pivot value is the first element

auto left = begin;
auto right= std::prev(end);
while (std::distance(left,right)>0) {

        // move left-to-right while value is greater than elements
        while(value>= *left && std::distance(left, end) > 0){
           if (std::next(left) == end)
                    break;

                left=std::next(left);
          }

        // move right-to-left while value is less than elements
        while (value <*right)
        {
                right= std::prev(right);
        }

// exchange so elements less than value are on the left
// and elements greater than value are on the right
        std::swap(*left, *right);
}


std::swap(*begin,*right); // exchange pivot and final location

// post-condition
assert(std::all_of(begin, right, [right](int n){ return n <= *right; }));
assert(std::all_of(right, end,   [right](int n){ return n >= *right; }));
```

Difficult to Maintain

```
/************************************************
 *                                              *
 * myCoolFunction                               *
 *                                              *
 * This is my cool function. Isn't it cool?     *
 *                                              *
 ************************************************/

 /*
  * myCoolFunction
  *
  * This is my cool function. Isn't it cool?
  *
  */
```

Problems with a Coding Standard

- Lack of formal training
- Programming language differences
- Difficult to formally define, check, or correct
- Difficult to maintain
- Lots of corner cases
- Religious arguments
- bike shed painting

Indentation & Whitespace

- Indentation must be consistent

- Indentation should appear consistent, even when moved out of the IDE, e.g., Gists, web examples, etc.

- Indentation is based on the "flow of control", not importance/difficulty

Indentation Composition

- How much do you indent for each level? 2? 4? 8?
- What is the indentation made up of? Spaces? Tabs? Tabs/Spaces?
- Problem: Most of the time, you can't tell by looking
- Problem: Lack of understanding of a tab (and tab stop).
- Confusion: Developers who use spaces to indent often have the editor expand tabs to spaces

Inconsistent Indentation

```
// constructor
srcMLOut(TokenStream* ints,
         xmlOutputBuffer * output_buffer,
         const char* language);

    // constructor
    srcMLOut(TokenStream* ints,
                    xmlOutputBuffer * output_buffer,
                    const char* language);
```

```
····// constructor
····srcMLOut(TokenStream* ints,
············xmlOutputBuffer * output_buffer,
············const char* language);

————————// constructor
————————srcMLOut(TokenStream* ints,
————————————————————·xmlOutputBuffer * output_buffer,
————————————————————·const char* language);
```

Indentation Guidelines

- Don't mix tabs and spaces for the indentation on different lines
- Don't mix tabs and spaces for the indentation on the same line
- Many of the advantages of tabs have been replaced by IDE features. Advice: Use spaces (llvm coding standard), especially with code that may get used in multiple projects
- If you are going to use tabs, only use them for flow-of-control indentation and not to line things up
- Indentation level: Linux is 8, llvm is 2(!), 4 is good

Know Your Editor/IDE Settings

- Sublime Text

- Xcode - Preferences/Text Editing/Indentation

- Visual Studio

Newlines

```
// number of unprocessed characters [cursor, cursorEnd)
size_t unprocessed = std::distance(cursor, cursorEnd);

// move unprocessed characters, [cursor, cursorEnd), to start of the buffer
std::copy(cursor, cursorEnd, buffer.begin());

// reset cursors
cursor = buffer.begin();
cursorEnd = cursor + unprocessed;

// read in whole blocks
ssize_t readBytes = 0;
while (((readBytes = READ(0, static_cast<void*>(buffer.data() + unprocessed),
    std::distance(cursorEnd, buffer.cend()))) == -1) && (errno == EINTR)) {
}
if (readBytes == -1) {
    // error in read
    return -1;
}
```

- Single newlines are sufficient. Multiple newlines are not needed

- Again, consistency

Whitespace

```
// number of unprocessed characters [cursor, cursorEnd)
size_t unprocessed = std::distance(cursor, cursorEnd);

// move unprocessed characters, [cursor, cursorEnd), to start of the buffer
std::copy(cursor, cursorEnd, buffer.begin());

// reset cursors
cursor = buffer.begin();
cursorEnd = cursor + unprocessed;

// read in whole blocks
ssize_t readBytes = 0;
while (((readBytes = READ(0, static_cast<void*>(buffer.data() + unprocessed),
    std::distance(cursorEnd, buffer.cend()))) == -1) && (errno == EINTR)) {
}
if (readBytes == -1) {
    // error in read
    return -1;
}
```

- Space around operators
- Spaces before parentheses after keywords

Comments

```
// number of unprocessed characters [cursor, cursorEnd)
size_t unprocessed = std::distance(cursor, cursorEnd);

// move unprocessed characters, [cursor, cursorEnd), to start of the buffer
std::copy(cursor, cursorEnd, buffer.begin());

// reset cursors
cursor = buffer.begin();
cursorEnd = cursor + unprocessed;

// read in whole blocks
ssize_t readBytes = 0;
while (((readBytes = READ(0, static_cast<void*>(buffer.data() + unprocessed),
    std::distance(cursorEnd, buffer.cend()))) == -1) && (errno == EINTR)) {
}
if (readBytes == -1) {
    // error in read
    return -1;
}
```

- Code is written in paragraphs, chunks/hunks of code
- Comments appear before, indented the same, space after "//"
- Prefer line comments unless multiple lines
- Tend towards doxygen comment markup
- There are many styles of Doxygen markup, so follow one consistently

Consistent, Scalable, Maintainable

```
// pivot value is the first element
auto value = *begin;

auto left = begin;
auto right = std::prev(end);
while (std::distance(left, right) > 0) {

    // move left-to-right while value is greater than elements
    while (value >= *left && std::distance(left, end) > 0) {
        if (std::next(left) == end)
            break;
        left = std::next(left);
    }

    // move right-to-left while value is less than elements
    while (value < *right) {
        right = std::prev(right);
    }

    // exchange so elements less than value are on the left
    // and elements greater than value are on the right
    std::swap(*left, *right);
}

// exchange pivot and final location
std::swap(*begin, *right);

// post-condition
assert(std::all_of(begin, right, [right](int n){ return n <= *right; }));
assert(std::all_of(right, end,   [right](int n){ return n >= *right; }));
```

```cpp
// pivot value is the first element
auto value = *begin;

auto left = begin;
auto right = std::prev(end);
while (std::distance(left, right) > 0) {

    // move left-to-right while value is greater than elements
    while (value >= *left && std::distance(left, end) > 0) {
        if (std::next(left) == end)
            break;
        left = std::next(left);
    }

    // move right-to-left while value is less than elements
    while (value < *right) {
        right = std::prev(right);
    }

    // exchange so elements less than value are on the left
    // and elements greater than value are on the right
    std::swap(*left, *right);
}

// exchange pivot and final location
std::swap(*begin, *right);

// post-condition
assert(std::all_of(begin, right, [right](int n){ return n <= *right; }));
assert(std::all_of(right, end,   [right](int n){ return n >= *right; }));
```

Summary

- Consistency, Consistency, Consistency
- Coding styles often reflect a compromise
- With existing code, adapt to the coding style of the existing code/project
- Be open to updating your coding style over time
- There are many more essential things in a project than minor coding style decisions, so consistency is the most important