This document focuses on the essential constraints of coding style in C programming language, part of which is summarized and modified from *Google C++ Style Guide*.

1 Comments

1.1 File

The file-level comment should broadly describe the contents of the file, and how the abstractions are related Do not duplicate comments in both the .h and the .c. Duplicated comments diverge.

Code Example:

```
// ------------------------------------------------------------------------

// str_cat.h
// ------------------------------------------------------------------------

//
// This header file contains functions for efficiently concatenating and appending
// strings: StrCat() and StrAppend(). Most of the work within these routines is
// actually handled through use of a special AlphaNum type, which was designed
// to be used as a parameter type that efficiently manages conversion to
// strings and avoids copies in the above operations.
```

1.2 Struct

. h: usage comment: how & when to use.

. c: implementation comment: how the methods are implemented

Code Example:

```
// Iterates over the contents of a GargantuanTable.
// Example(Usage of this struct:
):
//
struct GargantuanTableIterator {
```

```
  ...
};
```

1.3 Variables

All global variables should have a comment describing what they are, what they are used for, and (if unclear) why they need to be global.

No need to write in full sentences, but do capitalize the comment's first word (unless it's the name of a function, variable, or the like), and do leave one space between the // and your comment's first character, as in:

Code Example:

```
// Full Sentence Example
// The total number of test cases that we run through in this
    regression test.
const int kNumTestCases = 6;
// Not Full Sentence
// Convert Fahrenheit to Celsius
float c = 5.0 / 9.0 * (f - 32.0);
```

2 Naming

2.1 Indentation

Default indentation is 2 spaces.

Wrapped parameters have a 4 space indent.

Code Example:

```
ReturnType LongClassName::ReallyReallyReallyLongFunctionName(
    Type par_name1,  // 4 space indent
    Type par_name2,
    Type par_name3) {
  DoSomething();  // 2 space indent
  ...
}
```

2.2 Variables

lower_case with _ attributive_noun

Code Example:

```
var
my_var
temp_var
```

2.3 Pointers

No spaces around period or arrow. Pointer operators do not have trailing spaces.

Code Example:

```
x = *p;
p = &x;
x = r.y;
x = r->y;
```

2.4 Macro

Macro should be named with all capitals and underscores

Code Example:

```
#define PI_ROUNDED 3.0
```

2.5 Functions

Ordinarily, functions should start with a capital letter and have a capital letter for each new word.

Accessors and mutators (get and set functions) may be named like variables. These often correspond to actual member variables, but this is not required. For example, int count() and void set_count(int count). Verb+Noun

Code Example:

```
AddTableEntry()
DeleteUrl()
OpenFileOrDie()

void set_count(int count)
```

2.6 Structures

Name style as functions Attributive+Noun All data members should end with _

Code Example:

```
struct UrlTable{
  int num_total_entries_;…

}
```

3 Conditions

3.1 If

In an if statement, including its optional else if and else clauses, put one space between the if and the opening parenthesis, and between the closing parenthesis and the curly brace (if any), but no spaces between the parentheses and the condition or initializer. If the optional initializer is present, put a space or newline after the semicolon, but not before.

Code Example:

```
if (condition) {                    // no spaces inside parentheses,
    space before brace
  DoOneThing();                     // two space indent
  DoAnotherThing();
} else if (int a = f(); a != 3) {   // closing brace on new line, else
   on same line
  DoAThirdThing(a);
} else {
  DoNothing();
}
```

3.2 Switches

If not conditional on an enumerated value, switch statements should always have a default case.

If the default case should never execute, treat this as an error. For example:

Code Example:

```
switch (var) {
  case 0: {  // 2 space indent
    ...      // 4 space indent
    break;
```

```
  }
  case 1: {
    ...
    break;
  }
  default: {
    assert(false);
  }
}
```

4 Loops

4.1 For

No Description Available

Code Example:

```
for (int i = 0; i < kSomeNumber; ++i) {
  printf("I take it back\n");
}
```

4.2 While

Empty loop bodies should use either an empty pair of braces or continue with no braces, rather than a single semicolon.

Code Example:

```
while (condition) {
  // Repeat test until it returns false.
}
while (condition) continue;  // Good - continue indicates no logic.
```

5 Header

5.1 Include

In dir/foo.c or dir/foo_test.c, whose main purpose is to implement or test the stuff in dir2/foo2.h, order your includes as follows:

1. dir2/foo2.h.

2. A blank line
3. C system headers (more precisely: headers in angle brackets with the .h extension), e.g., <unistd.h>, <stdlib.h>.
4. A blank line
5. Other libraries' .h files.
6. A blank line
7. Your project's .h files.

Code Example:

```
#include "foo/server/fooserver.h"

#include <sys/types.h>
#include <unistd.h>

#include "base/basictypes.h"
#include "base/commandlineflags.h"
#include "foo/server/bar.h"
```