C++编码规范

命名规范

函数命名, 变量命名, 文件命名要有描述性

文件命名

- 文件名要全部小写, 可以包含下划线 (`_`) 或连字符 (`-`)。具体可依照项目的约定， 如果没有约定, 那么 “`_`” 更好。

```
my_useful_class.cc
my-useful-class.cc
myusefulclass.cc
```

类型命名

- 类型名称（类，结构体，枚举, `typedef`等）的每个单词首字母均大写, 不包含下划线。

```
// 类和结构体
class UrlTable { ...
class UrlTableTester { ...
struct UrlTableProperties { ...

// 类型定义
typedef hash_map<UrlTableProperties *, string> PropertiesMap;

// using 别名
using PropertiesMap = hash_map<UrlTableProperties *, string>;

// 枚举
enum UrlTableErrors { ...
```

变量命名

- 变量和数据成员名一律小写，单词之间用下划线连接。类的成员变量以下划线结尾。

```
string table_name;  //用下划线
string tablename;   //全小写
```

```
class TableInfo {
  ...
 private:
  string table_name_;
  string tablename_;
  static Pool<TableInfo>* pool_;
};
```

函数命名

- 常规函数使用大小写混合, 取值和设值函数则要求与变量名匹配。

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

命名空间命名

- 命名空间以小写字母命名，最高级命名空间的名字取决于项目名称。注意避免命名冲突。

常量命名

- 声明为 `constexpr` 或 `const` 的变量，或在程序运行期间其值始终保持不变的， 命名时以 “k” 开头, 大小写混合。

```
const int kDaysInAWeek = 7;
```

宏命名

- 全部大写，使用下划线。

```
#define ROUND(x) ...
#define PI_ROUNDED 3.0
```

枚举命名

- 枚举的命名应当和常量或宏一致。 `kEnumName` 或是 `ENUM_NAME`。

```
enum UrlTableErrors {
    kOK = 0,
    kErrorOutOfMemory,
    kErrorMalformedInput,
};
enum AlternateUrlTableErrors {
    OK = 0,
    OUT_OF_MEMORY = 1,
    MALFORMED_INPUT = 2,
};
```

格式规范

整个项目应统一编程风格。

- 每一行代码字符数不超过 80。
- 函数返回类型和函数名在同一行, 参数也尽量放在同一行, 如果放不下就对形参分行。

```
ReturnType ClassName::ReallyLongFunctionName(Type par_name1, Type par_name2,
                                             Type par_name3) {
  DoSomething();
  ...
}
```

- Lambda 表达式对形参和函数体的格式化和其他函数一致。若用引用捕获, 在变量名和 `&` 之间不留空格；短 lambda格式和内联函数一致。

```
int x = 0;
auto add_to_x = [&x](int n) { x += n; };
```

- 一般不在圆括号内使用空格，关键字 `if` 和 `else` 另起一行。单行语句不需要使用大括号， 复杂的条件或循环语句用大括号可读性会更好。

```
if (condition) {  // 圆括号里没有空格.
  ...  // 2 空格缩进.
} else if (...) {  // else 与 if 的右括号同一行.
  ...
} else {
  ...
}
```

- `switch` 语句可以使用大括号分段，如果有不满足 `case` 条件的枚举值, `switch` 应该总是包含一个 `default` 匹配。如果 `default` 应该永远执行不到, 简单的加条 `assert`

```
switch (var) {
  case 0: {  // 2 空格缩进
    ...      // 4 空格缩进
    break;
  }
  case 1: {
    ...
    break;
  }
  default: {
    assert(false);
  }
}
```

- 在单语句循环里， 括号可用可不用。

```
for (int i = 0; i < kSomeNumber; ++i)
  printf("I love you\n");

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

- 空循环体应使用 `{}` 或 `continue`。

```
while (condition) {
  // 反复循环直到条件失效.
}
for (int i = 0; i < kSomeNumber; ++i) {}  //空循环体.
while (condition) continue;  //contunue 表明没有逻辑.
```

- 访问成员时，句点或箭头前后不要有空格，指针/地址操作符 (`*, &`) 之后不能有空格。

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

- 类内访问控制块的声明依次序是 `public:`, `protected:`, `private:`, 每个都缩进 1 个空格。

```
class MyClass : public OtherClass {
 public:      // 注意有一个空格的缩进
  MyClass();  // 标准的两空格缩进
  explicit MyClass(int var);
  ~MyClass() {}

  void SomeFunction();
  void SomeFunctionThatDoesNothing() {
  }

  void set_some_var(int var) { some_var_ = var; }
  int some_var() const { return some_var_; }

 private:
  bool SomeInternalFunction();

  int some_var_;
  int some_other_var_;
};
```

函数&变量规范

- 所有头文件都应该使用 `#define` 来防止头文件被多重包含, 命名格式当是: `<PROJECT>_<PATH>_<FILE>_H_`

```
#ifndef FOO_BAR_BAZ_H_
#define FOO_BAR_BAZ_H_
...
#endif // FOO_BAR_BAZ_H_
```

- 尽量不要用裸的全局函数，将函数置于命名空间中，使用静态成员函数或命名空间内的非成员函数。
- 函数的参数顺序为：输入参数在先，后跟输出参数。输入参数通常是值参或 `const` 引用, 输出参数或输入/输出参数则一般为非 `const` 指针。
- 局部变量尽可能置于最小作用域内, 并在变量声明时进行初始化。
- 所有按引用传递的参数必须加上 `const`。在可能的情况下尽可能使用 `const`. 此外有时改用 C++11 推出的 `constexpr`更好。

```
void Foo(const string &in, string *out);
```

- 只允许在非虚函数中使用缺省参数, 且必须保证缺省参数的值始终一致。
- C++11 可以在函数名前使用 `auto` 关键字, 在参数列表之后后置返回类型

```
auto foo(int x) -> int;
```

注释规范

良好的注释对提高代码可读性至关重要，好的代码应当本身就是文档。

- 文件注释描述该文件的内容。
- 每个类的定义都要附带一份注释, 描述类的功能和用法。
- 函数声明处前都应当加上注释, 描述函数的功能和用途。
- `TODO` 注释标记一些未完成或已经够好但仍不完美的代码。

```
// TODO(kl@gmail.com): Use a "*" here for concatenation operator.
// TODO(Zeke) change this to use relations.
// TODO(bug 12345): remove the "Last visitors" feature
```

更多内容请参考Google C++风格指南