C++

Code Style Guide

Что такое Style Guide?

Это некий принятый свод правил о том:

- как оформлять код
- как именовать переменные, классы, файлы...
- как комментировать код
- и многое другое

Зачем он нужен?

- *Для* ускорения разработки
- *Для* увеличения качества программного продукта
- *Для* более приятной работы с кодом

Обо всем по порядку

“Любой дурак может написать программу, которую поймет компилятор.

Хорошие программисты пишут программы, которые смогут понять другие программисты.”

M. Fowler

Обо всем по порядку

Большинство компаний и команд разработчиков не располагает возможностью писать документацию для своего продукта

Обо всем по порядку

По ряду причин:

- очень мало хороших технических писателей
- нет времени или желания писать документацию
- документация не успевает за проектом

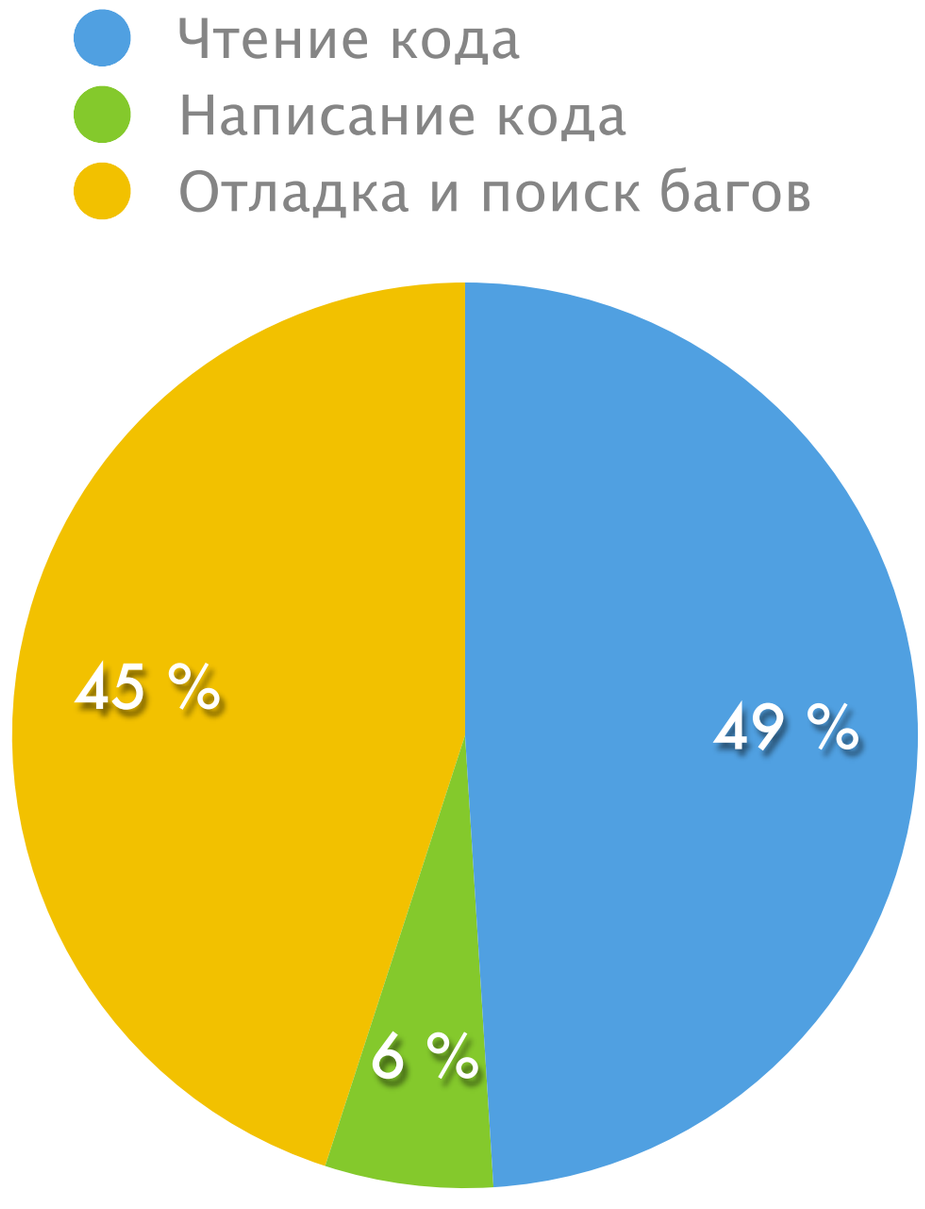
Обо всем по порядку

Очень быстро программисты пришли к выводу, что сам код и есть документация

Причем всегда полная и актуальная

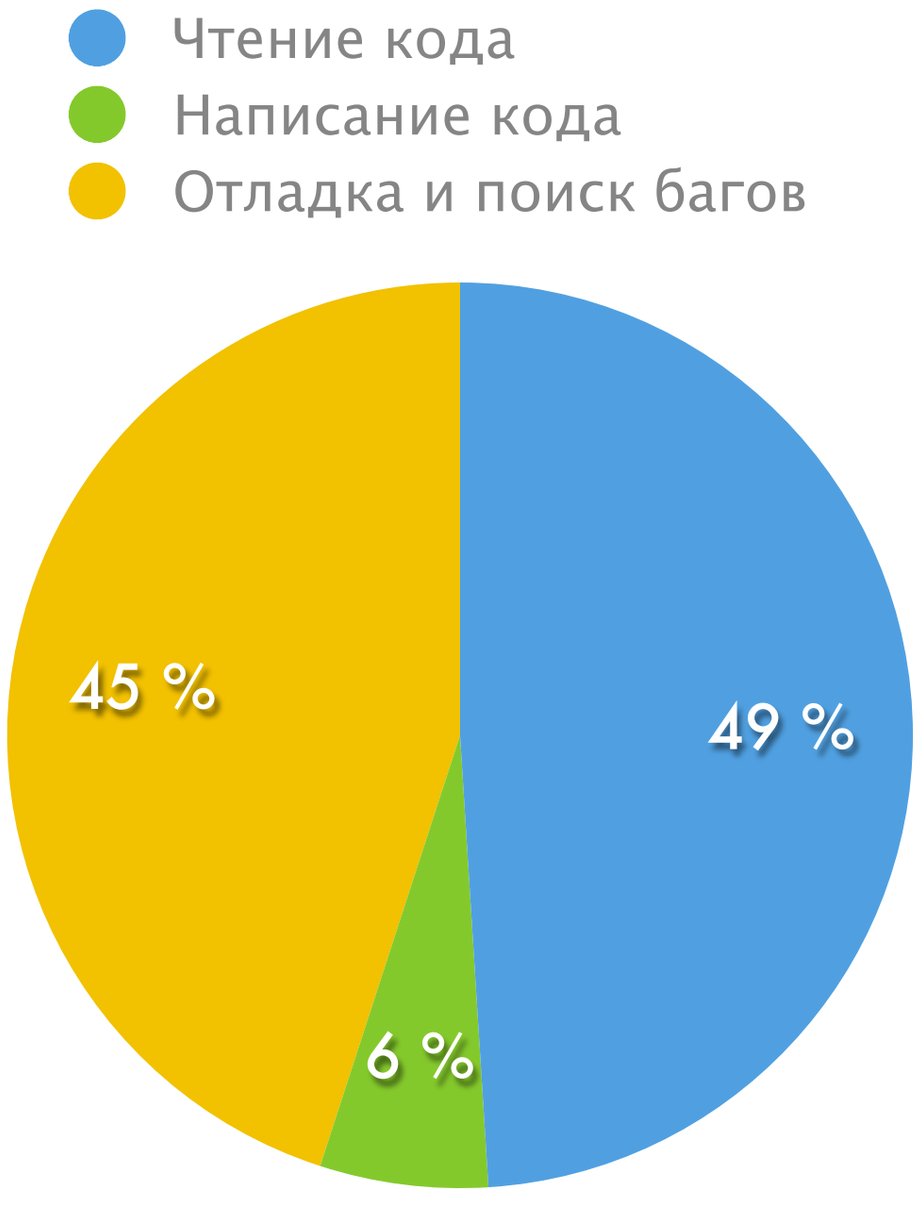
На что тратится время?

Большая часть времени программиста тратится не на написание кода



На что тратится время?

Гораздо выгоднее нанять программиста, который печатает одним пальцем хороший код, чем программиста, который печатает плохой код всеми десятью



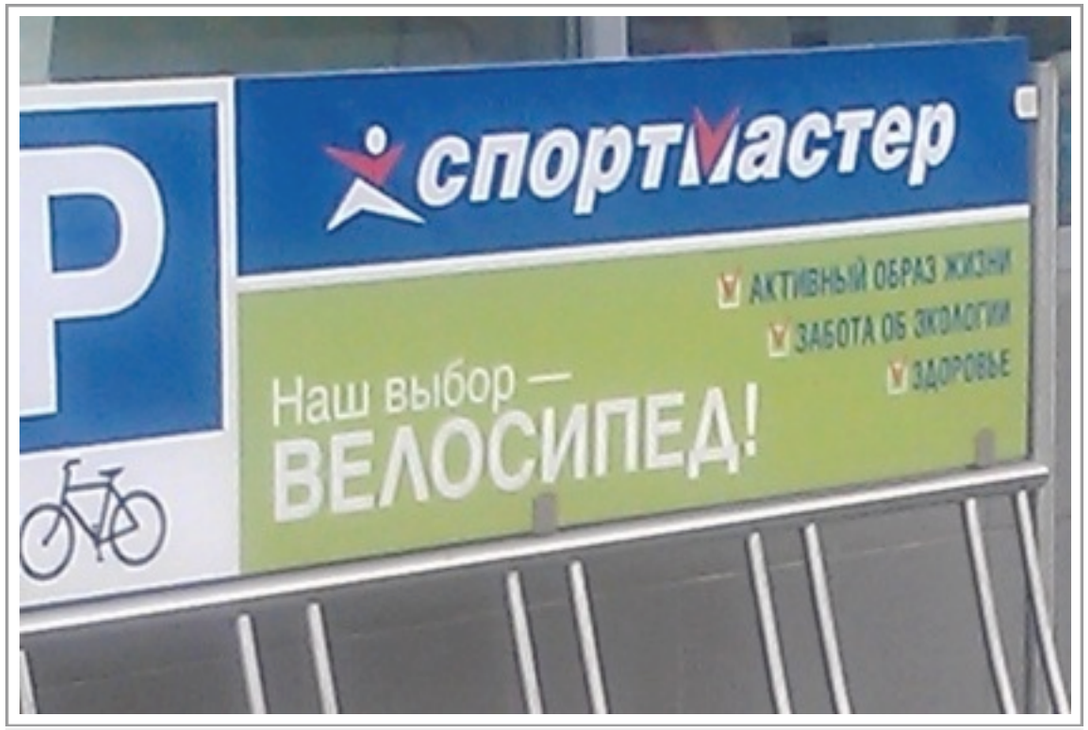
К чему это все?

Понимание того, как работает программа или ее часть, должно приходить как можно быстрее

Сэкономив на красоте кода вы потеряете теряете куда больше времени отлаживая его

О стилях кодирования в языке C++

О стилях кодирования в C++



В сообществе “плюсеров” нет общего свода стилевых правил

О стилях кодирования в C++

- Google C++ Style Guide
- GNU Coding Standards
- Linux Kernel Coding Style
- Mozilla Coding Style Guide
- Road Intranet's C++ Guidelines
- Qt Style Guide

О стилях кодирования в C++

И это только самые популярные

Какой же выбрать?

- Серебряной пули здесь нет
- Все стандарты по-своему хороши и по-своему плохи
- В наших проектах принято использовать стиль Google с некоторыми оговорками

Google C++ Style Guide

И сразу оговорка

В стандарте от Google в качестве отступа берется 2 пробела

В наших проектах в качестве отступов используются **4 пробела**

Но как и в стандарте от Google мы не используем в коде символы табуляции

Длина строки

Длина строки не должна превышать **80 символов**

Как укладываться в 80 символов будет показано далее

Далее внимательно следите за тем, где стоят пробелы, а где не стоят

Это важно!

The #define guards

```
#ifndef NAME_OF_HEADER_FILE_H_
#define NAME_OF_HEADER_FILE_H_

/* code of header file */

#endif // NAME_OF_HEADER_FILE_H_
```

Все заголовочные файлы должны быть обернуты в эту конструкцию, чтобы избежать множественного подключения

Порядок подключения заголовков

```
#include <sys/types.h> // C libs
#include <unistd.h>

#include <hash_map> // C++ libs
#include <vector>

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

Именование файлов

Комплект файлов класса UrlTable:

url_table.h

url_table.cc

Иногда требуется файл **url_table-inl.h**, который содержит объемные inline-функции (от 1КБ)

Структуры и классы

Структуры используются только для реализации объектов, хранящих **только** данные

Все остальные объекты реализуются классами

Порядок определений в *.h файлах

- Typedef'ы и Enum'ы
- Константы
- Конструкторы
- Деструктор
- Методы и статические методы
- Переменные и статические переменные

Именование переменных

Имена переменных должны быть осмысленными

```
int num_errors;                    // Хорошо
int num_completed_connections; // Хорошо

int n;                    // Плохо
int nerr;                 // Плохо
int n_comp_conns; // Плохо
```

Именование переменных

Имена переменных должны быть понятны сторонним разработчикам

```
// Хорошо
int num_dns_connections; // Большинство программистов знают, что такое DNS

int price_count; // Нормально, из контекста скорее всего будет ясен смысл
// Но как раз тот случай, когда не лишним будет комментарий

// Плохо!
// Аббревиатуры могут вводить в заблуждение

int wgc_connections; // Возможно только вы знаете расшифровку WGC

int pc_reader; // Эта переменная может означать практически что угодно
```

Именование переменных

Не сокращайте имена переменных,
выбрасывая из них буквы

```
int error_count;  // Хорошо

int error_cnt;    // Плохо
```

Именование переменных

```
string table_name; // Хорошо
string tablename;  // Нормально
string tableName;  // Плохо, смешанный стиль


// Глобальная переменная ( префикс g_ )
int g_max_integer;

// Константа (префикс k и смешанный стиль)
const int kDaysInAWeek = 7;
```

Локальные переменные

```
int i;   // Неправильно, нельзя разделять объявление и
i = f(); // первое присваивание
int j = g(); // Хорошо

// Нормально
while (const char* p = strchr(str, '/')) str = p + 1;

for (int i = 0; i < 1000000; ++i) {
   Foo f; // Плохо: объявление переменной внутри цикла
   f.DoSomething(i);
}

Foo f;
for (int i = 0; i < 1000000; ++i) {
   f.DoSomething(i);
}
```

Именование типов и классов

```
// classes and structs
class UrlTable { ...
class UrlTableTester { ...
struct UrlTableProperties { ...

// typedefs
typedef hash_map<UrlTable *, string> GlobalMap;

// enums
enum UrlTableErrors { ...
```

Именование функций

```
AddTableEntry() // Глобальные функции
DeleteUrl()

class MyClass {

public:
    // методы
    int num_entries() const { return num_entries_; }
    void set_num_entries(int num_entries) {
        num_entries_ = num_entries;
    }

private:
    int num_entries_; // приватная переменная
};
```

Именование enum'ов

```
enum UrlTableErrors {
    kOK = 0,
    kErrorOutOfMemory,
    kErrorMalformedInput,
};

enum AlternateUrlTableErrors {
    OK = 0,
    OUT_OF_MEMORY = 1,
    MALFORMED_INPUT = 2,
};
```

Вызовы функций

Длина строки не должна превышать 80 символов

```
bool retval = DoSomething(arg1, arg2, arg3);
bool retval = DoSomething(averyveryveryverylongargument1,
                          argument2, argument3);
bool retval = DoSomething(argument1,
                          argument2,
                          argument3,
                          argument4);
```

Условный оператор

```
// Правильно
if (condition) {
    // ...
} else {
    // ...
}

// Неправильно расставлены пробелы
if(condition)
if (condition){
if(condition){

// Не хватает фигурных скобок (опасная конструкция)
if (x) DoThis();
else DoThat();
```

Условный оператор

```
// Следите за правильностью фигурных скобок
if (condition) {
    foo();
} else
    bar();

if (condition)
    foo();
else {
    bar();
}
```

Hint! Тернарный оператор

Если у нам нужно посчитать некоторое значение по сложному дереву условий, то можно сделать так:

```
a = (x > 0)
    ? (y > 0)
        ? 1
        : 2
    : (y > 0)
        ? 3
        : 4;
```

Hint! Тернарный оператор

Предыдущий пример выглядит гораздо понятнее, чем тот же код без форматирования:

```
a = (x > 0) ? (y > 0) ? 1 : 2 : (y > 0) ? 3 : 4;
```

Циклы

```
while (condition) {
    // ...
}

// Цикл с пустым телом
for (int i = 0; i < kSomeNumber; ++i) {}

while (condition) continue; // Хорошо

while (condition); // Плохо, в коде будет смотреться как
окончание конструкции do { } while ();
```

Конструкция Switch

```
switch (var) {
    case 0: {  // отступ 4 пробела
        // ... // отступ 8 пробелов
        break;
    }
    case 1: {
        // ...
        break;
    }
    default: {
        assert(false);
    }
}
```

Ссылки

Google C++ Code Style Guide

http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

Вопросы ?