Правила оформления кода:

**1**. Функциям (методам, возвращающим какие-либо значения) следует давать имена в зависимости от того, что они возвращают, а процедурам — в зависимости от того, что они выполняют (методы void).

**2**. Ширина отступа для вложенных блоков - 4 пробела

Определение методов и функций в форме   
TYPE func\_first(){

...

cicle(){

TYPE var = act();

}

...

}

**3**. Имена, представляющие типы, должны быть обязательно написаны в смешанном регистре, начиная с верхнего.

Line, SavingsAccount

**4**. Имена переменных должны быть записаны в нижнем регистре, начиная с нижнего, с нижним подчёркиванием в качестве разделителя.

line, savings\_account

Общая практика в сообществе разработчиков C++. Позволяет легко отличать переменные от типов, предотвращает потенциальные коллизии имён, например: Line line;

**5**. Именованные константы (включая значения перечислений) должны быть записаны в верхнем регистре с нижним подчёркиванием в качестве разделителя.

MAX\_ITERATIONS, COLOR\_RED, PI

**6**. Названия методов и функций должны быть глаголами, быть записаны в нижнем регистре, начиная с нижнего, с нижним подчёркиванием в качестве разделителя.

get\_name(), compute\_total\_width()

Совпадает с правилом для переменных, но отличие между ними состоит в их специфических формах.

**7**. Названия пространств имён следует записывать в нижнем регистре.

model::analyzer, io::iomanager, common::math::geometry

Общая практика в сообществе разработчиков C++.

**8**. Следует называть имена типов в шаблонах одной заглавной буквой.

template<class T> ...

template<class C, class D> ...

Общая практика в сообществе разработчиков C++. Позволяет выделить имена шаблонов среди других используемых имён.

**9**. Аббревиатуры и сокращения в именах должны записываться в нижнем регистре.

export\_html\_source();

open\_dvd\_player();

**10**. Глобальные переменные всегда следует использовать с оператором разрешения области видимости (::).

::main\_window.open(), ::application\_context.get\_name()

Следует избегать использования глобальных переменных. Предпочтительнее использование синглтонов.

**11**. Членам класса с модификатором private следует присваивать суффикс-подчёркивание.

class SomeClass {

private:

int length\_;

}

Не считая имени и типа, область видимости — наиболее важное свойство переменной. Явное указание модификатора доступа в виде подчёркивания избавляет от путаницы между членами класса и локальными переменными. Это важно, поскольку переменные класса имеют большее значение, нежели переменные в методах, и к ним следует относиться более осторожно.

Дополнительным эффектом от суффикса-подчёркивания является разрешение проблемы именования в методах, устанавливающих значения, а также в конструкторах:

void set\_depth (int depth)

{

depth\_ = depth;

}

**12**. Настраиваемым переменным следует давать то же имя, что и у их типа.

void set\_topic(Topic\* topic);

void connect(Database\* database);

Сокращайте сложность путём уменьшения числа используемых терминов и имён. Также упрощает распознавание типа просто по имени переменной.

Если по какой-то причине эта рекомендация кажется неподходщей, это означает, что имя типа выбрано неверно.

Не являющиеся настраиваемыми переменные могут быть названы по их назначению и типу:

Point starting\_point, center\_point;

Name login\_name;

**13**. Все имена следует записывать по-английски.

file\_name; // НЕ РЕКОМЕНДУЕТСЯ: imyaFayla

Английский наиболее предпочитетелен для интернациональной разработки.

**14**. Переменные, имеющие большую область видимости, следует называть длинными именами, имеющие небольшую область видимости — короткими.

Имена временных переменных, использующихся для хранения временных значений или индексов, лучше всего делать короткими. Программист, читающий такие переменные, должен иметь возможность предположить, что их значения не используются за пределами нескольких строк кода. Обычно это переменные i, j, k, l, m, n (для целых), а также c и d (для символов).

**15**. Имена объектов не указываются явно, следует избегать указания названий объектов в именах методов.

line.get\_length(); // НЕ РЕКОМЕНДУЕТСЯ: line.getLineLength();

Второй вариант смотрится вполне естественно в объявлении класса, но совершенно избыточен при использовании, как это и показано в примере.

***Особые правила именования***

**17**. Слова get/set должны быть использованы везде, где осуществляется прямой доступ к атрибуту.

employee.get\_name();

employee.set\_name(name);

matrix.get\_element(2, 4);

matrix.set\_element(2, 4, value);

Общая практика в сообществе разработчиков C++.

**18**. Слово compute может быть использовано в методах, вычисляющих что-либо.

value\_set->compute\_average();

matrix->compute\_inverse()

**19**. Слово find может быть использовано в методах, осуществляющих какой-либо поиск.

vertex.find\_nearest\_vertex();

matrix.find\_min\_element();

**20**. Слово init может быть использовано там, где объект или сущность инициализируется.

printer.init\_font\_set();

**22**. Множественное число следует использовать для представления наборов (коллекций) объектов.

vector<Point> points;

int values[];

**23**. Префикс n следует использовать для представления числа объектов.

n\_points, n\_ines

**25**. Переменным-итераторам следует давать имена i, j, k и т. д.

for (int i = 0; i < n\_tables); i++) {

:

}

**26**. Префикс is следует использовать только для булевых (логических) переменных и методов.

is\_set, is\_visible

Общая практика в сообществе разработчиков C++, иногда используемая и в Java.

**27**. Симметричные имена должны использоваться для соответствующих операций.

get/set, add/remove, create/destroy, start/stop, insert/delete,

increment/decrement, old/new, begin/end, first/last, up/down, min/max,

next/previous, old/new, open/close, show/hide, suspend/resume, и т. д.

**28**. Следует сокращать слишком длинные слова в именах(

в пределах читаемости)

comp\_average();

**29**. Следует дополнительно именовать указатели, когда это необходимо.

Line\* line\_ptr;

30. Нельзя давать булевым (логическим) переменным имена, содержащие отрицание.

bool is\_error; // НЕЛЬЗЯ: is\_no \_error

**31**. Константы в перечислениях могут иметь префикс — общее имя типа.

enum Color {

COLOR\_RED,

COLOR\_GREEN,

COLOR\_BLUE

};

Это даёт дополнительную информацию о том, где находится объявление, какие константы описаны в одном перечислении, а также какую концепцию являют собой константы.

Обратите внимание, что имя перечисления обычно записано в единственном числе, например: enum Color {...}. Имя во множественном числе хорошо выглядит при объявлении, но не очень хорошо подходит для практического использования.

**32**. Классам исключений следует присваивать суффикс Exception.

class AccessException

{

:

}