Правила кодирования (Coding Conventions) C++

1. Имя сущности должно отражать её назначение, исключая двусмысленность.
2. Все имена следует записывать по-английски

```
fileName;   // НЕ РЕКОМЕНДУЕТСЯ: imyaFayla
```

3. Следует избегать "магических" чисел в коде.

   Числа, отличные от **0** или **1**, следует объявлять как именованные константы или создавать метод для доступа к константе.

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

```
MAX_ITERATIONS, COLOR_RED, PI
```

5. Названия пространств имён следует записывать в нижнем регистре

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

6. Имена типов должны быть записаны в смешанном регистре, начиная с верхнего *(UpperCamelCase)*

```
Line, SavingsAccount
```

7. Имена типов в шаблонах следует называть одной заглавной буквой

```
template<class T>
//...
template<class C, class D>
//...
```

8. Имена переменных должны быть записаны в смешанном регистре, начиная с нижнего *(lowerCamelCase)*

```
line, savingsAccount
```

   Аббревиатуры и сокращения в именах должны записываться в нижнем регистре

```
exportHtmlSource(); // НЕЛЬЗЯ: exportHTMLSource();
openDvdPlayer();    // НЕЛЬЗЯ: openDVDPlayer();
```

   Переменным пользовательских типов в общем случае следует давать то же имя, что и у их типа

```
void setTopic(Topic* topic)      // НЕ РЕКОМЕНДУЕТСЯ: void setTopic(Topic* value)
                                 // НЕ РЕКОМЕНДУЕТСЯ: void setTopic(Topic* aTopic)
                                 // НЕ РЕКОМЕНДУЕТСЯ: void setTopic(Topic* t)

void connect(Database* database) // НЕ РЕКОМЕНДУЕТСЯ: void connect(Database* db)
                                 // НЕ РЕКОМЕНДУЕТСЯ: void connect (Database* oracleDB)
```

   Если название переменной не подходит по смыслу, то, скорее всего, и имя типа выбрано неверно.

Переменные пользовательских типов могут быть названы по назначению и типу, если нужно подчеркнуть их предназначение

```
Point startingPoint, centerPoint;
Name loginName;
```

Для представления наборов (коллекций) объектов следует использовать множественное число

```
vector<Point> points;
int values[];
```

Переменные, имеющие большую область видимости, следует называть длинными именами, а небольшую – короткими.

Имена переменных для хранения временных значений или индексов в пределах нескольких строк кода следует делать короткими. Обычно это переменные **i, j, k, l, m, n** для целых, а также **c** и **d** для символов.

9. Названия функций и методов должны быть записаны в смешанном регистре и начинаться с нижнего *(lowerCamelCase)*

```
getName(), computeTotalWidth()
```

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

```
employee.getName();
employee.setName(name);

matrix.getElement(2, 4);
matrix.setElement(2, 4, value);
```

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

```
isSet, isVisible, isFinished, isFound, isOpen
```

В некоторых ситуациях префикс **is** лучше заменить на другой: **has**, **can** или **should**

```
bool hasLicense();
bool canEvaluate();
bool shouldSort();
```

Нельзя давать булевым (логическим) переменным имена, содержащие отрицание

```
bool isError; // НЕЛЬЗЯ: isNoError или noError
bool isFound; // НЕЛЬЗЯ: isNotFound или notFound
```

Комментарии к функциям и методам следует давать при объявлении в соответствии с соглашениями **JavaDoc**.

10. Использования глобальных переменных следует избегать.

Глобальные переменные и функции всегда следует использовать с оператором разрешения области видимости **::**

```
::mainWindow.open(), ::applicationContext.getName()
::getMaxWindowWidth()
```

11. Инициализировать переменные лучше при объявлении.

    Если инициализация при объявлении невозможна или не имеет смысла, то лучше оставить переменные неинициализированными, чем присваивать им какие-либо значения

```
int x, y, z;
getCenter(&x, &y, &z);
```

12. Приведение типов должно быть явным.

    Приведения типов в стиле языка *C* следует избегать

```
floatValue = static_cast<float>(intValue);  // НЕ РЕКОМЕНДУЕТСЯ: floatValue = intValue;
                                            // НЕЛЬЗЯ:    floatValue = (float)intValue;
```

13. Пробелы и отступы следует использовать для улучшения читабельности

    - Операторы следует отбивать пробелами.
    - После зарезервированных ключевых слов языка *C++* следует ставить пробел.
    - После запятых следует ставить пробелы.
    - В списках инициализации конструкторов двоеточие следует отбивать пробелами.
    - После точек с запятой в цикле **for** следует ставить пробелы.

    Основной отступ – табуляция (четыре пробела).

```
a = (b + c) * d;             // НЕ РЕКОМЕНДУЕТСЯ: a=(b+c)*d;
while (true) {               // НЕ РЕКОМЕНДУЕТСЯ: while(true){
doSomething(a, b, c, d);     // НЕ РЕКОМЕНДУЕТСЯ: doSomething(a,b,c,d);
SomeClass::SomeClass() :     // НЕ РЕКОМЕНДУЕТСЯ: SomeClass::SomeClass():
for (i = 0; i < 10; i++) {   // НЕ РЕКОМЕНДУЕТСЯ: for(i=0;i<10;i++){
```

    Логически связанные блоки в коде следует отделять пустой строкой и/или строкой с поясняющими комментариями

```
Matrix4x4 matrix = new Matrix4x4();

// Пояснения...
double cosAngle = Math.cos(angle);
double sinAngle = Math.sin(angle);

// Пояснения...
matrix.setElement(1, 1, cosAngle);
matrix.setElement(1, 2, sinAngle);
matrix.setElement(2, 1, -sinAngle);
matrix.setElement(2, 2, cosAngle);

// Пояснения...
multiply(matrix);
```

14. Циклы следует оформлять следующим образом

```
for (initialization; condition; update) {
    //...
}
```

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

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

Циклов с пост-условием **do-while** следует избегать.

Для бесконечных циклов следует использовать форму **while (true)**.

Выражения, не относящиеся к управлению циклом, в конструкцию **for** включать нельзя

```
sum = 0;                        // НЕЛЬЗЯ:
for (i = 0; i < n; i++) {       // for (i = 0, sum = 0; i < n; i++) {
    sum += value[i];            //     sum += value[i];
}                               // }
```

Переменные, относящиеся к циклу, следует инициализировать непосредственно перед ним

```
isDone = false;     // НЕ РЕКОМЕНДУЕТСЯ:
while (!isDone) {   // bool isDone = false;
    //...           // ...
}                   // while (!isDone) {
                    //     ...
                    // }
```

15. Конструкцию **if-else** следует оформлять следующим образом

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

Условие следует размещать в отдельной строке

```
if (isDone) {      // НЕ РЕКОМЕНДУЕТСЯ: if (isDone) doCleanup();
    doCleanup();
}
```

Сложных условных выражений следует избегать. Лучше вместо этого использовать булевы переменные

```
bool isFinished = (elementNo < 0) || (elementNo > maxElement);
bool isRepeatedEntry = elementNo == lastElement;
if (isFinished || isRepeatedEntry) {
    //...
}

// НЕ РЕКОМЕНДУЕТСЯ:
// if ((elementNo < 0) || (elementNo > maxElement)||
//         elementNo == lastElement) {
//     ...
// }
```

Исполняемых выражений в условиях следует избегать

```
File* fileHandle = open(fileName, "w");
if (!fileHandle) {
    //...
}

// НЕ РЕКОМЕНДУЕТСЯ:
// if (!(fileHandle = open(fileName, "w"))) {
//     ...
// }
```

16. Конструкцию **switch** следует оформлять следующим образом

```
switch (condition) {
case 1:
    //...
    // Отсутствует "break"
case 2:
    //...
    break;
case 3:
    //...
    break;
default:
    //...
    break;
}
```

Если где-то отсутствует ключевое слово **break**, то следует предупреждать об этом в комментарии.

17. Конструкцию **try-catch** следует оформлять следующим образом

```
try {
    //...
} catch (Exception &exception) {
    //...
}
```

18. Объявления классов следует оформлять следующим образом

```
class SomeClass: public BaseClass {
public:
    SomeClass();
    virtual ~SomeClass();
    //...

protected:
    //...

private:
    //...
};
```

Вначале объявляются друзья (вне какого-либо раздела).

Разделы класса **public**, **protected** и **private** должны быть явно указаны, сгруппированы и следовать в данном порядке.

Статические элементы должны объявляться вначале своего раздела.

Не следует объявлять переменные класса (поля, атрибуты) как **public**. Если это необходимо – используйте структуры.

Использование структур следует избегать – ухудшает читаемость кода.

Комментарии к классам, полям и методам следует делать при объявлении в соответствии с соглашениями **JavaDoc**.

19. Определение (реализацию) методов следует оформлять следующим образом

```
SomeClass::SomeClass() :
        someField(0) {
    //...
}
```

```
void SomeClass::someMethod() {
    //...
}
```

20. Заголовочные (.h) файлы объявляют интерфейс, файлы исходного кода (.cpp) его реализовывают.

    Заголовочным (включаемым) файлам следует давать расширение **h** (рекомендуется) либо **hpp**.

    Файлы исходных кодов (компилируемые) могут иметь расширения **cpp** (рекомендуется), **C**, **cc** либо **c++**.

    Класс следует объявлять в заголовочном (.h) файле и определять (реализовывать) в компилируемом (.cpp) файле.

    Имя файла должно совпадать с именем класса.

```
MyClass.h, MyClass.cpp
```

    Исключение – шаблонные (параметризованные) классы, которые должны быть объявлены и определены во включаемых файлах.

    Заголовочные файлы должны содержать защиту от повторного включения

```
#ifndef FILENAME_H
#define FILENAME_H
    //...
#endif
```

    Заголовочные файлы не должны содержать директиву **using namespace**.

    Включать заголовочные файлы нужно по мере необходимости в следующем порядке:

    - собственный заголовочный файл;
    - дополнительные заголовочные файлы;
    - системные заголовочные файлы.
