**Гайд по оформлению кода на С++ от Стэнфордского университета**

Стэнфордский университет представил [гайд](http://stanford.edu/class/archive/cs/cs106b/cs106b.1158/styleguide.shtml) по основным стандартам оформления кода на С++. Умение корректно оформить ваш код является ценным навыком, так как это в разы облегчает работу других. Также у нас есть [подобная статья](https://tproger.ru/articles/15-tips-selfdoc-js/), посвящённая написанию самодокументируемого кода.

**Пробелы и отступы**

Отделяйте пробелами фигурные скобки:

// Плохая практика

int x = 3, y = 7; double z = 4.25; x++;

if (a == b) { foo(); }

// Хорошая практика

int x = 3;

int y = 7;

double z = 4.25;

x++;

if (a == b) {

foo();

}

Ставьте пробелы между операторами и операндами:

int x = (a + b) \* c / d + foo();

Когда строка становится длиннее 100 символов, разделите её на две, сделав перевод на новую строку после оператора, и продолжайте писать:

int result = reallyLongFunctionOne() + reallyLongFunctionTwo() +

reallyLongFunctionThree() + reallyLongFunctionFour();

int result2 = reallyLongFunction(parameterOne, parameterTwo, parameterThree,

parameterFour, parameterFive, parameterSix);

Оставляйте пустые линии между функциями и между группами выражений:

void foo() {

...

}

// пустая линия

void bar() {

...

}

**Названия и переменные**

Давайте переменным описательные имена, такие как firstName или homeworkScore. Избегайте однобуквенных названий вроде x или c, за исключением итераторов вроде i.

Называйте переменные и функции, используя верблюжийРегистр. Называйте классы ПаскальнымРегистром, а константы — в ВЕРХНЕМ\_РЕГИСТРЕ. Узнать подробнее про верблюжий регистр вы можете в [этой статье](https://tproger.ru/translations/camelcase-vs-underscores-scientific-showdown/).

Если переменная используется лишь внутри определенного if, то делайте её локальной, объявляя в том же блоке кода, а не глобальной.

Выбирайте подходящий тип данных для ваших переменных. Если переменная содержит лишь целые числа, то определяйте её как int, а не double.

Используйте текстовую строку, стандартную для C++, а не С. С++ путает тем, что имеет два вида текстовых строк: класс string из С++ и старый char\* (массив символов) из С:

// Плохая практика: текстовая строка в стиле Cи

char\* str = "Hello there";

// Хорошая практика: текстовая строка в стиле C++

string str = "Hello there";

Если определенная константа часто используется в вашем коде, то обозначьте её как const и всегда ссылайтесь на данную константу, а не на её значение:

const int VOTING\_AGE = 18;

Никогда не объявляйте изменяемую глобальную переменную. Глобальными переменными должны быть только константы. Вместо того, чтобы делать значение глобальным, сделайте его параметром и возвращайте значение, когда необходимо:

// Плохая практика

int count; // Глобальная переменная

void func1() {

count = 42;

}

void func2() {

count++;

}

int main() {

func1();

func2();

}

// Хорошая практика!

int func1() {

return 42;

}

void func2(int& count) {

count++;

}

int main() {

int count = func1();

func2(count);

}

**Базовые выражения С++**

С++ основан на С, поэтому всегда есть вариант решить задачу «путём С++» и «путём С». Например, когда вы желаете вывести что-либо на системную консоль, вы можете сделать это «путём С++», использовав оператор вывода cout, в то время как «путём С» вы бы использовали глобальную функцию вроде printf:

// Плохая практика

printf("Hello, world!\n");

// Хорошая практика

cout << "Hello, world!" << endl;

Частенько затрудняетесь с выбором между for и while? Используйте цикл for, когда вы знаете количество повторений, а цикл while, когда количество повторений неизвестно:

// Повторяет 'size' раз

for (int i = 0; i < size; i++) {

...

}

// Повторяет, пока больше не будет строк

string str;

while (input >> str) {

...

}

Когда используете операторы управления вроде if / else, for, while, всегда используйте {} и соответствующие отступы, даже если тело всего оператора управления состоит лишь из одной строки:

// Плохая практика

if (size == 0) return;

else

for (int i = 0; i < 10; i++) cout << "ok" << endl;

// Хорошая практика

if (size == 0) {

return;

} else {

for (int i = 0; i < 10; i++) {

cout << "ok" << endl;

}

}

Старайтесь избегать использования выражений break или continue. Используйте их только в том случае, если это абсолютно необходимо.

В C++ есть функция exit, которая немедленно завершает программу. Настоятельно не рекомендуется использовать данную функцию. Программа всегда должна заканчиваться естественно, достигая оператора return функции main.

Используя выражения if / else, подобающе выбирайте между разнообразными if и else шаблонами в зависимости от условий, относящихся к друг другу. Избегайте излишних тестов if:

// Плохая практика

if (grade >= 90) {

cout << "You got an A!";

}

if (grade >= 80 && grade < 90) {

cout << "You got a B!";

}

if (grade >= 70 && grade < 80) {

cout << "You got a C!";

}

// Хорошая практика

if (grade >= 90) {

cout << "You got an A!";

} else if (grade >= 80) {

cout << "You got a B!";

} else if (grade >= 70) {

cout << "You got a C!";

}

...

Если у вас есть выражение if / else, которое возвращает логическое значение, возвращайте результаты теста напрямую:

// Плохая практика

if (score1 == score2) {

return true;

} else {

return false;

}

// Хорошая практика

return score1 == score2;

Никогда не проверяйте значения логического типа, используя == или != с true или false:

// Плохая практика

if (x == true) {

...

} else if (x != true) {

...

}

// Хорошая практика

if (x) {

...

} else {

...

}

**Чрезмерность**

Если вы используете один и тот же код дважды или более, то найдите способ удалить излишний код, чтобы он не повторялся. К примеру, его можно поместить во вспомогательную функцию. Если повторяемый код похож, но не совсем, то постарайтесь сделать вспомогательную функцию, которая принимает параметры и представляет разнящуюся часть:

// Плохая практика

foo();

x = 10;

y++;

...

foo();

x = 15;

y++;

// Хорошая практика

helper(10);

helper(15);

...

void helper(int newX) {

foo();

x = newX;

y++;

}

Переместите общий код из выражения if / else, чтобы он не повторялся:

// Плохая практика

if (x < y) {

foo();

x++;

cout << "Привет!";

} else {

foo();

y++;

cout << "Привет!";

}

// Хорошая практика

foo();

if (x < y) {

x++;

} else {

y++;

}

cout << "Привет!";

**Комментарии**

**Заглавный комментарий**. Размещайте заглавный комментарий, который описывает назначение файла, вверху каждого файла. Предположите, что читатель вашего комментария является продвинутым программистом, но не кем-то, кто уже видел ваш код ранее.

**Заголовок функции / конструктора**. Разместите заголовочный комментарий на каждом конструкторе и функции вашего файла. Заголовок должен описывать поведение и / или цель функции.

**Параметры / возврат**. Если ваша функцию принимает параметры, то кратко опишите их цель и смысл. Если ваша функция возвращает значение — кратко опишите, что она возвращает.

**Исключения**. Если ваша функция намеренно выдаёт какие-то исключения для определенных ошибочных случаев, то это требует упоминания.

**Комментарии на одной строке**. Если внутри функции имеется секция кода, которая длинна, сложна или непонятна, то кратко опишите её назначение.

**TODO**. Следует удалить все // TODO комментарии перед тем, как заканчивать и сдавать программу.

**Эффективность**

Вызывая большую функцию и используя результат несколько раз, сохраните результат в переменной вместо того, чтобы постоянно вызывать данную функцию:

// Плохая практика

if (reallySlowSearchForIndex("abc") >= 0) {

remove(reallySlowSearchForIndex("abc"));

}

// Хорошая практика

int index = reallySlowSearchForIndex("abc");

if (index >= 0) {

remove(index);

}

**Функции и процедурное проектирование**

Хорошо спроектированная функция имеет следующие характеристики:

* Полностью выполняет чётко поставленную задачу;
* Не берет на себя слишком много работы;
* Не связана с другими функциями бесцельно;
* Хранит данные максимально сжато;
* Помогает распознать и разделить структуру программы;
* Помогает избавиться от излишков, которые иначе присутствовали бы в программе.

Используйте параметры, чтобы отправлять информацию из функции или когда функции нужно возвратить несколько значений. Не используйте параметры без необходимости. Заметьте, что a,b, и с не являются параметрами в нижеприведённой функции, так как это не нужно:

/\*

\* Решает квадратное уравнение ax^2 + bx + c = 0,

\* внося результаты в root1 и root2.

\* Предполагается, что данные уравнения имеют два корня.

\*/

void quadratic(double a, double b, double c,

double& root1, double& root2) {

double d = sqrt(b \* b - 4 \* a \* c);

root1 = (-b + d) / (2 \* a);

root2 = (-b - d) / (2 \* a);

}

Когда требуется вернуть значение из функции, используйте значение return:

// Плохая практика

void max(int a, int b, int& result) {

if (a > b) {

result = a;

} else {

result = b;

}

}

// Хорошая практика

int max(int a, int b) {

if (a > b) {

return a;

} else {

return b;

}

}

Отправляя объект в функцию как параметр, вы должны передавать его по ссылке, так как если он будет передан как значение, то будет скопирован весь объект. Копирование объектов требует больших затрат памяти.

Используйте ссылочные переменные, а не указатели. Одна из причин — это то, что ссылочные переменные, в отличие от указателей, не могут принимать значение NULL:

// Плохая практика

// Принимает указатель

void process(BankAccount\* account) {

...

}

// Хорошая практика

// Принимает адресную ссылку

void process(BankAccount& account) {

...

}

Если вы передаёте объект в функцию и код не изменит вида объекта — передайте его как const-ссылку:

// Плохая практика

// Принимает указатель

void display(BankAccount account) {

...

}

// Хорошая практика

// Принимает константную ссылку

void display(const BankAccount& account) {

...

}

Избегайте «цепных» вызовов, когда множество функций вызывают друг друга по цепочке, не возвращая значение в main. Убедитесь, что main является кратким описанием всей программы:

// Плохая практика

main

|

+-- function1

|

+-- function2

|

+-- function3

|

+-- function4

|

+-- function5

|

+-- function6

// Хорошая практика

main

|

+-- function1

|

+-- function2

| |

| +-- function3

| |

| +-- function4

|

+-- function5

| |

| +-- function6

**Проектирование классов**

**Инкапсуляция**. Отделяйте ваши объекты, делая все поля данных в вашем классе private:

class Student {

private:

int homeworkScore;

...

**.h vs .cpp.** Всегда размещайте объявления классов и их частей в собственные файлы, ClassName.h. Всегда сворачивайте файлы объявления классов .h в блок препроцессоров #ifndef / define / endif, чтобы избежать множественных объявлений одного класса:

// Point.h

#ifndef \_point\_h

#define \_point\_h

class Point {

public:

Point(int x, int y);

int getX() const;

int getY() const;

void translate(int dx, int dy);

private:

int m\_x;

int m\_y;

};

#endif

// Point.cpp

#include "Point.h"

Point::Point(int x, int y) {

m\_x = x;

m\_y = y;

}

void Point::translate(int dx, int dy) {

m\_x += dx;

m\_y += dy;

}

...

**class vs struct.** Всегда используйте class, только если вы не создаёте очень маленький и простой вид данных, для которого нужны лишь несколько публичных переменных и, возможно, конструктор для их инициализации.

**Избегайте ненужных полей**. Используйте поля, чтобы хранить важные данные о ваших объектах, но не временные значения, которые используются единожды.