# **Конспект №1: #include guard, enum, using.**

## *Include Guard в С++*

Во избежание ошибок при наложении заголовочных файлов используется #include guard, иначе называемая защитой подключения. Данная конструкция состоит из директив #ifndef и #define:

#ifndef H\_BMW // Проверка наличия макроса H\_BMW, а если его нет

#define H\_BWM // То определяется макрос и подключается код

int name() {

// Исполняемый код...

}

#endif // Граница исполняемого директивой #ifndef кода

Итак, если заголовочный файл был подключён, его повторное подключение не вызовет ошибки. Аналогично #ifndef используется #pragma once:

#pragma once // Определяется только в начале

int name() {

// Исполняемый код...

}

Источники:

* [защита подключения](https://ru.wikipedia.org/wiki/Include_guard)
* [макрос](https://www.opennet.ru/docs/RUS/cpp/cpp-5.html)
* [директива #ifndef](http://www.c-cpp.ru/books/ifdef-u-ifndef)
* [директива #define](http://www.c-cpp.ru/books/define)

## *Перечисляемый тип в C++*

Для удобочитаемости кода и визуальной группировки безымянных параметров (цвет, звук, сигнал) используются перечисляемые типы. Они задаются ключевым словом enum. Например:

#include <iostream>

enum Colors {

COLOR\_RED, // Присваивается значение 0

COLOR\_GREEN, // ... значение 1

COLOR\_BLUE, // ... значение 2

};

int main() {

Colors tomato = COLOR\_RED; // Красный цвет - 0

Colors cucumber = COLOR\_GREEN; // Зелёный цвет - 1

Colors berry = COLOR\_BLUE; // Синий цвет - 2

std::cout << tomato <<std::endl;

return 0;

}

Перечисляемый тип является группировочным аналогом констант и используется для избавления от магических чисел. Перечисляемые типы можно изменять, но нельзя вводить через std::cin.

Понятие «Магическое число» в программировании имеет три значения:

* Сигнатура данных
* Выделенные уникальные значения, которые не должны совпадать с другими значениями (например, UUID)
* Плохая практика программирования

Источники:

* [перечисляемый тип](https://ravesli.com/urok-58-perechisleniya-tip-enum/)
* [магическое число](https://ravesli.com/urok-36-literaly-magicheskie-chisla/#toc-3)

## *Ссылки на функции в C++ с использованием using*

Для удобочитаемости кода используются using-стейтменты. Они значительно сокращают время написания кода и его читабельность, однако повышают вероятность ошибок. Обычный вариант:

#include <iostream>

int main() {

std::cout << 450 << endl;

}

Использование using-объявления сокращает конкретную функцию, определенную в пространстве имен, не затрагивая остальные функции. Этот способ не приводит к большим рискам.

#include <iostream>

int main() {

using std::cout;

cout << 450 << endl;

}

Для глобального исключения упоминания пространства имён используется using-директива. Этот способ значительно упрощает написание и читаемость кода, однако приводит к значительным рискам.

#include <iostream>

int main() {

using namespase std;

cout << 450 << endl;

}

Под рисками понимается вероятность совпадения программного идентификатора какой-либо переменной или функции с ключевым словом функции, определяемой библиотекой без пространства имён. Помимо этого using используется для упрощения сложных конструкций, например указателя на функцию:

int foo(int a) {

return a;

}

int main() {

using uzel = int(\*)(int);

uzel f = foo;

int b = f(7);

return b;

}

Источники:

* [пространство имен](https://ravesli.com/urok-53-prostranstva-imen/)
* [using-стейтменты](https://ravesli.com/urok-54-using-statements/)
* [указатели на функции](https://ravesli.com/urok-104-ukazateli-na-funktsii/)