



CLion

Авто дополнение

CLion помогает писать код быстрее с помощью функции авто дополнения. Начните вводить ключевое слово, имя переменной, функции или класса — CLion предложит список подходящих вариантов авто дополнения. Чтобы сэкономить еще больше времени, вы можете ввести только заглавные буквы имени или даже любую его часть. Кроме того, IDE предлагает варианты авто дополнения для символов из внешних библиотек и фреймворков, которые используются в вашем проекте. Постфиксное авто дополнение для C и C++ подставляет нужный код с обеих сторон введенного выражения. Оно поможет заключить выражение в часто используемую языковую конструкцию или передать его в качестве первого аргумента свободной функции.

```
typedef struct PersonData {  
    string name;  
    string family_name;  
    int age;  
    int id;  
} PersonData;  
  
void handlePerson(PersonData *p) {  
    string str = p->  
}
```

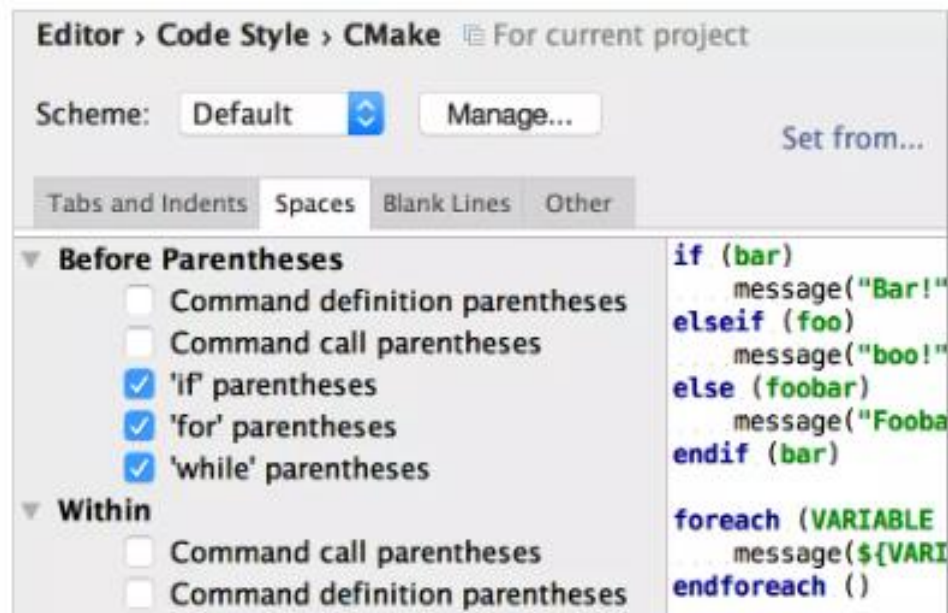


Стиль кода

CLion помогает поддерживать единый стиль оформления кода и соблюдать настройки форматирования. Настройки применяются автоматически, когда вы пишете код, а также с помощью действия Reformat Code

Стиль кода настраивается в меню. Вы можете настроить общую конфигурацию или опции для конкретного языка (например, C/C++, CMake, HTML или XML).

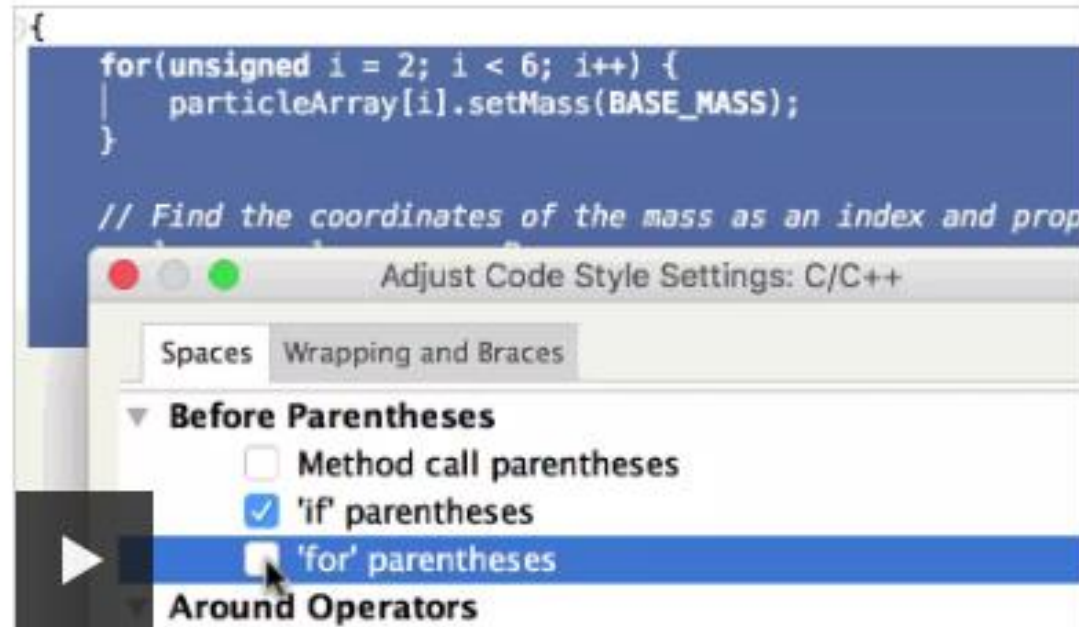
Настройки стиля оформления кода определяют правила выравнивания, расстановки пробелов, табуляции и отступов, генерации кода и использования пустых строк. Если настройка стиля влияет на код, CLion предложит предварительный просмотр, выделив строку, затронутую изменением.



Расширенные настройки

CLion позволяет не применять некоторые настройки форматирования к коду в выбранных файлах. Если отступы в файле отличаются от действующих настроек форматирования, CLion предупредит об этом при открытии файла.

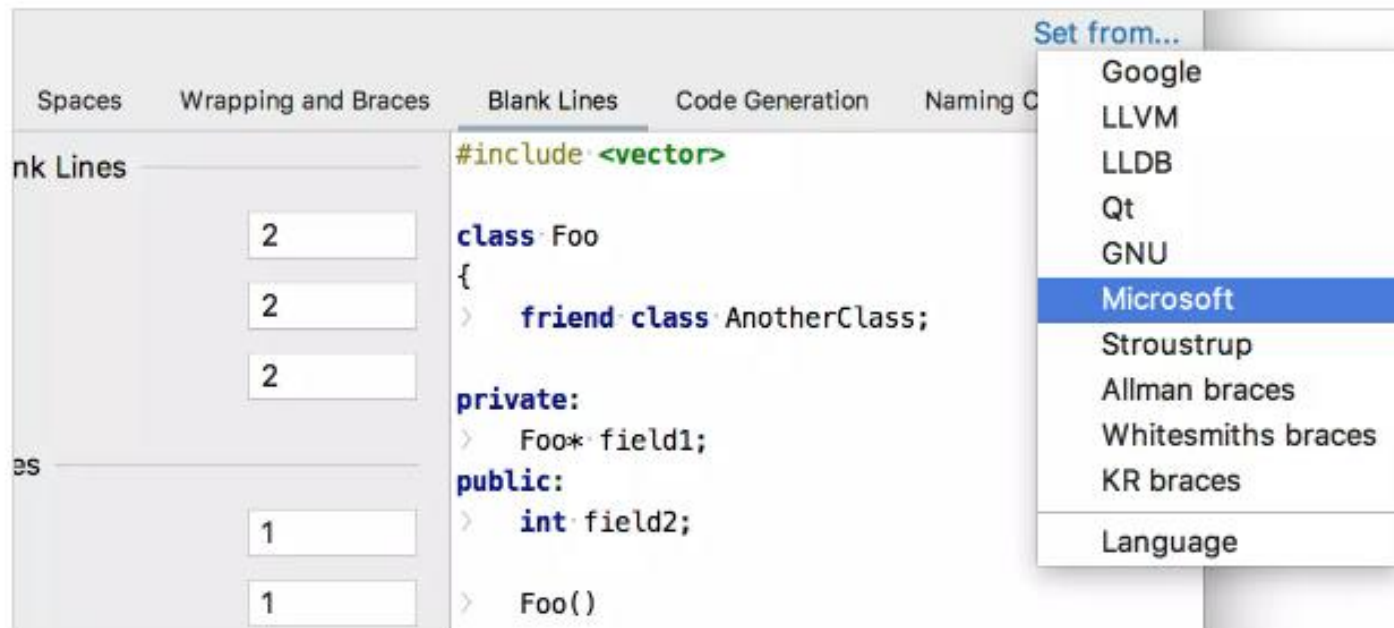
Вы можете выбрать принудительное применение настроек стиля или оставить существующие отступы. Эту функцию можно



Предопределенные стили оформления кода

Вы можете выбрать предопределенный стиль оформления кода и применить его к вашему проекту. IDE поддерживает стандартные стили [Google](#), [LLVM](#), [LLDB](#), [GNU](#), [Microsoft](#), [Qt](#) и [Stroustrup](#), а также правила расстановки фигурных скобок [Allman](#), [Whitesmiths](#) и [K&R](#). Выберите подходящий стиль в меню

Стили применяются к настройкам форматирования, правилам именования C/C++, стилям header guards.



Правила именования

CLion соблюдает заданную схему именования. Выбранные настройки будут использоваться во всех действиях IDE, включая:

Автодополнение

Генерацию кода

Рефакторинг кода

Применение быстрых исправлений

На вкладке Naming Convention в настройках Settings | Editor | Code Style | C/C++ можно выбрать один из predefined стилей или вручную настроить свой.

Header Guard Style: **`_${PROJECT_NAME}_${FILE_NAME}_${EXT}`**

Entity Kind	Naming Convention	Visibility	Specifier	Prefix	Suffix
Namespace	▼ camelCase ▼				
Macro	▼ SCREAMING_SNAKE_CASE ▼				
Class	▼ camelCase ▼				
Enum	▼ camelCase ▼				
Enumerator	▼ PascalCase ▼				
Typedef	▼ camelCase ▼				
Union	▼ camelCase ▼				
Member Function	▼ camelCase ▼	Any ▼	Any ▼		
Member Field	▼ camelCase ▼	Any ▼	Any ▼		
Global Function	▼ camelCase ▼				
Global Variable	▼ camelCase ▼		Any ▼		
Parameter	▼ camelCase ▼		Any ▼		
Local Variable	▼ camelCase ▼		Any ▼		

Подсказки параметров

Благодаря подсказкам имен параметров вам не придется переключаться на сигнатуру функции во время изучения вызова функции. В свою очередь, это помогает улучшить читаемость вашего кода.

Для вызовов функций, лямбда-выражений, конструкторов, списков инициализаторов и макросов Clion показывает имена параметров для переданных аргументов. Подсказки работают, если аргументом является литерал или выражение с более чем одним операндом.

```
std::vector<int> myvec( n: len);
std::generate( first: myvec.begin(), last: myvec.end(),
               gen: [] -> int { return rand() % 100; });

print_vec( vec: myvec, label: std::string{ s: "Let's start with: "});

sort_vec_iter(myvec);
print_vec( vec: myvec, label: std::string{ s: "Let's start with: "});
sort_vec_loop(len, myvec);
print_vec( vec: myvec, label: std::string{ s: "Let's start with: "});
```

Подсказки типов

Для улучшения читаемости кода в CLion показываются подсказки с выведенными типами. Эти подсказки полезны для переменных типа `auto`, в конструкциях структурного связывания и для типов значений, возвращаемых из лямбд.

Включить или отключить подсказки тех или иных типов можно в меню *Settings | Editor | Inlay Hints | C/C++* или прямо в контекстном меню подсказки.

```
template<typename T, typename U>
auto doOperation(T t, U u) -> decltype(t + u) -> <dependent type> {
    return t + u;
}

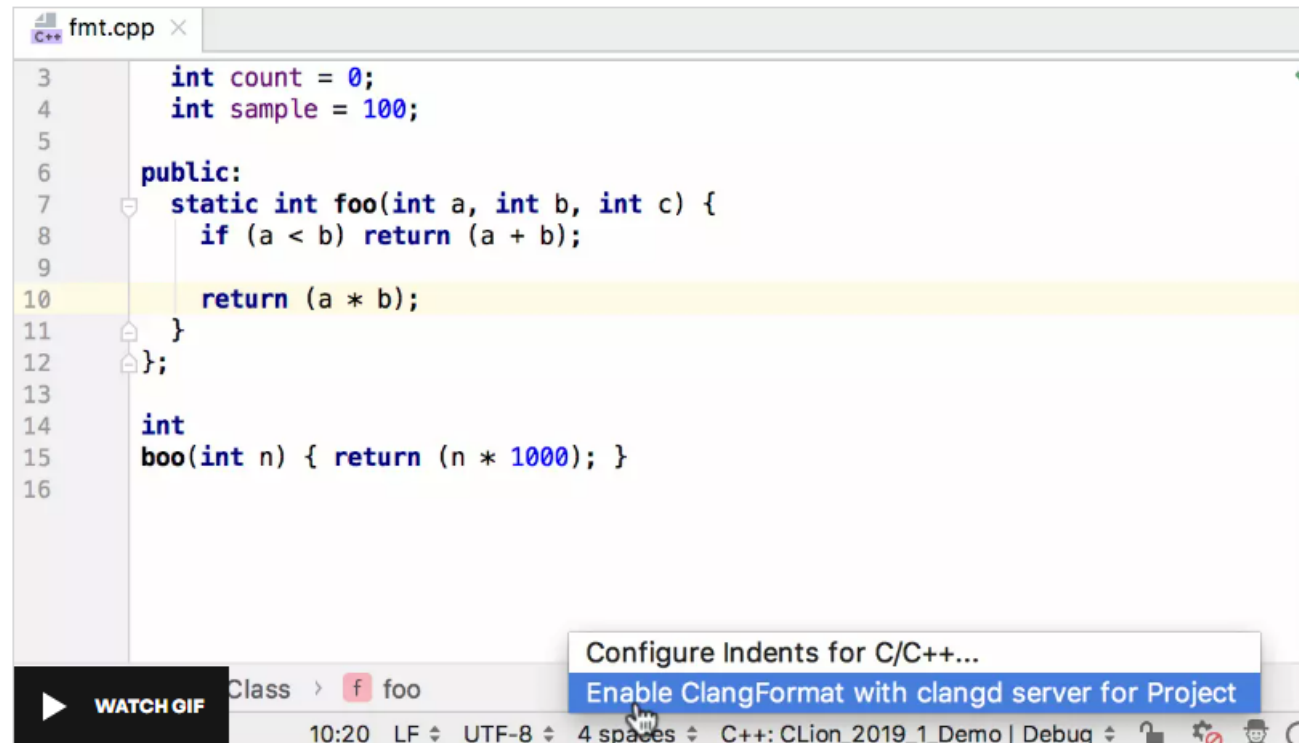
void fun_type() {
    auto op : int = doOperation( t: 3, u: 0);
    auto op1 : long = doOperation( t: 3L, u: 0);
    auto op2 : double = doOperation( t: 3.0, u: 0);

    std::cout << op << " " << op1 << " " << op2;
}
```


ClangFormat

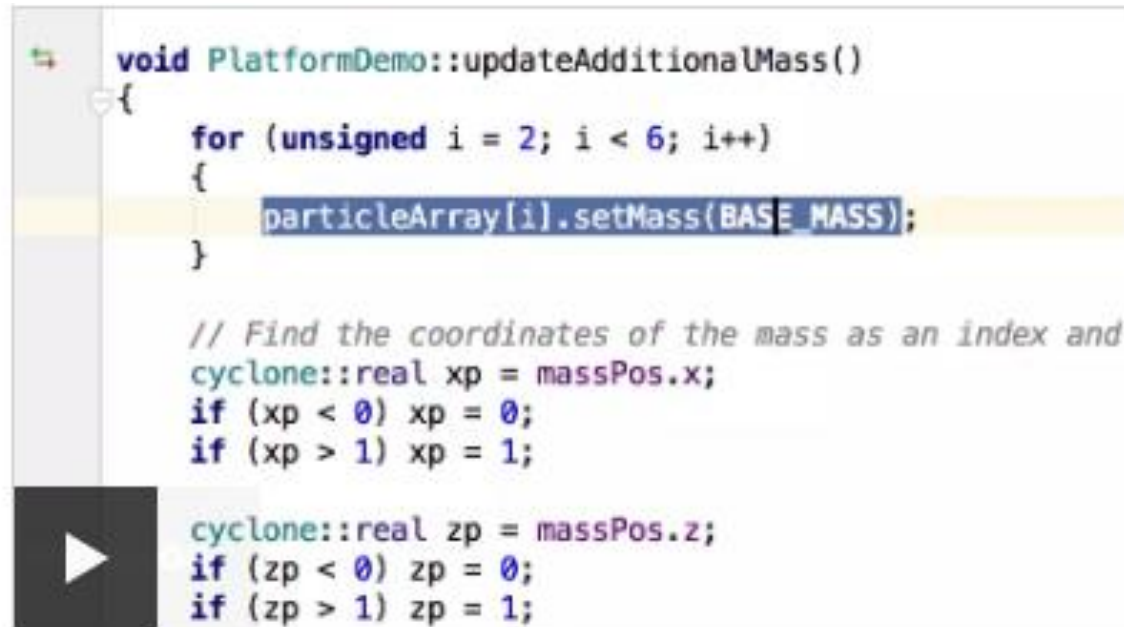
В качестве альтернативного средства форматирования кода CLion поддерживает инструмент ClangFormat. Вы можете включить его в настройках и использовать для форматирования кода как в текущем проекте, так и во всех проектах CLion.

Если в проекте будет обнаружен конфигурационный файл *.clang-format*, CLion предложит переключиться на ClangFormat.



Выделение фрагментов кода и комментарии

Чтобы быстро выделить блок кода, поместите в него курсор и нажмите необходимое количество раз, чтобы расширить выделение до конца выражения, строки, логического блока и так далее. Аналогичным образом, чтобы снять выделение с логической части, нажмите столько раз, сколько нужно.



```
void PlatformDemo::updateAdditionalMass()
{
    for (unsigned i = 2; i < 6; i++)
    {
        particleArray[i].setMass(BASE_MASS);
    }

    // Find the coordinates of the mass as an index and
    cyclone::real xp = massPos.x;
    if (xp < 0) xp = 0;
    if (xp > 1) xp = 1;

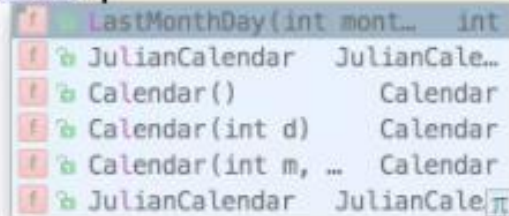
    cyclone::real zp = massPos.z;
    if (zp < 0) zp = 0;
    if (zp > 1) zp = 1;
```

Многокурсорность

Clion позволяет делать несколько вещей одновременно. В режиме многокурсорного редактирования вы можете вносить изменения в файл в нескольких местах одновременно.


Выберите места для редактирования (нажмите и поместите курсор в нужные места) или просто добавьте пару следующих совпадений в выделение с помощью

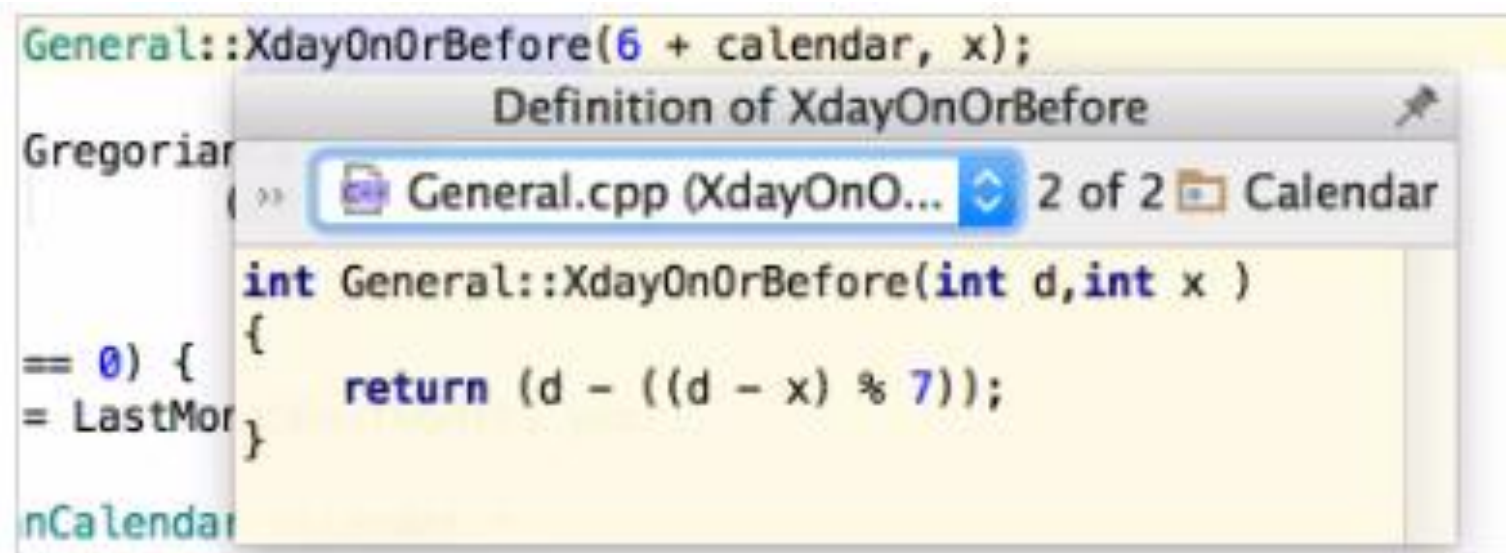
```
gregorian_calendar = a;  
a = gregorian_calendar;  
cout << "    = GregorianCalendar date " << gregorian_calendar  
    << " = absolute date " << a  
    << " & last data "  
    << gregorian_calendar.L  
    << "\n";  
  
JulianCalendar julian_calendar(a);  
a = julian_calendar;  
cout << "    = JulianCalendar date " << julian_calendar  
    << " = absolute date " << a  
    << " & last data "  
    << julian_calendar.L  
    << "\n";
```



LastMonthDay(int mont...	int
JulianCalendar	JulianCale...
Calendar()	Calendar
Calendar(int d)	Calendar
Calendar(int m, ...	Calendar
JulianCalendar	JulianCale...

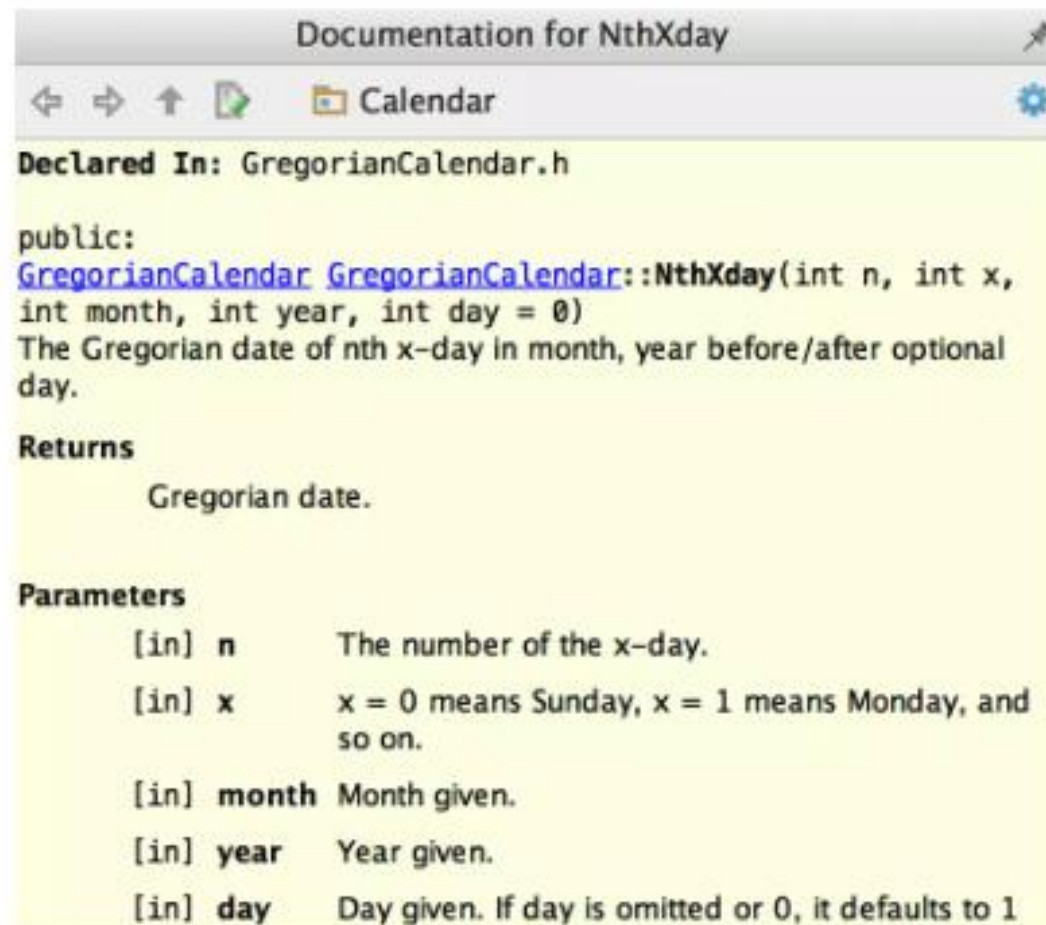
Быстрый просмотр определений

Посмотреть реализацию или объявление функции можно прямо в редакторе, не переходя на другой файл. Просто поместите курсор на нужный символ и нажмите  чтобы открыть всплывающее окно Quick Definition.



Быстрый просмотр документации

В CLion есть специальное окно для быстрого просмотра документации (Quick Documentation popup, вызывается нажатием в котором отображается документация для выбранного класса, функции, переменной, параметра или макроса.



Информация о параметрах

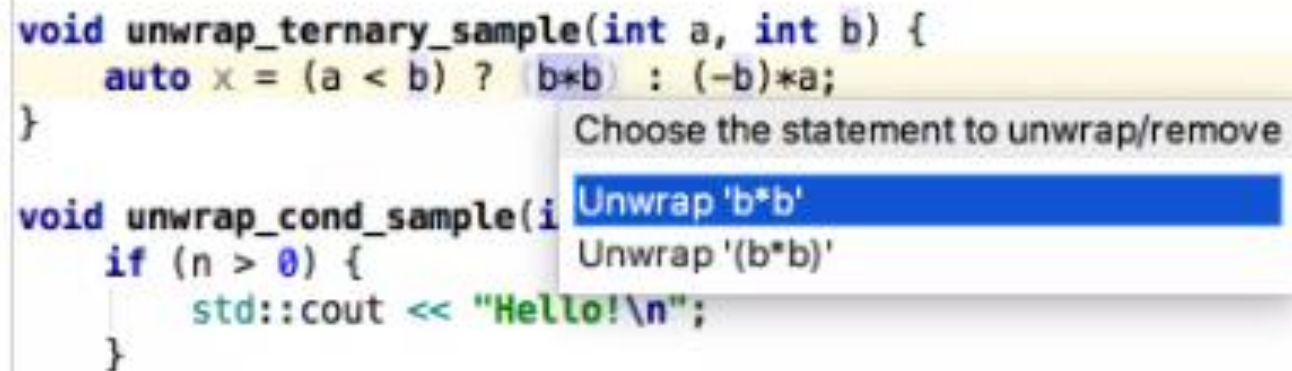
CLion придет на помощь, если вы не уверены, какие параметры принимает функция. Вызовите информацию о параметрах функции чтобы увидеть все доступные сигнатуры и параметры функции. При редактировании параметров CLion скроет все неподходящие сигнатуры.

```
GregorianCalendar gregorian_calendar(month, day, );  
int a = gregorian_calendar;  
cout << gregorian_calendar  
    << " = " << a << " = "  
    << DayName[gregorian_calendar % 7]
```

int date
int m, int d, int y

Действие Unwrap/Remove

Чтобы безопасно удалять фрагменты в сложном коде со множеством вложенных операторов, используйте действие Unwrap/Remove... IDE предлагает разные варианты преобразований в зависимости от того, где находится курсор, и умеет вынимать код из управляющих конструкций



```
void unwrap_ternary_sample(int a, int b) {  
    auto x = (a < b) ? b*b : (-b)*a;  
}  
  
void unwrap_cond_sample(int n) {  
    if (n > 0) {  
        std::cout << "Hello!\n";  
    }  
}
```

Choose the statement to unwrap/remove

- Unwrap 'b*b'
- Unwrap '(b*b)'

Автоматическое импортирование

Когда вы используете символ, который еще не был импортирован, CLion выполнит поиск и предложит добавить нужную директиву `#include` или сделает это автоматически.

