

Ravesli [Ravesli](#)


- [Уроки по C++](#)
- [OpenGL](#)
- [SFML](#)
- [Qt5](#)
- [RegExр](#)
- [Ассемблер](#)
- [Купить .PDF](#)

Урок №23. Header guards и #pragma once

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 |

 49963

[↑](#)  29

На этом уроке мы рассмотрим, что такое header guards и #pragma once в языке C++, а также зачем они нужны и как их правильно использовать.

Оглавление:

1. [Проблема дублирования объявлений](#)
2. [Header guards](#)
3. [#pragma once](#)
4. [Тест](#)

Проблема дублирования объявлений

Как мы уже знаем из урока о [предварительных объявлениях](#), идентификатор может иметь только одно объявление. Таким образом, программа с двумя объявлениями одной переменной получит ошибку компиляции:

```
1 int main()
2 {
3     int x; // это объявление идентификатора x
4     int x; // ошибка компиляции: дублирование объявлений
5
6     return 0;
7 }
```

То же самое касается и функций:

```
1 #include <iostream>
2
```

```
3 | int boo()
4 | {
5 |     return 7;
6 | }
7 |
8 | int boo() // ошибка компиляции: дублирование определений
9 | {
10 |     return 7;
11 | }
12 |
13 | int main()
14 | {
15 |     std::cout << boo();
16 |     return 0;
17 | }
```

Хотя вышеприведенные ошибки легко исправить (достаточно просто удалить дублирование), с заголовочными файлами дела обстоят несколько иначе. Довольно легко можно попасть в ситуацию, когда определения одних и тех же заголовочных файлов будут подключаться больше одного раза в файл .cpp. Очень часто это случается при подключении одного заголовочного файла другим.

Рассмотрим следующую программу:

math.h:

```
1 | int getSquareSides()
2 | {
3 |     return 4;
4 | }
```

geometry.h:

```
1 | #include "math.h"
```

main.cpp:

```
1 | #include "math.h"
2 | #include "geometry.h"
3 |
4 | int main()
5 | {
6 |     return 0;
7 | }
```

Эта, казалось бы, невинная программа, не скомпилируется! Проблема кроется в определении функции в файле math.h. Давайте детально рассмотрим, что здесь происходит:

- ➔ Сначала main.cpp подключает заголовочный файл math.h, вследствие чего определение функции getSquareSides копируется в main.cpp.
- ➔ Затем main.cpp подключает заголовочный файл geometry.h, который, в свою очередь, подключает math.h.

→ В `geometry.h` находится копия функции `getSquareSides` (из файла `math.h`), которая уже во второй раз копируется в `main.cpp`.

Таким образом, после выполнения всех [директив #include](#), `main.cpp` будет выглядеть следующим образом:

```
1 int getSquareSides() // из math.h
2 {
3     return 4;
4 }
5
6 int getSquareSides() // из geometry.h
7 {
8     return 4;
9 }
10
11 int main()
12 {
13     return 0;
14 }
```

Мы получим дублирование определений и ошибку компиляции. Если рассматривать каждый файл по отдельности, то ошибок нет. Однако в `main.cpp`, который подключает сразу два заголовочных файла с одним и тем же определением функции, мы столкнемся с проблемами. Если для `geometry.h` нужна функция `getSquareSides()`, а для `main.cpp` нужен как `geometry.h`, так и `math.h`, то какое же решение?

Header guards

На самом деле решение простое — использовать header guards (защиту подключения в языке C++).

Header guards — это директивы [условной компиляции](#), которые состоят из следующего:

```
1 #ifndef SOME_UNIQUE_NAME_HERE
2 #define SOME_UNIQUE_NAME_HERE
3
4 // Основная часть кода
5
6 #endif
```

Если подключить этот заголовочный файл, то первое, что он сделает — это проверит, был ли ранее определен идентификатор `SOME_UNIQUE_NAME_HERE`. Если мы впервые подключаем этот заголовок, то `SOME_UNIQUE_NAME_HERE` еще не был определен. Следовательно, мы определяем `SOME_UNIQUE_NAME_HERE` (с помощью директивы `#define`) и выполняется основная часть заголовочного файла. Если же мы раньше подключали этот заголовочный файл, то `SOME_UNIQUE_NAME_HERE` уже был определен. В таком случае, при подключении этого заголовочного файла во второй раз, его содержимое будет проигнорировано.

Все ваши заголовочные файлы должны иметь header guards. `SOME_UNIQUE_NAME_HERE` может быть любым идентификатором, но, как правило, в качестве идентификатора используется имя заголовочного файла с окончанием `_H`. Например, в файле `math.h` идентификатор будет `MATH_H`:

`math.h`:

```
1 #ifndef MATH_H
2 #define MATH_H
3
4 int getSquareSides()
5 {
6     return 4;
7 }
8
9 #endif
```

Даже заголовочные файлы из Стандартной библиотеки C++ используют header guards. Если бы вы взглянули на содержимое заголовочного файла `iostream`, то вы бы увидели:

```
1 #ifndef _IOSTREAM_
2 #define _IOSTREAM_
3
4 // основная часть кода
5
6 #endif
```

Но сейчас вернемся к нашему примеру с `math.h`, где мы попытаемся исправить ситуацию с помощью header guards:

`math.h`:

```
1 #ifndef MATH_H
2 #define MATH_H
3
4 int getSquareSides()
5 {
6     return 4;
7 }
8
9 #endif
```

`geometry.h`:

```
1 #include "math.h"
```

`main.cpp`:

```
1 #include "math.h"
2 #include "geometry.h"
3
4 int main()
5 {
6     return 0;
7 }
```

Теперь, при подключении в `main.cpp` заголовочного файла `math.h`, препроцессор увидит, что `MATH_H` не был определен, следовательно, выполнится директива определения `MATH_H` и содержимое `math.h` скопируется в `main.cpp`. Затем `main.cpp` подключает заголовочный файл `geometry.h`, который, в свою

очередь, подключает `math.h`. Препроцессор видит, что `MATH_H` уже был ранее определен и содержимое `geometry.h` не будет скопировано в `main.cpp`.

Вот так можно бороться с дублированием определений с помощью `header guards`.

#pragma once

Большинство компиляторов поддерживают более простую, альтернативную форму `header guards` — директиву `#pragma`:

```
1 #pragma once
2
3 // основная часть кода
```

`#pragma once` используется в качестве `header guards`, но имеет дополнительные преимущества — она короче и менее подвержена ошибкам.

Однако, `#pragma once` не является официальной частью языка C++, и не все компиляторы её поддерживают (хотя большинство современных компиляторов поддерживают).

Я же рекомендую использовать `header guards`, чтобы сохранить максимальную совместимость вашего кода.

Тест

Добавьте `header guards` к следующему заголовочному файлу:

`add.h`:

```
1 int add(int x, int y);
```

Ответ

```
1 #ifndef ADD_H
2 #define ADD_H
3
4 int add(int x, int y);
5
6 #endif
```

Оценить статью:

★★★★★ (494 оценок, среднее: 4,85 из 5)



← [Урок №22. Директивы препроцессора](#)

[Урок №24. Конфликт имен и std namespace](#)

Комментариев: 29



1. Роман:

[7 мая 2020 в 22:54](#)

Я рад, что нашёл это, когда я смотрел урок за уроком, моя реакция была что-то типа: "Вааау, так вот оно что", всё понятно описано и даже такой ленивый человек, как я смог прочитать всё и извлечь из этого пользу, спасибо

[Ответить](#)

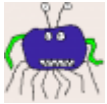


1. Юрий:

[8 мая 2020 в 12:54](#)

Пожалуйста))

[Ответить](#)



2. Artemiy:

[17 января 2020 в 19:38](#)

2019 VS и твои уроки — просто! БОМБА! Давай больше материалов и создай раздел отдельный для тестов, хочу больше задач!

[Ответить](#)



1. Юрий:

[18 января 2020 в 01:00](#)

Вот [задачки по C++](#) уже есть, решай 😊

[Ответить](#)



3. Sasha:

[7 января 2020 в 15:25](#)

Решил поэкспериментировать и убрал header guards с заголовочного файла, оставил чисто прототип функции. Затем подключил заголовочный файл несколько раз. И это не вызвало конфликта имен, почему? Пробовал и в visual studio и в code::blocks

[Ответить](#)



4. Владимир:

[14 ноября 2019 в 15:15](#)

Получилось вот что:

main.cpp

```
1 #include<iostream>
2 #include"input.h"
3 using namespace std;
4
5 int main()
6 {
7     int x = getinteger();
8     int y = getinteger();
9     cout << x << "+" << y << "is" << x + y << endl;
10 }
```

input.cpp

```
1 #include<iostream>
2 using namespace std;
3
4 int getinteger()
5 {
6     int x;
7     cout << "Enter an integer";
8     cin >> x;
9     return x;
10 }
```

input.h

```
1 #ifndef INPUT_H
2 #define INPUT_H
3     int getinteger();
4 #endif
```

[Ответить](#)



5. *Владимир:*

[14 ноября 2019 в 14:56](#)

Вчера читал уроки с 20 по 23 и не понял почти ничего, в голове все перемешалось. Сегодня перечитал и понял абсолютно все. Ура! Классная форма изложения! Непонятка остается только в одном — объявление или все же определение функций в заголовочных файлах.

[Ответить](#)

R

1. *Юрий:*

[14 ноября 2019 в 23:12](#)

Объявление в заголовочных файлах, определение в основном в файлах .cpp.

[Ответить](#)



6. Евгений:

[26 августа 2019 в 15:26](#)

Хотел бы поинтересоваться у автора статьи вот чем... Вы постоянно используете выражения на английском языке в русскоязычной статье. И тут я говорю не про куски кода в примерах, а например про "Header guards". Почему нельзя это назвать по русски? Получается статья на двух языках. Конечно, сейчас нет проблем переводом англоязычных текстов, да и программисту полезно, и даже необходимо знать английский, но выглядит это, откровенно говоря, нелепо.

[Ответить](#)



1. Юрий:

[26 августа 2019 в 20:07](#)

Значит это нелепые уроки, вы можете найти другие "лепые" уроки и спокойно их читать)

[Ответить](#)



1. Игорь:

[8 сентября 2019 в 18:50](#)

Не перевести такое простое выражение -это крайняя невежественность. Да и империалисты пишут программы почему-то на английском, а не русском. вот сволочи..

[Ответить](#)



1. Евгений:

[23 января 2020 в 13:47](#)

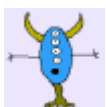
Перевести — это другой вопрос, и то, на каком языке пишут программы, также другой вопрос. Статья русскоязычная, соответственно и вся терминология должна быть на русском. Header guards в контексте данной статьи — жаргон! А использовать жаргон в технической литературе это признак необразованности.



2. Юрий:

[23 января 2020 в 23:34](#)

Интересная логика)

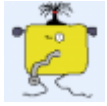


7. zashiki:

[5 июня 2019 в 11:44](#)

Здравствуйтесь, кстати, а почему прототип функции можно дублировать сколько угодно?

[Ответить](#)



8. *Easy:*

[30 апреля 2019 в 16:44](#)

Суть `define` плохо раскрыта. Он выполняет разные действия и что между этими действиями общего здесь не показано. Ранее он удалял то, что рядом написано, например функция `define NAMEFUNCTION`, которая указана рядом, будет удалена в будущем. А теперь он копирует содержимое заголовочного файла? `Define` переводится как "Определять". По логике для безопасного копирования заголовочного файла достаточно `#ifndef NAMEFILE_H`если он ещё не был скопирован, то копируется, а если уже был скопирован, тогда не копируется. Поэтому мне не ясно для чего после `#ifndef NAMEFILE_H` нужен `#define NAMEFILE_H`

[Ответить](#)



9. *Easy:*

[26 апреля 2019 в 16:41](#)

Я запутался с этим `define`. Почему в прошлом уроке он удалял и заменял, а здесь наоборот?

[Ответить](#)



10. *Alex:*

[14 февраля 2019 в 13:44](#)

Старина!

Крутой курс!

Реально простым языком!

Круто, что ты пишешь о таких вещах, о которых обычно никто и не пишет (например `header guards`).

все по-порядку и обо всех тонкостях!

Это реально "для начинающих".

Буда многих курсов - они сразу приучают будущих быдлокодеров в принципе "не думай как работает эта библиотека/функция, просто юзай ее". Я ни разу не встречал чтоб кто-то хоть чуток попытался объяснить зачем эти `namespace`, `#ifdef`.

Дружище, я с радостью приобрету PDF вариант, но в полном объеме, все 200+ уроков, что есть на данный момент.

Не проблема заплатить в 2 раза больше, я вижу содержаник курса и у меня перья шевелятся от осознания как много времени ты потратил на подготовку материала!!!

Мне напрашивается мысль о трех вариантах PDFок- 100 уроков, и 200 уроков, и 200+ уроков, с разными ценниками.

Продолжай!

[Ответить](#)



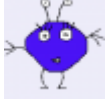
1. *Юрий:*

[14 февраля 2019 в 17:10](#)

Хорошо, зафиксировал. Спасибо, что читаешь 😊

[Ответить](#)11. *Kaladgan:*[7 февраля 2019 в 17:12](#)

На счет добавления pch.h во все файлы проекта, у меня почему-то компилятор ругался на все файлы в которых не было этой строчки #include "pch.h" Когда добавил во все 3 файла только тогда все нормально отработало.

[Ответить](#)12. *nickatin:*[27 января 2019 в 22:59](#)

Юрий, очень приятно читать ваши уроки. Все очень доступно!

[Ответить](#)1. *Юрий:*[28 января 2019 в 13:07](#)

И мне приятно, что читаете 😊

[Ответить](#)13. *Коля:*[29 декабря 2018 в 13:18](#)

Добрый день, подскажите пожалуйста, в чём может быть проблема. Сделал все по уроку, проверил несколько раз, использовал и header guards и #pragma once, все равно выдает ошибку. Использую Visual Studio 2017.

[Ответить](#)1. *Юрий:*[29 декабря 2018 в 17:16](#)

Код добавьте.

[Ответить](#)1. *Коля:*[29 декабря 2018 в 19:16](#)

math.h

```
1 | #include "pch.h"
2 | #ifndef MATH_H
3 | #define MATH_H
4 |
```

```

5 | int getSquareSides()
6 | {
7 |     return 4;
8 | }
9 |
10| #endif

```

geometry.h

```

1 | #include "pch.h"
2 | #include "math.h"

```

main.cpp

```

1 | #include "pch.h"
2 | #include "math.h"
3 | #include "geometry.h"
4 |
5 |
6 | int main()
7 | {
8 |     return 0;
9 | }

```

Ответить



1. *Юрий:*
[31 декабря 2018 в 03:07](#)

Во-первых, предварительно скомпилированный заголовок pch.h подключается в угловых скобках, а не в кавычках:

```
1 | #include <pch.h>
```

Во-вторых, зачем вы его подключаете во всех файлах одного и того же проекта? Это вызовет ошибку дублирования кода. Подключать нужно только в main.cpp.

В-третьих, зачем вы подключаете в main.cpp файлы math.h и geometry.h? Файл geometry.h уже подключает math.h и используется для того, чтобы не прописывать в main.cpp строчку:

```
1 | #include "math.h"
```

Перечитайте внимательнее этот урок и два предыдущих.



2. *Коля:*
[16 января 2019 в 14:10](#)

Перечитал, и правда был не внимателен в предыдущих уроках, все получилось, спасибо вам, за ваш проект

[Ответить](#)

1. Юрий:
[16 января 2019 в 21:36](#)

Пожалуйста 😊



14. Олег:
[7 ноября 2018 в 15:31](#)

Не совсем понятно. В предыдущем уроке вы говорили, что директивы распространяются только на код, написанный в данном файле, а здесь выходит так, что `#ifndef` из `geometry.h` видит `#define` из `math.h`.

[Ответить](#)

1. Андрей:
[4 декабря 2018 в 05:42](#)

Он видит `#define` из `math.h` потому, что весь `math.h` был `#include` в `geometry.h`. В том числе и со всеми его `#define`. А `geometry.h` в свою очередь уже был `#include` в `main.cpp`, т.е. полностью переписан туда. Так же, как и `math.h`.

В итоге, когда препроцессор закончит (надеюсь так правильно говорить) со всеми нашими `.h` файлами (но не закончит еще с `main.cpp`), мы получим какой-то такой "ужас", хотя по сути и не увидим его глазами (наверное поэтому сложно это воспринимать?):

```
1 // здесь тоже должен быть переписан весь набор iostream, а не эта строка
2 #include <iostream>
3
4 #ifndef MATH_H
5 #define MATH_H
6
7 int getSquareSides()
8 {
9     return 4;
10 }
11
12 #endif // MATH_H
13
14 #ifndef GEOMETRY_H
15 #define GEOMETRY_H
16
17 #ifndef MATH_H
18 #define MATH_H
19
20 int getSquareSides()
21 {
22     return 4;
23 }
24
```

```
25 #endif // MATH_H
26
27 #endif // GEOMETRY_H
28
29 using namespace std;
30
31 main()
32 {
33     cout << "Square sides: " << getSquareSides();
34     return 0;
35 }
```

И только потом будут выполнены следующие команды препроцессора в main.cpp (помним, что они выполняются по очереди, т.е. сначала переписывается весь #include, а потом ниже по тексту все наши #define), после чего все это уже компилируется и отработывает. А cout вполне отлично будет выводить циферку 4.

P.S.

Я наверное так "хорошо" объяснил этот момент, что стало только еще более непонятно?)) А вообще хотелось бы услышать и ответ Юрия — так оно или не так работает? Вдруг я не правильно понял, но решил что правильно только потому, что оно у меня работает?)) Хотя если работает, то видимо так и правильно)

[Ответить](#)

Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены *

Имя *

Email *

Комментарий

☐ Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию

☐ Получать уведомления о новых комментариях по электронной почте. Вы можете [подписаться](#) без комментирования.






[TELEGRAM](#)  [КАНАЛ](#)



[ПАБЛИК](#) 



ТОП СТАТЬИ

-  [Словарь программиста. Сленг, который должен знать каждый кодер](#)
-  [Урок №1. Введение в программирование](#)
-  [70+ бесплатных ресурсов для изучения программирования](#)
-  [Урок №1: Введение в создание игры «Same Game»](#)
-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

- [Ravesli](#)
- - [О проекте](#) -
- - [Пользовательское Соглашение](#) -
- - [Все статьи](#) -
- Copyright © 2015 - 2020