

Ravesli [Ravesli](#)

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



Урок №21. Заголовочные файлы

[Юрий](#) |

- [Уроки C++](#)

|

Обновл. 2 Сен 2020 |

122983

[↓](#) 58

По мере увеличения размера программ весь код уже не помещается в нескольких файлах, записывать каждый раз **предварительные объявления** для функций, которые мы хотим использовать, но которые находятся в других файлах, становится всё утомительнее и утомительнее. Хорошо было бы, если бы все предварительные объявления находились в одном месте, не так ли?

Файлы .сpp не являются единственными файлами в проектах. Есть еще один тип файлов — **заголовочные файлы** (или «*заголовки*»), которые имеют расширение .h. Целью заголовочных файлов является удобное хранение набора объявлений объектов для их последующего использования в других программах.

Оглавление:

1. [Заголовочные файлы из Стандартной библиотеки C++](#)
2. [Пишем свои собственные заголовочные файлы](#)
3. [Угловые скобки \(<>\) vs. Двойные кавычки \(«»\)](#)
4. [Почему iostream пишется без окончания .h?](#)
5. [Можно ли записывать определения в заголовочных файлах?](#)
6. [Советы](#)

Заголовочные файлы из Стандартной библиотеки C++

Рассмотрим следующую программу:

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

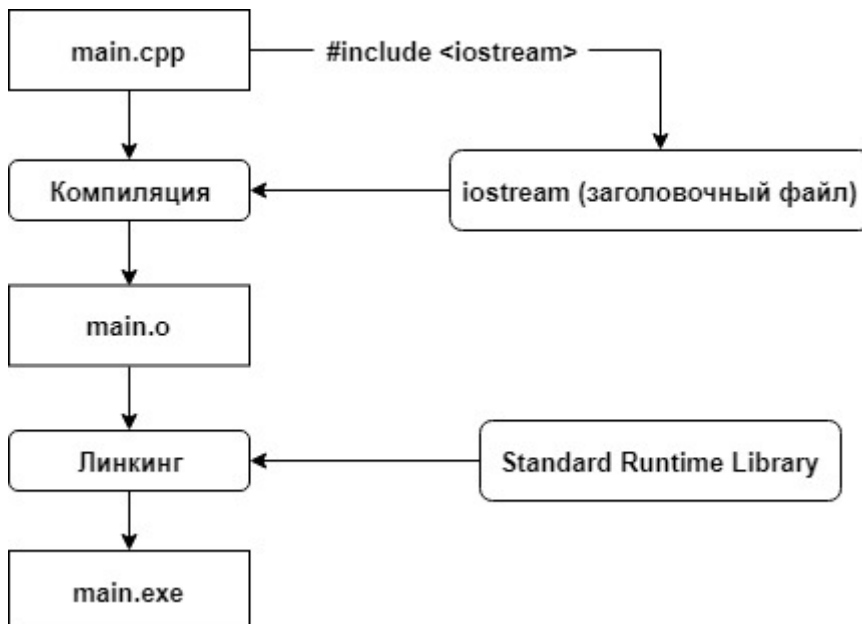
```
3 int main()
4 {
5     std::cout << "Hello, world!" << std::endl;
6     return 0;
7 }
```

Результат выполнения программы:

Hello, world!

В этой программе мы используем `cout`, который нигде не определяем. Откуда компилятор знает, что это такое? Дело в том, что `cout` объявлен в заголовочном файле `iostream`. Когда мы пишем `#include <iostream>`, мы делаем запрос, чтобы всё содержимое заголовочного файла `iostream` было скопировано в наш файл. Таким образом, всё содержимое библиотеки `iostream` становится доступным для использования.

Как правило, в заголовочных файлах записываются только объявления, без определений. Следовательно, если `cout` только объявлен в заголовочном файле `iostream`, то где же он определяется? Ответ: в Стандартной библиотеке C++, которая автоматически подключается к вашему проекту на этапе линкинга.



Подумайте о последствиях отсутствия заголовочного файла `iostream`. Каждый раз, при использовании `cout`, вам бы приходилось вручную копировать все предварительные объявления, связанные с `cout` в верхнюю часть вашего файла! Хорошо ведь, что можно просто указать `#include <iostream>`, не так ли?

Пишем свои собственные заголовочные файлы

Теперь давайте вернемся к примеру, который мы обсуждали на предыдущем уроке. У нас было два файла: `add.cpp` и `main.cpp`.

`add.cpp`:

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

```
2 | {  
3 |     return x + y;  
4 | }
```

main.cpp:

```
1 | #include <iostream>  
2 |  
3 | int add(int x, int y); // предварительное объявление с использованием прототипа функции  
4 |  
5 | int main()  
6 | {  
7 |     std::cout << "The sum of 3 and 4 is " << add(3, 4) << std::endl;  
8 |     return 0;  
9 | }
```

Примечание: Если вы создаете все файлы заново, то не забудьте добавить add.cpp в свой проект, чтобы он был подключен к компиляции.

Мы использовали предварительное объявление, чтобы сообщить компилятору, что такое add(). Как мы уже говорили, записывать в каждом файле предварительные объявления используемых функций — дело не слишком увлекательное.

И здесь нам на помощь приходят заголовочные файлы. Достаточно просто написать один заголовочный файл и его можно будет повторно использовать в любом количестве программ. Также и вносить изменения в такой код (например, добавление еще одного параметра) гораздо легче, нежели чем шерстить по всем файлам в поисках используемых функций.

Написать свой собственный заголовочный файл не так уж и сложно. Заголовочные файлы состоят из двух частей:

- **Директивы препроцессора** — в частности, header guards, которые предотвращают вызов заголовочного файла больше одного раза из одного и того же файла (об этом детально на следующем уроке).
- **Содержимое заголовочного файла** — набор объявлений.

Все ваши заголовочные файлы (которые вы написали самостоятельно) должны иметь расширение `.h`.

add.h:

```
1 | // Начнем с директив препроцессора. ADD_H — это произвольное уникальное имя (обычно  
2 | #ifndef ADD_H  
3 | #define ADD_H  
4 |  
5 | // А это уже содержимое заголовочного файла  
6 | int add(int x, int y); // прототип функции add() (не забывайте точку с запятой в конце)  
7 |  
8 | // Заканчиваем директивой препроцессора  
9 | #endif
```

Чтобы использовать этот файл в main.cpp, вам сначала нужно будет подключить его к проекту.

main.cpp, в котором мы подключаем add.h:

```
1 #include <iostream>
2 #include "add.h"
3
4 int main()
5 {
6     std::cout << "The sum of 3 and 4 is " << add(3, 4) << std::endl;
7     return 0;
8 }
```

add.cpp остается без изменений:

```
1 int add(int x, int y)
2 {
3     return x + y;
4 }
```

Когда компилятор встречает `#include "add.h"`, он копирует всё содержимое `add.h` в текущий файл. Таким образом, мы получаем предварительное объявление функции `add()`.

Примечание: При подключении заголовочного файла, всё его содержимое вставляется сразу же после строки `#include`

Если вы получили ошибку от компилятора, что `add.h` не найден, то убедитесь, что имя вашего файла точно «`add.h`». Вполне возможно, что вы могли сделать опечатку, например, просто «`add`» (без «`.h`») или «`add.h.txt`» или «`add.hpp`».

Если вы получили ошибку от линкера, что функция `add()` не определена, то убедитесь, что вы корректно подключили `add.cpp` к вашему проекту (и к компиляции тоже)!

Угловые скобки (<>) vs. Двойные кавычки («»)

Вы, наверное, хотите узнать, почему используются угловые скобки для `iostream` и двойные кавычки для `add.h`. Дело в том, что, используя угловые скобки, мы сообщаем компилятору, что подключаемый заголовочный файл написан не нами (он является «системным», т.е. предоставляется Стандартной библиотекой C++), так что искать этот заголовочный файл следует в системных директориях. Двойные кавычки сообщают компилятору, что мы подключаем наш собственный заголовочный файл, который мы написали самостоятельно, поэтому искать его следует в текущей директории нашего проекта. Если файла там не окажется, то компилятор начнет проверять другие пути, в том числе и системные директории.

Правило: Используйте угловые скобки для подключения «системных» заголовочных файлов и двойные кавычки для ваших заголовочных файлов.

Стоит отметить, что одни заголовочные файлы могут подключать другие заголовочные файлы. Тем не менее, так делать не рекомендуется.

Почему `iostream` пишется без окончания `.h`?

Еще один часто задаваемый вопрос: «Почему `iostream` (или любой другой из стандартных заголовочных файлов) при подключении пишется без окончания `«.h»?`». Дело в том, что есть 2 отдельных файла: `iostream.h` (заголовочный файл) и просто `iostream!` Для объяснения потребуется краткий экскурс в историю.

Когда C++ только создавался, все файлы библиотеки Runtime имели окончание `.h`. Оригинальные версии `cout` и `cin` объявлены в `iostream.h`. При стандартизации языка C++ комитетом ANSI, решили перенести все функции из библиотеки Runtime в пространство имен `std`, чтобы предотвратить возможность возникновения конфликтов имен с пользовательскими идентификаторами (что, между прочим, является хорошей идеей). Тем не менее, возникла проблема: если все функции переместить в пространство имен `std`, то старые программы переставали работать!

Для обеспечения обратной совместимости ввели новый набор заголовочных файлов с теми же именами, но без окончания `«.h»`. Весь их функционал находится в пространстве имен `std`. Таким образом, старые программы с `#include <iostream.h>` не нужно было переписывать, а новые программы уже могли использовать `#include <iostream>`.

Когда вы подключаете заголовочный файл из Стандартной библиотеки C++, убедитесь, что вы используете версию без `.h` (если она существует). В противном случае, вы будете использовать устаревшую версию заголовочного файла, который уже больше не поддерживается.

Кроме того, многие библиотеки, унаследованные от языка Си, которые до сих пор используются в C++, также были продублированы с добавлением префикса `c` (например, `stdlib.h` стал `cstdlib`). Функционал этих библиотек также перенесли в пространство имен `std`, чтобы избежать возможность возникновения конфликтов имен с пользовательскими идентификаторами.

Правило: При подключении заголовочных файлов из Стандартной библиотеки C++, используйте версию без `«.h»` (если она существует). Пользовательские заголовочные файлы должны иметь окончание `«.h»`.

Можно ли записывать определения в заголовочных файлах?

C++ не будет жаловаться, если вы это сделаете, но так делать не принято.

Как уже было сказано выше, при подключении заголовочного файла, всё его содержимое вставляется сразу же после строки с `#include`. Это означает, что любые определения, которые есть в заголовочном файле, скопируются в ваш файл.

Для небольших проектов, это, скорее всего, не будет проблемой. Но для более крупных это может способствовать увеличению времени компиляции (так как код будет повторно компилироваться) и размеру исполняемого файла. Если внести изменения в определения, которые находятся в файле `.cpp`, то перекомпилировать придется только этот файл. Если же внести изменения в определения, которые записаны в заголовочном файле, то перекомпилировать придется каждый файл, который подключает этот заголовок, используя директиву препроцессора `#include`. И вероятность того, что из-за одного небольшого изменения вам придется перекомпилировать весь проект, резко возрастает!

Иногда делаются исключения для простых функций, которые вряд ли изменятся (например, где определение состоит всего лишь из одной строки).

Советы

Вот несколько советов по написанию собственных заголовочных файлов:

- Всегда используйте директивы препроцессора.
- Не определяйте переменные в заголовочных файлах, если это не константы. Заголовочные файлы следует использовать только для объявлений.
- Не определяйте функции в заголовочных файлах.
- Каждый заголовочный файл должен выполнять свое конкретное задание и быть как можно более независимым. Например, вы можете поместить все ваши объявления, связанные с файлом A.cpp в файл A.h, а все ваши объявления, связанные с B.cpp — в файл B.h. Таким образом, если вы будете работать только с A.cpp, то вам будет достаточно подключить только A.h и наоборот.
- Используйте имена ваших рабочих файлов в качестве имен для ваших заголовочных файлов (например, grades.h работает с grades.cpp).
- Не подключайте одни заголовочные файлы из других заголовочных файлов.
- Не подключайте файлы .cpp, используя директиву препроцессора #include.

Оценить статью:

★★★★★ (705 оценок, среднее: 4,87 из 5)



← [Урок №20. Многофайловые программы](#)

[Урок №22. Директивы препроцессора](#)



Комментариев: 58



1. Сергей:

[18 мая 2020 в 15:34](#)

Добрый день, тут сказано в заголовочные файлы не подключать другие заголовочные файлы, а что если например функция имеет параметр: string, как мы укажем ее в заголовочном файле не подключив #include <string> ?

[Ответить](#)

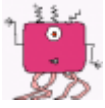


1. Малик:

[10 июня 2020 в 19:28](#)

Поддерживаю вопрос. Если, например, вместо класса std::string, будет использоваться класс std::array.

[Ответить](#)



2. Андрей:

[19 января 2020 в 19:12](#)

Заработало только после того как положил сохраненный `add.h` в папку с проектом и законнектил его к проекту из директории проекта... не знаю или профукал этот нюанс или же о нем не было сказано.

Просто решил создать отдельный файл от проекта, сделал для таких файлов отдельную директорию, т.к. по логике изложения говорится что эти заголовочные файлы можно использовать для разных проектов... а получается только если скопируешь физически из в директорию проекта. Странно реализовано или я что-то неверно делаю?

[Ответить](#)



1. Торвольт:

[30 сентября 2020 в 12:59](#)

Поиск в стандартной и текущей директории автоматический...
Для любой другой директории нужно указывать путь !!!)))

[Ответить](#)



3. Владимир:

[14 ноября 2019 в 00:23](#)

Или здесь имеются в виду случаи когда "Иногда делаются исключения для простых функций, которые вряд ли изменятся (например, где определение состоит всего лишь из одной строки)." Просто очень уж путает и кашу в голове делает)))
Еще раз благодарю за Вашу работу.

[Ответить](#)



1. Юрий:

[14 ноября 2019 в 23:15](#)

Исключения для простых функций (которые состоят из меньше чем 5 строчек кода) действительно есть. Но правильнее определять функции в `.cpp` файлах.

[Ответить](#)



4. Владимир:

[14 ноября 2019 в 00:10](#)

В первую очередь большая Вам, Юрий, благодарность за такую отличную и полезную работу! И вопрос. В этом уроке случайно не перепутаны кое-где понятия "объявление" и "определение"? Вот, к примеру цитата "...Оригинальные версии `cout` и `cin` определялись в `iostream.h`...". Но, вместе с этим, мы же учим, что заголовочные (`.h`) файлы служат именно для объявления, а определение уже идет в других файлах системной директории? Буду благодарен, если поможет разобраться с этим моментом.

[Ответить](#)

1. Юрий:

[14 ноября 2019 в 23:10](#)

Всё верно подметили, действительно — это моя опечатка, уже исправил. Спасибо за замечание))

[Ответить](#)

5. Абакар:

[28 октября 2019 в 23:08](#)

До 20 урока очень простые уроки были) 20-й урок чуть сложноват оказался т.к не понятно было, как это сделать по словам, но благодаря гуглу разобрался. А вот 21 урок значительно сложным оказался(((Но благо нашёл подходящий ролик и смог это реализовать. Спасибо) По вам занимаюсь) Очень доступно и оформление на высшем уровне

[Ответить](#)

1. Юрий:

[29 октября 2019 в 19:21](#)

Согласен, что иногда может быть трудно, но вы всё верно сделали: если что-то осталось непонятным — Гугл с Ютубом в помощь 😊

[Ответить](#)

6. Алексей:

[19 сентября 2019 в 20:27](#)

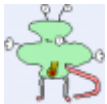
Пытался вынести одну единственную функцию в отдельный файл, потому что везде ее использую, но даже после подключения к проекту и после всех моих попыток ничего не выходит (создал заголовочный файл, который объявляет мою функцию), но VS выдает мне кучу синтаксических ошибок, которых в помине нет...

[Ответить](#)

7. ProstoProgrammist:

[27 августа 2019 в 19:23](#)

C:\Users\\collect2.exe [Error] ld returned 1 exit status — пишет что делать??

[Ответить](#)

8. Юлия:

[24 августа 2019 в 11:32](#)

Почему "Не подключайте одни заголовочные файлы из других заголовочных файлов"? По-моему, очень удобно все .h включить в одном общем заголовочном файле, например, header.h. Тогда в

каждом заголовочном включаем только один общий заголовочный `#include "header.h"`. Придется меньше думать 😊

[Ответить](#)



9. Алексей:

[11 июля 2019 в 16:40](#)

Доброго времени суток! Столкнулся со следующей проблемой: в точности скопировал ответ задания №3 в Code::Blocks, но возникают ошибки: `"readNumber" was not declared in this scope` и `"writeAnswer" was not declared in this scope`. Можете помочь?

[Ответить](#)



10. Роман:

[16 июня 2019 в 13:20](#)

Почему не `#pragma once`?

[Ответить](#)



1. Юлия:

[24 августа 2019 в 11:22](#)

Да, присоединяюсь. Очень интересует этот вопрос. По умолчанию в VS всегда появляется `#pragma once`

[Ответить](#)



2. Юлия:

[24 августа 2019 в 18:43](#)

Снимаю вопрос — объяснение в уроке №23

[Ответить](#)



11. Андрей:

[30 мая 2019 в 20:04](#)

Здравствуйте, столкнулся с такой проблемой, при запуске программы не удается найти EXE файл.

[Ответить](#)



1. Никита:

[2 июля 2019 в 21:59](#)

Забыли подключить `#include "pch.h"` в новом созданном .cpp файле.

[Ответить](#)

12. *ALEXA:*[30 мая 2019 в 19:18](#)

Все сделала, как описано в уроке, но стабильно получаю ошибку линкера о том, что функция не определена. Все перепроверила. Что еще посоветуете?

[Ответить](#)13. *Александр:*[17 апреля 2019 в 04:08](#)

Разобраться с темой не так просто как кажется на первый взгляд.
Спасибо Юрий за понимание и терпение.

[Ответить](#)14. *Анна:*[1 февраля 2019 в 16:20](#)

В Dev C++ создала проект как конс. приложение, поместила в разные файлы .cpp разные функции, в заголовочном файле все прототипы, в мэйне все вызовы работали. В параметрах проекта все файлы были видны. НО! ПОСЛЕ СОХРАНЕНИЯ ПРОЕКТА (сохранить проект как...) и ЗАКРЫТИЯ IDE после повторного открытия в этом проекте нет ни одного файла. Если открыть код файла проекта то там ничего не прописано, ниже фрагмент:

```
"...[Project]
FileName=pr1
Name=PR1
Type=1
Ver=2
ObjFiles=
Includes=
Libs=
PrivateResource=
ResourceIncludes=
MakeIncludes=
Compiler=
CppCompiler=
Linker=
IsCpp=1..."
```

Где-то что не так установлено? Или сохранять пути к файлам где- надо отдельно? Подскажите пожалуйста! Заранее спасибо!

[Ответить](#)15. *Анатолий:*[28 января 2019 в 00:10](#)

Спасибо за сайт! Спасибо за помощь в комментариях

[Ответить](#)

1. *Юрий:*
[28 января 2019 в 13:08](#)

Помочь всем в комментариях не всегда получается, но пожалуйста 😊

[Ответить](#)

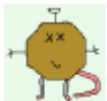
16. *Владимир:*
[4 января 2019 в 02:03](#)

Огромное спасибо! Совершенно не мог разобраться с многофайловыми проектами сам и не видел ни одного толкового обучения. Ваша помощь в обучении просто неоценима!)

[Ответить](#)

1. *Юрий:*
[4 января 2019 в 03:31](#)

Спасибо, что читаете 😊

[Ответить](#)

17. *Алексей:*
[16 ноября 2018 в 21:13](#)

Как сохранить файл .exe, когда программа уже готова?

[Ответить](#)

1. *Андрей:*
[3 декабря 2018 в 09:15](#)

Не скажу как в VS и code::blocks, но если просто через командную строку, то как-то так:

```
1 | g++ main.cpp add.cpp -o main.exe
```

Заголовочный файл (add.h в данном случае), если он лежит с ними же в папке, подхватится автоматом. Если он в другой папке — выше в уроке есть для этого команда.

Только что попробовал, все прекрасно работает, exe-шник сохраняется, запускается.

P.S.

Может быть есть более "правильные" варианты этого дела, но на данном этапе этого вполне достаточно.

[Ответить](#)

2. *Никита:*

[2 июля 2019 в 22:06](#)

Сверху под меню "Отладка", если VS выбираешь relese и разрядность 86 или 64. После где создал проект в нем будет папка "relese" и там будет собранный .exe проект ваш.

[Ответить](#)



18. *Александр:*

[3 ноября 2018 в 19:49](#)

А почему в файле add.h не объявляется `#include "add.cpp"`? Без этого не компилируется программа.

[Ответить](#)



1. *Алексей:*

[21 июня 2019 в 15:46](#)

Сам в убунту это делаю, и реально без указания — ругань.

[Ответить](#)



19. *Ramazan:*

[24 сентября 2018 в 12:54](#)

Моя задача чтобы были одинаковые цифры. Но почему-то не получается совместить. Почему? очень прошу подсказать

//main.cpp

```
1 #include <iostream>
2 #include "raz.h"
3 using namespace std;
4
5 int main()
6 {
7     prim1();
8
9 }
```

//raz.cpp

```
1 #include<iostream>
2 #include<math.h>
3
4 using namespace std;
5 void prim1(){
6     cout<<"Tel151a"<<endl;
7     cout<<"Vvedite a, b"<<endl;
8     double r1, r2, r, F1, F2, a, b;
9     cin>>a>>b;
10    r1=(a/(b*b-a*b));
```

```

11     r2=(b/(a*a-a*b));
12     r=(r1+r2);
13     r1=(a*b)/(b-a);
14     F1=r*r1;
15     F2=(a+b)/(a-b);
16     cout<<"F1="<<F1<<" F2="<<F2<<endl;
17 }

```

//raz.h

```

1  #ifndef VR10LAB1_H_INCLUDED
2  #define VR10LAB1_H_INCLUDED
3
4  void prim1();
5
6  #endif

```

[Ответить](#)



1. *Koregazawarudo:*
[13 октября 2018 в 23:14](#)

А где берёшь задачи ?

[Ответить](#)



20. *Евген:*
[8 августа 2018 в 15:01](#)

вопрос:

попытался создать файл заголовков:

```

1  // Здесь начинается header guard. ADD_H – любое уникальное имя. Обычно используе
2  #ifndef ZF
3  #define ZF
4
5  // А это уже контент файла .h, где и находятся все предварительные объявления
6  int add(int x, int y); // прототип функции add.h (не забывайте точку с запятой!)
7
8                                     // Конец header guard
9  #endif

```

и сам `main` выглядит так:

```

1  #include "stdafx.h"
2
3
4  #include <iostream>
5
6  #include "ZF.h"
7

```

```

8 | int main()
9 | {
10 |     using namespace std;
11 |     cout << "The sum of 3 and 4 is: " << add(3, 4) << endl;
12 |     return 0;
13 | }

```

пишет, что не удалось открыть источник файла ZF.h
что я не понял?

[Ответить](#)



1. *Евген:*

[9 августа 2018 в 13:38](#)

а, разобрался с вопросом выше, но возник новый: какую бы я информацию не вносил в поле

```

1 | #ifndef zf_h
2 | #define zf_h

```

а именно

ADD_H – любое уникальное имя. Обычно используют имя заголовочного файла в качестве этого идентификатора — сюда, прога все равно компилируется и работает. я так понял, что неважно что пишешь на месте ADD_H? оно будет работать в любом случае?

[Ответить](#)



21. *Антонина:*

[22 июня 2018 в 14:36](#)

Не поняла, когда к main.cpp подключается реализация add.cpp. Ясно, что add.h подключен, но там только объявление. Спасибо.

[Ответить](#)



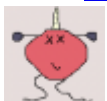
1. *Юрий:*

[23 июня 2018 в 14:33](#)

Вместо строки add.h вставляется содержимое этого файла, т.е. предварительное объявление функции add, реализация которой находится в файле add.cpp. Поскольку эти файлы находятся в одном проекте, то мы можем спокойно, используя предварительное объявление функции, вызывать функции из других файлов. Если бы мы add.cpp добавили в другой проект и так же подключали, то ничего бы не вышло.

Суть в том, что если файлы находятся в одном проекте, то вызов функций с разных файлов возможен при использовании предварительных объявлений требуемых функций.

[Ответить](#)



22. *Илья:*

[15 июня 2018 в 14:32](#)

для написания программ требуется мощный ПК?
у меня процессор amd a9 2 ядра
2 гига оперативной
достаточно?

[Ответить](#)

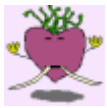


1. Юрий:

[15 июня 2018 в 18:27](#)

Для прохождения этих уроков мощности вашего ПК будет вполне достаточно.

[Ответить](#)



23. artem:

[26 мая 2018 в 19:53](#)

Если я пройду этот туториал меня возьмут в google)?

[Ответить](#)



1. Юрий:

[27 мая 2018 в 12:05](#)

Если пройдете — создадите сами свой Google.

[Ответить](#)



24. painkiller:

[16 мая 2018 в 18:11](#)

Добрый день!

Меня вот какие вопросы интересуют:

1. Для создания заголовочного файла прописывать эти строки обязательно, верно?

```
1 #ifndef ADD_H
2 #define ADD_H
3 -----
4 #endif
```

Пробовал создать заголовочный файл и без этих строк — работает.

2. Обязательно ли использовать верхний регистр для обозначения имени (в данном случае ADD_H)? Пробовал в нижнем регистре — компилятор молчит.

3. Если включить два прототипа функций в заголовочный файл, то выполняется в итоге только первая функция. Когда каждый прототип поместил между строк

```
1 #ifndef ADD_H
```

```
2 #define ADD_H
3
4 int add(int x, int y);
5
6 #endif
7
8 #ifndef ADD_H
9 #define ADD_H
10
11 int subtract(int x, int y);
12
13 #endif
```

то программа заработала и все было правильно подсчитано. Так и должно быть?

Спасибо!

[Ответить](#)



1. *painkiller:*
[16 мая 2018 в 18:21](#)

Уточнение к вопросу №3:

Да, не подключая header guard, заголовочные файлы тоже работают, в каком бы порядке я не вписывал туда прототипы функций. Видимо, я в прошлый раз не скомпилировал код



[Ответить](#)



2. *Юрий:*
[18 мая 2018 в 23:06](#)

Привет.

1. О header guards есть отдельный урок, где рассказывается зачем они нужны и как их использовать. Их можно и не использовать — дело ваше, работать будет и без них. А качество такой работы — уже отдельная тема.

2. Не обязательно. Верхний регистр используется, как уже устоявшийся и в целях выделения названий заголовочных файлов от названий других файлов.

[Ответить](#)



25. *Константин:*
[14 мая 2018 в 18:17](#)

В каких ситуациях необходимо использовать Header guard, точнее при каких обстоятельствах заголовочный файл может быть подключен более 1 раза?

[Ответить](#)



1. Юрий:

[16 мая 2018 в 20:37](#)

В каких ситуациях и для чего используются header guards — есть отдельный урок 23.

При каких обстоятельствах заголовочный файл может быть подключен более 1 раза? Заголовочный файл содержит вызовы других файлов/библиотек/заголовочных файлов, объявления функций/классов и прочего. Если вы уже подключили заголовочный файл с содержимым, то зачем подключать его еще раз — чего вы этим добьетесь, кроме дублирования кода и ошибки компилятора? Какой смысл может быть в подключении одного и того же файла 2 раза? (если я правильно понял суть вашего вопроса)

[Ответить](#)



26. Максим():

[8 мая 2018 в 17:37](#)

У меня такой интересный вопрос: если определение базовых команд реализуется в C++ Runtime Support Library, можно ли создать самому что-то подобное для своих функций?

[Ответить](#)

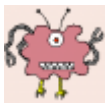


1. Юрий:

[10 мая 2018 в 18:00](#)

Утверждать ничего не могу, с этим вопросом детально не знаком. Но Runtime Library поставляется вместе с компилятором и вашей IDE, для ваших функций и так уже используется этот встроенный Runtime Library, т.е. переписывать уже существующую библиотеку в Visual Studio — смысла не вижу. Если же вы хотите написать свою среду разработки, то да, можно переписать эту библиотеку. Но детальнее как это делается и более углубленно рассказать об этом вопросе я не могу, так как сам не знаю.

[Ответить](#)



27. Виталий:

[10 апреля 2018 в 03:22](#)

"Не #include файлы .cpp" — это на каком языке? Вся статья на таком сленге. Ну неудобно же читать, тем более тем, кто не мыслит шаблонами C++.

Имхо.

[Ответить](#)



1. Юрий:

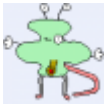
[10 апреля 2018 в 16:18](#)

В статье ведь указывается, что такое include несколько раз: "Примечание: когда вы подключаете (#include) файл, всё его содержимое вставляется сразу после строчки include в текущий файл."

Исходя из многократных повторений, чтобы материал лучше усвоился — в статье и используется этот "сленг".

А насчет неудобно читать, то это вам ведь не художественная литература. Учить язык программирования — штука вообще неудобная.

[Ответить](#)



2. *Юлия:*

[24 августа 2019 в 11:15](#)

Хочешь стать программистом — мысли шаблонами C++

[Ответить](#)



28. *Aleksandr:*

[16 февраля 2018 в 10:51](#)

1. Для (например) трех функций `f1`, `f2`, `f3` созданы три файла `f1.cpp`, `f2.cpp`, `f3.cpp`.
 2. В общем для них (отдельные для каждой заголовочные не создавались) заголовочном `f.h` прописаны объявления `: f1(); f2(); f3();`.
 3. В файле `stdafx` дополнительно прописано `#include "f.h"`.
 4. `#include "stdafx"` в самом начале — есть, `#include "f.h"` — не прописываем.
- Задача — упростить код(ихмо). При запуске небольшого кода — работает.
А можно так делать ?

[Ответить](#)



1. *Юрий:*

[17 февраля 2018 в 11:56](#)

Делать можно, но не во всех случаях. Есть два нюанса.

Сам файл `stdafx.h` является просто объединением нескольких уже подключенных библиотек, чтобы каждый раз в коде не прописывать вверху десятки `#include`, а просто вызвать один `stdafx.h`. Т.е. он служит для упрощения кода (относительно), поэтому включать свои заголовки в этот файл можно. Но эти заголовки, которые вы подключаете, должны очень редко изменяться и действительно использоваться в вашем коде.

Первый нюанс. Если у вас десятки файлов, в каждом подключен `stdafx.h`, в котором находится подключенный ваш собственный заголовок, который используется только в 2 файлов из 30, то смысла его засовывать в `stdafx.h` — нет. Если же почти все ваши файлы с кодом будут использовать этот заголовок, то в качестве упрощения кода можно внести этот заголовок в `stdafx.h`.

Второй нюанс. Код в вашем подключенном заголовке должен очень редко меняться, а лучше, чтобы вообще не менялся. Так как, если у вас десятки файлов с кодом, в каждом подключен `stdafx.h`, в который вы подключили свой заголовок и после этого вам потребовалось изменить что-либо в подключенном вами заголовке, то вам придется перекомпилировать весь проект (десятки файлов) + может быть высока вероятность возникновения конфликтов имен и других ошибок.

[Ответить](#)1. *Aleksandr:*[17 февраля 2018 в 12:11](#)

Спасибо за развёрнутый ответ и еще раз за ваш сайт.

[Ответить](#)2. *Александр:*[17 февраля 2018 в 12:21](#)

Ребят не по теме но все же, подскажите хорошую литературу по c++ в бумажном варианте, перед компьютером неудобно постоянно находиться, а с книжкой было бы сподручнее. Я в книжный зашел, а там сотня разнообразных книг, боюсь что возьму не очень полезную и сломаю себе голову (уже имел опыт в других сферах с некачественной литературой, из за которой больше проблем чем пользы), подскажите автора.
Спасибо.

[Ответить](#)1. *Юрий:*[17 февраля 2018 в 13:15](#)

Лично я не могу порекомендовать вам какую-либо литературу по C++, так как сам ничего не читал. Но вот постоянно сталкиваюсь с такими книгами на разных порталах о программировании и C++:

1. **Бьёрн Страуструп** (создатель C++) — **«Программирование. Принципы и практика использования C++»**.
2. **Герберт Шилдт** (у него есть целые циклы книг о C++ и других языках программирования) — **«C++: базовый курс»**.
3. **Стивен Прата** — **«Язык программирования C++»**.

Примечание: Их книги лучше читать в оригинале, так как перевод может быть не всегда корректным, но для этого нужно знать английский не на базовом уровне.

Поищите в Интернете электронные версии этих книг, окиньте их все быстрым взглядом, выберите одну, какая подойдет вам лучше (все сразу не нужно — информация у них в большей степени будет дублироваться) и тогда уже, если захотите, купите именно эту книгу в магазине.

29. *Александр:*[15 февраля 2018 в 18:07](#)

Долго мучился с заголовочным файлом (часа 2 точно, а все из за невнимательности на скобки), но в итоге получилось, спасибо Вам ребят за доходчивое объяснение.

[Ответить](#)

1. Юрий:

[15 февраля 2018 в 19:34](#)

Пожалуйста 😊

[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