

Ravesli [Ravesli](#)


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


Урок №22. Директивы препроцессора

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 |

 68833

[1](#)  37

Препроцессор лучше всего рассматривать как отдельную программу, которая выполняется перед компиляцией. При запуске программы, препроцессор просматривает код сверху вниз, файл за файлом, в поиске директив. **Директивы** — это специальные команды, которые начинаются с символа # и НЕ заканчиваются точкой с запятой. Есть несколько типов директив, которые мы рассмотрим ниже.

Оглавление:

1. [Директива #include](#)
2. [Директива #define](#)
3. [Макросы-объекты с текст_замена](#)
4. [Макросы-объекты без текст_замена](#)
5. [Условная компиляция](#)
6. [Область видимости директивы #define](#)

Директива #include

Вы уже видели директиву #include в действии. Когда вы подключаете файл с помощью директивы #include, препроцессор копирует содержимое подключаемого файла в текущий файл сразу после строки с #include. Это очень полезно при использовании определенных данных (например, [предварительных объявлений](#) функций) сразу в нескольких местах.

Директива #include имеет две формы:

➔ **#include <filename>**, которая сообщает препроцессору искать файл в системных путях (в местах хранения системных библиотек языка C++). Чаще всего вы будете использовать эту форму при подключении [заголовочных файлов](#) из Стандартной библиотеки C++.

→ **#include "filename"**, которая сообщает препроцессору искать файл в текущей директории проекта. Если его там не окажется, то препроцессор начнет проверять системные пути и любые другие, которые вы указали в настройках вашей **IDE**. Эта форма используется для подключения пользовательских заголовочных файлов.

Директива #define

Директиву **#define** можно использовать для создания макросов. **Макрос** — это правило, которое определяет конвертацию идентификатора в указанные данные.

Есть два основных типа макросов: макросы-функции и макросы-объекты.

Макросы-функции ведут себя как функции и используются в тех же целях. Мы не будем сейчас их обсуждать, так как их использование, как правило, считается опасным, и почти всё, что они могут сделать, можно осуществить с помощью простой (линейной) функции.

Макросы-объекты можно определить одним из следующих двух способов:

#define идентификатор

Или:

#define идентификатор текст_замена

Верхнее определение не имеет никакого **текст_замена**, в то время как нижнее — имеет. Поскольку это директивы препроцессора (а не простые **стейтменты**), то ни одна из форм не заканчивается точкой с запятой.

Макросы-объекты с текст_замена

Когда препроцессор встречает макросы-объекты с **текст_замена**, то любое дальнейшее появление идентификатор заменяется на **текст_замена**. идентификатор обычно пишется заглавными буквами с символами подчёркивания вместо пробелов.

Рассмотрим следующий фрагмент кода:

```
1 #define MY_FAVORITE_NUMBER 9
2
3 std::cout << "My favorite number is: " << MY_FAVORITE_NUMBER << std::endl;
```

Препроцессор преобразует вышеприведенный код в:

```
1 std::cout << "My favorite number is: " << 9 << std::endl;
```

Результат выполнения:

My favorite number is: 9

Мы обсудим это детально, и почему так не стоит делать, на следующих уроках.

Макросы-объекты без текст_замена

Макросы-объекты также могут быть определены без `текст_замена`, например:

```
1 #define USE_YEN
```

Любое дальнейшее появление идентификатора `USE_YEN` удаляется и заменяется «ничем» (пустым местом)!

Это может показаться довольно бесполезным, однако, это не основное предназначение подобных директив. В отличие от макросов-объектов с `текст_замена`, эта форма макросов считается приемлемой для использования.

Условная компиляция

Директивы препроцессора условной компиляции позволяют определить, при каких условиях код будет компилироваться, а при каких — нет. На этом уроке мы рассмотрим только **три директивы условной компиляции**:

- `#ifdef`
- `#ifndef`
- `#endif`

Директива `#ifdef` (сокр. от «*if defined*» = «если определено») позволяет препроцессору проверить, было ли значение ранее определено с помощью директивы `#define`. Если да, то код между `#ifdef` и `#endif` скомпилируется. Если нет, то код будет проигнорирован. Например:

```
1 #define PRINT_JOE
2
3 #ifdef PRINT_JOE
4     std::cout << "Joe" << std::endl;
5 #endif
6
7 #ifdef PRINT_BOB
8     std::cout << "Bob" << std::endl;
9 #endif
```

Поскольку `PRINT_JOE` уже был определен, то строка `std::cout << "Joe" << std::endl;` скомпилируется и выполнится. А поскольку `PRINT_BOB` не был определен, то строка `std::cout << "Bob" << std::endl;` не скомпилируется и, следовательно, не выполнится.

Директива `#ifndef` (сокр. от «*if not defined*» = «если не определено») — это полная противоположность к `#ifdef`, которая позволяет проверить, не было ли значение ранее определено. Например:

```
1 #ifndef PRINT_BOB
2     std::cout << "Bob" << std::endl;
3 #endif
```

Результатом выполнения этого фрагмента кода будет `Bob`, так как `PRINT_BOB` ранее никогда не был определен. Условная компиляция очень часто используется в качестве `header guards` (о них мы поговорим на следующем уроке).

Область видимости директивы #define

Директивы выполняются перед компиляцией программы: сверху вниз, файл за файлом. Рассмотрим следующую программу:

```
1 #include <iostream>
2
3 void boo()
4 {
5     #define MY_NAME "Alex"
6 }
7
8 int main()
9 {
10     std::cout << "My name is: " << MY_NAME;
11
12     return 0;
13 }
```

Несмотря на то, что директива `#define MY_NAME "Alex"` определена внутри функции `boo()`, преппроцессор этого не заметит, так как он не понимает такие понятия языка C++, как функции. Следовательно, выполнение этой программы будет идентично той, в которой бы `#define MY_NAME "Alex"` было определено ДО, либо сразу ПОСЛЕ функции `boo()`. Для лучше читабельности кода определяйте идентификаторы (с помощью `#define`) вне функций.

После того, как преппроцессор завершит свое выполнение, все идентификаторы (определенные с помощью `#define`) из этого файла — отбрасываются. Это означает, что директивы действительны только с точки определения и до конца файла, в котором они определены. Директивы, определенные в одном файле кода, не влияют на директивы, определенные внутри других файлов этого же проекта.

Рассмотрим следующий пример:

function.cpp:

```
1 #include <iostream>
2
3 void doSomething()
4 {
5     #ifdef PRINT
6         std::cout << "Printing!";
7     #endif
8     #ifndef PRINT
9         std::cout << "Not printing!";
10    #endif
11 }
```

main.cpp:

```
1 void doSomething(); // предварительное объявление функции doSomething()
2
3 int main()
```

```
4 {  
5 #define PRINT  
6  
7     doSomething();  
8  
9     return 0;  
10 }
```

Результат выполнения программы:

Not printing!

Несмотря на то, что мы объявили PRINT в main.cpp (#define PRINT), это все равно не имеет никакого влияния на что-либо в function.cpp. Поэтому, при выполнении функции doSomething(), у нас выводится Not printing!, так как в файле function.cpp мы не объявляли идентификатор PRINT (с помощью директивы #define). Это связано с [header guards](#).

Оценить статью:

★★★★★ (546 оценок, среднее: 4,71 из 5)



← [Урок №21. Заголовочные файлы](#)

[Урок №23. Header guards и #pragma once](#) →



Комментариев: 37



1. *Салех:*

[13 мая 2020 в 12:14](#)

Спасибо огромное за ваш труд!

Как я понял шаги компиляции (согласно нашим знаниям, полученным до этого урока) следующие:

- 1) препроцессор прочесывает весь код в поисках директив: вставляет все объявления, определяет все макросы;
- 2) далее прочесываются другие команды;

Тогда вопрос: Вот в такой программе

```
1 int main(){  
2     std::cout << MY_NUMBER;  
3     #define MY_NUMBER 8  
4 }
```

происходит ошибка компиляции. Получается, что хоть препроцессор и прочесал код в поисках директив, мой компилятор не знает о существовании MY_NUMBER.

Мне искать дальше или я что-то пропустил из предыдущих уроков?

[Ответить](#)



1. *Руслан:*

[16 мая 2020 в 14:34](#)

Нужно MY_NUMBER объявить до её вызова.

[Ответить](#)



1. *Салех:*

[16 мая 2020 в 16:10](#)

Получается, что да. Думал, что сначала препроцессор проверяет все директивы и делает себе что-то вроде словаря. Оказывается не совсем

[Ответить](#)



2. *AndreyOlegovich.ru:*

[6 апреля 2020 в 15:56](#)

Спасибо за эти уроки. Пока что ничего даже близкого по качеству о C++ я не видел. Даже на Pluralsight всё намного хуже.

[Ответить](#)



1. *Юрий:*

[6 апреля 2020 в 20:18](#)

Пожалуйста))

[Ответить](#)



3. *Павел:*

[1 апреля 2020 в 21:03](#)

Менее половины дня потребовалось дойти до этого урока, всё настолько просто и доходчиво объяснено, что вопросов не остаётся.

Вам учебники нужно писать для образовательных учреждений.

[Ответить](#)



1. *Юрий:*

[1 апреля 2020 в 23:51](#)

Мне вот бы тоже хотелось, чтобы такая подача была и в образовательных учреждениях. Но этому вряд ли быть))

[Ответить](#)



4. *Георгий:*

[21 января 2020 в 12:22](#)

Годнее контента я ещё не видел.... Автору прям большой плюс и спасибо за столь хорошее объяснение. Столько книг уже по C++ в 21 веке и не в одной так доходчиво и на примере, не разъяснено как тут. Не понимаю зачем книги по C++, если есть эта божественная статья. Эти статьи напрочь отбивают вопросы которые бы возникли где нибудь ещё. Если по C++ ты читаешь книгу, то там куча вопросов возникает еще в начале первой странице. А тут все по порядку и красиво сделано

[Ответить](#)



1. *Юрий:*

[22 января 2020 в 00:51](#)

Спасибо, очень приятно 😊

[Ответить](#)



2. *Рустам:*

[11 февраля 2020 в 13:14](#)

Читаю, не могу остановиться!

[Ответить](#)



5. *Никита:*

[15 июня 2019 в 10:09](#)

Че так годно то?

[Ответить](#)

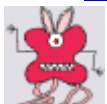


1. *Юрий:*

[18 июня 2019 в 13:25](#)

Ну шю есть, то есть)

[Ответить](#)



6. *Алексей:*

[2 июня 2019 в 09:21](#)

В предпоследнем примере вы утверждаете, что препроцессор не знает что такое функция и внутри неё работать не будет, а в конце статьи мы видим обратное. Что же верно?

[Ответить](#)



1. *Юрий:*

[6 июня 2019 в 14:54](#)

Верно то, что написано в уроке. Пройдите ещё с 10-ок уроков и возвратитесь к этому уроку и перечитайте его. Возможно, станет понятнее.

[Ответить](#)



2. *Олег:*

[17 января 2020 в 09:44](#)

Я понял так что препроцессор не знает что такое функция т.е. для него это пустое место, следовательно

```
1 | #define MY_NAME "Alex"
```

работать будет.

Просто для большей читабельности и возможно еще чего-нибудь, надо ставить его вверху.

[Ответить](#)



7. *VI:*

[22 мая 2019 в 01:38](#)

Насколько я понимаю, суть может быть вот в чем (очень приближенно). Вот у вас программа пишется для 64 бит системы и для 32. Но в сам код это вставлять не хочется. Пускай препроцессор сам выбирает тип компиляции в зависимости от системы. То есть на одном этапе надо так скомпилировать программу, а на другом эдак. Но получилось так, что мы можем использовать это как переменные)))

[Ответить](#)



8. *Сергей:*

[7 марта 2019 в 17:44](#)

Компилируется только без " std::cout << "Not printing!"; "

Как только пишу эту строчку компилятор выдает ошибку " error C2065: cout: необъявленный идентификатор ". В чем же проблема?

[Ответить](#)



1. *Steve Dekart:*

[18 июня 2019 в 18:42](#)

```
1 | std::cout << "Not printing!"; "
```

" — Это символ, который нужно тебе убрать, для чего ты его туда вообще поставил? Он лишний, т.к. сам по себе ты окончил стейтмент точкой с запятой, но зачем то ты ставишь ". Пиши так:


```
1 | std::cout << "Not printing!";
```

[Ответить](#)9. *Oleksiy:*[10 августа 2018 в 11:24](#)

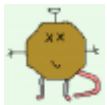
В общем понятно. Преппроцессор берет составленную нами прогу и заменяет в ней все директивы на что-то свое. В результате получаем прогу, которая все еще написана на C++, но директив со значками # там уже нет. Т.е. это предварительное "переживывание" программы перед тем, как она компилируется.

[Ответить](#)10. *Oleksiy:*[10 августа 2018 в 09:45](#)

Подтверждаю, эта глава написана не очень доходчиво для новичка. Может, проблема со стилем? Читатель (я) постоянно задает вопрос: а зачем эти директивы нужны? Нет мотивации схватить материал за рога. Буду перечитывать еще раз. С предыдущими главами такого не было.

[Ответить](#)11. *Вячеслав:*[8 июля 2018 в 14:58](#)

Директивы, определенные в одном файле кода, не влияют на код других файлов того же проекта. Не согласен, а если в файле `function.cpp` у нас будет написано `#define PRINT`, а в файле `main.cpp` напишем `#include "function.cpp"` — `PRINT` будет влиять на код в обоих файлах. Все зависит от порядка подключения. А определения передаваемые в аргументы компилятора видны, вообще, всем файлам проекта.

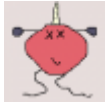
[Ответить](#)12. *Алексей:*[25 июня 2018 в 15:56](#)

Здравствуйте. Можно ли определить с помощью команды к примеру `#define MY_DEFINE` в файле `main.cpp` в файле `add.cpp`?

[Ответить](#)1. *Юрий:*[26 июня 2018 в 22:34](#)

В файле `main.cpp` в файле `add.cpp`? Я что-то не понял, что вы имеете в виду. `MY_DEFINE` можно определить только один раз и в одном файле проекта.

[Ответить](#)



13. *илья:*

[15 июня 2018 в 17:14](#)

а мы будем проходить работу с графикой в программировании?

[Ответить](#)

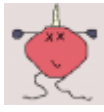


1. *Юрий:*

[15 июня 2018 в 18:29](#)

В этих уроках конкретно с графикой материала не будет. Здесь дается необходимый фундамент знаний, исходя из которого вы уже сами можете углубляться в любую отрасль программирования.

[Ответить](#)



1. *илья:*

[15 июня 2018 в 18:44](#)

ПОНЯТНО,спасибо

[Ответить](#)

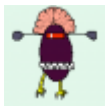


14. *Дима:*

[6 июня 2018 в 09:57](#)

Сложновато(
Пока только начинаем изучать, то вернемся попоже)

[Ответить](#)



15. *Михаил:*

[24 мая 2018 в 18:07](#)

тяжко заходит эта тема с дерективами (((

[Ответить](#)



1. *Михаил:*

[25 мая 2018 в 02:14](#)

есть может быть варианты как попроще объяснить эту тему? А то к меня шарики за ролики заехали. Эта тема важна, поэтому хочу в ней разобраться,чтобы не было пробелов

[Ответить](#)



1. *Юрий:*

[26 мая 2018 в 00:33](#)

Перечитайте статью несколько раз, поищите дополнительно информацию в Интернете, поищите в YouTube, загляните в буржунет (если знаете английский), спросите у знакомых программистов, чтобы объяснили вам детальнее, скачайте книги по C++ — посмотрите в них информацию насчет директив преппроцессора. Или, как вариант, мне переписать статью полностью?

[Ответить](#)



1. *Taueron:*

[9 января 2019 в 03:39](#)

Как вариант переписать полностью, раз уж взялись. Спасибо за Ваши труды.



16. *Максим():*

[13 мая 2018 в 21:44](#)

Да уж, тема сложновата... Придется посидеть...

[Ответить](#)



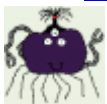
1. *painkiller:*

[18 мая 2018 в 17:38](#)

Да, мне сперва тоже так показалось, но после повторного прочтения всё стало на свои места.

В предыдущем уроке оставил несколько вопросов в комментариях, но благодаря этому уроку ответов на них уже не требуется — до всего дошел сам.

[Ответить](#)



17. *Сергей:*

[9 мая 2018 в 09:44](#)

Что-то вообще не понятно для чего эта ерунда нужна. Где она используется, для чего она? Лично мне видится, что она существует чтобы просто была.

[Ответить](#)

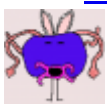


1. *Юрий:*

[9 мая 2018 в 13:14](#)

Если не увидели или не поняли смысла — значит его нет.

[Ответить](#)



18. *Vlados:*

[5 мая 2018 в 20:03](#)

Несмотря на то, что PRINT был определен в main.cpp, он все равно не имеет никакого влияния на что-либо в function.cpp.

PRINT был объявлен же, а не определён.

Иди я что-то пугаю

[Ответить](#)



1. *Юрий:*

[5 мая 2018 в 22:21](#)

Исправил. В этом случае директива #defined используется как в качестве объявления, так и определения. Т.е. это одно и то же (в этом случае).

[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