

Ravesli [Ravesli](#)


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


Урок №19. Прототип функции и Предварительное объявление

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 |

 55820

[1](#)  16

На этом уроке мы рассмотрим прототип функции и предварительное объявление в языке C++.

Оглавление:

1. [Наличие проблемы](#)
2. [Прототипы функций и Предварительное объявление](#)
3. [Предварительно объявили, но не определили](#)
4. [Объявление vs. Определение](#)
5. [Тест](#)
6. [Ответы](#)

Наличие проблемы

Посмотрите на этот, казалось бы, невинный кусочек кода под названием add.cpp:

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "The sum of 3 and 4 is: " << add(3, 4) << std::endl;
6      return 0;
7  }
8
9  int add(int x, int y)
10 {
11     return x + y;
12 }
```

Вы, наверное, ожидаете увидеть примерно следующий результат:

The sum of 3 and 4 is: 7

Но в действительности эта программа даже не скомпилируется. Причиной этому является то, что компилятор читает код последовательно. Когда он встречает вызов функции `add()` в строке №5 функции `main()`, он даже не знает, что такое `add()`, так как это еще не определили! В результате чего мы получим следующую ошибку:

`add`: идентификатор не найден

Чтобы устранить эту проблему, мы должны учитывать тот факт, что компилятор не знает, что такое `add()`. Есть 2 решения.

Решение №1: Поместить определение функции `add()` выше её вызова (т.е. перед функцией `main()`):

```
1  #include <iostream>
2
3  int add(int x, int y)
4  {
5      return x + y;
6  }
7
8  int main()
9  {
10     std::cout << "The sum of 3 and 4 is: " << add(3, 4) << std::endl;
11     return 0;
12 }
```

Таким образом, при вызове функции `add()` в функции `main()`, компилятор будет знать, что это такое. Так как это простая программа, то внести подобные изменения несложно. Однако в программах, содержащих большое количество строк кода, это может быть утомительно — узнавать кто кого вызывает и в каком порядке (чтобы соблюсти правильную последовательность).

Кроме того, этот вариант не всегда возможен. Например, мы пишем программу, которая имеет две функции: А и В. Если функция А вызывает функцию В, а функция В вызывает функцию А, то нет никакого способа упорядочить эти функции таким образом, чтобы они обе одновременно знали о существовании друг друга. Если вы объявите сначала А, то компилятор будет жаловаться, что не знает, что такое В. Если вы объявите сначала В, то компилятор будет жаловаться, что не знает, что такое А.

Прототипы функций и Предварительное объявление

Решение №2: Использовать предварительное объявление.

Предварительное объявление сообщает компилятору о существовании идентификатора ДО его фактического определения.

В случае функций, мы можем сообщить компилятору о существовании функции до её фактического определения. Для этого нам следует использовать прототип этой функции. **Прототип функции** (полноценный) состоит из типа возврата функции, её имени и параметров (тип + имя параметра). В кратком прототипе отсутствуют имена параметров функции. Основная часть (между фигурными

скобками) опускается. А поскольку прототип функции является стейтментом, то он также заканчивается точкой с запятой.

Вот прототип функции `add()`:

```
1 | int add(int x, int y); // прототип функции состоит из типа возврата функции, её имени
```

А вот вышеприведенная программа, но уже с прототипом функции в качестве предварительного объявления `add()`:

```
1 | #include <iostream>
2 |
3 | int add(int x, int y); // предварительное объявление функции add() (используется её
4 |
5 | int main()
6 | {
7 |     std::cout << "The sum of 3 and 4 is: " << add(3, 4) << std::endl; // это работае
8 |     return 0;
9 | }
10 |
11 | int add(int x, int y) // хотя определение функции add() находится ниже её вызова
12 | {
13 |     return x + y;
14 | }
```

Теперь, когда компилятор встречает вызов функции `add()` в `main()`, он знает, что это такое и где это искать.

Стоит отметить, что в прототипах функций можно и не указывать имена параметров. Например, прототип выше мы можем записать следующим образом:

```
1 | int add(int, int);
```

Тем не менее, предпочтительнее указывать имена параметров, чтобы не путаться лишний раз.

Лайфхак: Прототипы функций можно легко создавать с помощью копирования/вставки из фактического определения функции. Просто не забывайте указывать точку с запятой в конце.

Предварительно объявили, но не определили

Вопрос: «А что произойдет, если мы предварительно объявим функцию, но не запишем её определение?». Однозначного ответа нет. Если предварительное объявление записано, но функция никогда не вызывается, то программа может запуститься без ошибок. Однако, если предварительное объявление записано, функция вызывается, но её определения нет, то вы получите ошибку на этапе линкинга: программа просто не сможет обработать вызов этой функции.

Рассмотрим следующую программу:

```
1 | #include <iostream>
2 |
3 | int add(int x, int y); // предварительное объявление функции add() (используется её
4 |
```

```
5 | int main()
6 | {
7 |     std::cout << "The sum of 3 and 4 is: " << add(3, 4) << std::endl;
8 |     return 0;
9 | }
```

В этой программе мы предварительно объявили функцию `add()`, вызвали её в `main()`, но не записали её определения. При попытке компиляции этой программы мы получим ошибку от линкера.

Объявление vs. Определение

В языке C++ вы часто будете слышать слова «объявление» и «определение». Что это такое?

Определение фактически реализует (вызывает выделение памяти) идентификатор. Вот примеры определений:

```
1 | int add(int x, int y) // определяем функцию add()
2 | {
3 |     int z = x + y; // определяем переменную z
4 |
5 |     return z;
6 | }
```

Определение необходимо для корректной работы линкера. Если вы используете идентификатор без его определения, то линкер выдаст вам ошибку.

В языке C++ есть **правило одного определения**, которое состоит из 3-х частей:

- ➔ *Внутри файла* функция, объект, тип или шаблон могут иметь только одно определение.
- ➔ *Внутри программы* объект или обычная функция могут иметь только одно определение.
- ➔ *Внутри программы* типы, шаблоны функций и встроенные функции могут иметь несколько определений, если они идентичны.

Нарушение первой части правила приведет к ошибке компиляции. Нарушение второй или третьей части правила приведет к ошибке линкинга.

Объявление — это стейтмент, который сообщает компилятору о существовании идентификатора и о его типе. Вот примеры объявлений:

```
1 | int add(int x, int y); // сообщаем компилятору о функции add(), которая имеет два па
2 | int x; // объявляем целочисленную переменную x
```

Объявление — это всё, что необходимо для корректной работы компилятора, но недостаточно для корректной работы линкера. Определение — это то, что обеспечит корректную работу как компилятора, так и линкера.

Тест

Задание №1: В чём разница между прототипом функции и предварительным объявлением?

Задание №2: Запишите прототип следующей функции:

```
1 int doMath(int first, int second, int third, int fourth)
2 {
3     return first + second * third / fourth;
4 }
```

Задание №3: Выясните, какие из следующих программ не пройдут этап компиляции, какие не пройдут этап линкинга, а какие не пройдут и то, и другое?

Программа №1:

```
1 #include <iostream>
2
3 int add(int x, int y);
4
5 int main()
6 {
7     std::cout << "3 + 4 + 5 = " << add(3, 4, 5) << std::endl;
8     return 0;
9 }
10
11 int add(int x, int y)
12 {
13     return x + y;
14 }
```

Программа №2:

```
1 #include <iostream>
2
3 int add(int x, int y);
4
5 int main()
6 {
7     std::cout << "3 + 4 + 5 = " << add(3, 4, 5) << std::endl;
8     return 0;
9 }
10
11 int add(int x, int y, int z)
12 {
13     return x + y + z;
14 }
```

Программа №3:

```
1 #include <iostream>
2
3 int add(int x, int y);
4
5 int main()
6 {
7     std::cout << "3 + 4 + 5 = " << add(3, 4) << std::endl;
```

```
8     return 0;
9 }
10
11 int add(int x, int y, int z)
12 {
13     return x + y + z;
14 }
```

Программа №4:

```
1 #include <iostream>
2
3 int add(int x, int y, int z);
4
5 int main()
6 {
7     std::cout << "3 + 4 + 5 = " << add(3, 4, 5) << std::endl;
8     return 0;
9 }
10
11 int add(int x, int y, int z)
12 {
13     return x + y + z;
14 }
```

Ответы

Чтобы посмотреть ответ, кликните на него мышкой.

Ответ №1

Прототип функции (полноценный) — это стейтмент объявления функции, который состоит из типа возврата функции, её имени и параметров (тип + имя параметра). В кратком прототипе отсутствуют имена параметров функции. Тело функции не записывается.

Предварительное объявление сообщает компилятору о существовании идентификатора до его фактического определения.

Для функций прототип является предварительным объявлением.

Ответ №2

```
1 // Любой из этих прототипов является правильным
2 // Не забывайте указывать точку с запятой в конце
3
4 int doMath(int first, int second, int third, int fourth); // лучшее решение
5 int doMath(int, int, int, int);                          // альтернативное решение
```

Ответ №3

Программа №1: Не скомпилируется. Компилятор будет жаловаться, что слишком много аргументов в вызове функции `add()`.

Программа №2: Не скомпилируется. Компилятор будет жаловаться на то, что вызов функции `add()` не может принять столько аргументов.

Программа №3: Провал на этапе линкинга. Функция `add()`, которая принимает два параметра, не была определена (мы определили функцию, которая принимает 3 параметра).

Программа №4: Успешная компиляция и линкинг. Вызов функции `add()` соответствует и прототипу, который был объявлен, и определению функции.

Оценить статью:

★★★★★ (619 оценок, среднее: 4,93 из 5)



[← Урок №18. Базовое форматирование кода](#)



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

Комментариев: 16



1. *Наталья:*

[23 сентября 2020 в 22:19](#)

Извините, но правило одного определения, насколько я знаю, не только говорит о том, что у каждого объекта должно быть только одно определение, но и уточняет ситуацию с дублированием определений, а именно:

два определения класса, шаблона или встроенной функции приемлемы в качестве определения одной и той же сущности тогда и только тогда, когда:

1. они находятся в различных единицах компиляции;
2. они идентичны лексема за лексемой;
3. значения лексем одинаковы в обеих единицах компиляции.

[Ответить](#)



1. *Константин:*

[25 сентября 2020 в 19:36](#)

эй, Наталья-красотка! А ну дай ссылочку где таким академическим языком излагают — жуть как такое люблю читать!

[Ответить](#)



2. *Николай:*

[3 сентября 2020 в 18:22](#)

Юрий, добрый день.

Утверждение в ответе №1 "Для функций прототип является предварительным объявлением." понятно.

Уточните пожалуйста, для чего тогда прототип НЕ является предварительным объявлением (переменная, объект)?

Заранее благодарю.

[Ответить](#)



3. *Egor:*

[7 мая 2020 в 09:47](#)

Спасибо за статью!

[Ответить](#)



4. *Олег:*

[18 декабря 2018 в 14:44](#)

Юрий. огромное спасибо за Ваш труд и за ваше терпение в разборе ошибок.

[Ответить](#)



5. *Vadim:*

[5 апреля 2018 в 18:46](#)

```
1 #include <iostream>
2 #include "stdafx.h"
3
4 int add(int x, int y)
5 {
6     return x + y;
7 }
8
9 int main()
10 {
11     using namespace std;
12     cout << "The sum of 3 and 4 is: " << add(3, 4) << endl;
13     return 0;
14 }
```

[Ответить](#)



1. *Юрий:*

[5 апреля 2018 в 19:04](#)

Вы предыдущие уроки читали? Решение вашей ошибки находится сразу в двух уроках — 5 и 7.

`#include «stdafx.h»` всегда прописывается в коде первой строчкой, всегда первой (не второй и не третьей).

[Ответить](#)



1. *Vadim:*

[5 апреля 2018 в 19:09](#)

Спасибо за помощь в поиске ошибки.

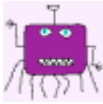
[Ответить](#)



1. *Юрий:*

[5 апреля 2018 в 19:36](#)

Пожалуйста.



2. *Никита:*

[15 июня 2019 в 06:16](#)

Ты конечно сверхразум написать `using namespace std` в функции `main()`

[Ответить](#)



1. *Константин:*

[8 мая 2020 в 13:26](#)

ну правильно — ограничил область видимости рамками `main()`

[Ответить](#)



6. *Vadim:*

[3 апреля 2018 в 18:38](#)

"Решение 1" не работает. Пишет: "непредвиденный конец файла во время поиска предкомпилированного заголовка. Возможно вы забыли добавить директиву `"#include \"stdafx.h\"`" в источник." Как исправить ошибку?

[Ответить](#)



1. *Юрий:*

[3 апреля 2018 в 20:58](#)

Добавить директиву `#include "stdafx.h"` в код.

[Ответить](#)



1. *Vadim:*

[3 апреля 2018 в 23:09](#)

После добавления возникают новые ошибки:

1. Не удаётся открыть источник
файл "stdafx.h".

2. cout: необъявленный
идентификатор (строка 11).

3. endl: необъявленный
идентификатор (строка 11).

[Ответить](#)



1. *Юрий:*

[4 апреля 2018 в 21:49](#)

Скиньте сюда ваш код, в котором возникают ошибки.



2. *Эрик:*

[3 февраля 2019 в 12:09](#)

Он в визуал студион 15, создал не проект видимо, а просто файл. Поэтому у него не создался заголовочный файл stdafx.h. И когда он объявляет её в начале, ошибка выходит, так как в проекте нет этого файла. Ему нужно создать именно проект, а не просто файл.

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

Электронная почта



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