

Ravesli [Ravesli](#)


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


Урок №50. Почему глобальные переменные – зло?

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 11 Сен 2020 |

 36427

[1](#)  16

Если вы попросите ветерана-программиста дать один дельный совет о программировании, то после некоторого раздумья он ответит: «Избегайте использования глобальных переменных!». И, частично, он будет прав. **Глобальные переменные** являются одними из самых злоупотребляемых объектов в языке C++. Хотя они и выглядят безвредными в небольших программах, использование их в крупных проектах зачастую чрезвычайно проблематично.

Новички часто используют огромное количество глобальных переменных, потому что с ними легко работать, особенно когда задействовано много функций. Это плохая идея. Многие разработчики считают, что неконстантные глобальные переменные вообще не следует использовать!

Но прежде, чем мы разберемся с вопросом «Почему?», нужно кое-что уточнить. Когда разработчики говорят, что глобальные переменные — это зло, они НЕ подразумевают полностью ВСЕ глобальные переменные. Они говорят о неконстантных глобальных переменных.

Оглавление:

1. [Почему \(неконстантные\) глобальные переменные — это зло?](#)
2. [В чём плюсы использования \(неконстантных\) глобальных переменных?](#)
3. [Как защититься от «глобального разрушения»?](#)
4. [Шутка](#)
5. [Заключение](#)

Почему (неконстантные) глобальные переменные — это зло?

Безусловно, причина №1, почему неконстантные глобальные переменные являются опасными, — это то, что их значения могут изменять любые вызываемые функции, при этом вы можете этого и не знать. Например, рассмотрим следующую программу:

```
1 #include <iostream>
2
3 // Объявление глобальной переменной
4 int g_mode;
5
6 void doSomething()
7 {
8     g_mode = 2; // присваиваем глобальной переменной g_mode значение 2
9 }
10
11 int main()
12 {
13     g_mode = 1; // примечание: Здесь мы присваиваем глобальной переменной g_mode значение 1
14
15     doSomething();
16
17     // Программист по-прежнему ожидает, что g_mode будет 1.
18     // Но функция doSomething() изменила значение этой переменной на 2!
19
20     if (g_mode == 1)
21         std::cout << "No threat detected.\n";
22     else
23         std::cout << "Launching nuclear missiles...\n";
24
25     return 0;
26 }
```

Результат выполнения программы:

Launching nuclear missiles...

Сначала мы присваиваем переменной `g_mode` значение 1, а затем вызываем функцию `doSomething()`. Если бы мы не знали заранее, что `doSomething()` изменит значение `g_mode`, то, вероятно, не ожидали бы дальнейшего развития событий (`g_mode = 2` => `Launching nuclear missiles...`)!

Неконстантные глобальные переменные делают каждую функцию потенциально опасной, и программист не может заранее знать, какая из используемых им функций является опасной, а какая — нет. Локальные переменные намного безопаснее, потому что другие функции не могут влиять на них напрямую.

Также есть много других веских причин не использовать неконстантные глобальные переменные. Например, нередко можно встретить примерно следующее:

```
1 void boo()
2 {
3     // Некоторый код
4
5     if (g_mode == 4) // делаем что-нибудь полезное
6 }
```

Предположим, что `g_mode` равно 3, а не 4 — наша программа выдаст неверные результаты. Как это исправить? Нужно будет отыскать все места, где предположительно могло измениться значение

переменной `g_mode`, а затем проследить ход выполнения кода в каждом потенциально опасном участке. Возможно, изменение глобальной переменной вы обнаружите вообще в другом коде, который, как вам казалось на первый взгляд, никак не связан с фрагментом, приведенным выше.

Одной из причин объявления локальных переменных максимально близко к месту их первого использования является уменьшение количества кода, которое нужно будет просмотреть, чтобы понять, что делает (зачем нужна?) переменная. С глобальными переменными дела обстоят несколько иначе — поскольку их можно использовать в любом месте программы, то вам придется просмотреть чуть ли не весь код, чтобы проследить логику выполнения и изменения значений переменных в вашей программе.

Например, вы можете обнаружить, что на `g_mode` ссылаются 442 раза в вашей программе. Если использования переменной `g_mode` не подкреплены комментариями, то вам придется просмотреть каждое упоминание `g_mode`, чтобы понять, как оно используется в разных случаях.

Также глобальные переменные делают вашу программу менее модульной и гибкой. Функция, которая использует только свои параметры и не имеет побочных эффектов, является идеальной в плане модульности. Модульность помогает понять структуру вашей программы, что она делает и как можно повторно использовать определенные участки кода в другой программе. Глобальные переменные значительно уменьшают эту возможность.

В частности, не используйте глобальные переменные в качестве важных переменных, которые выполняют главные или решающие функции в программе (например, переменные, которые используются в условных стејтментах, как `g_mode` выше). Ваша программа вряд ли сломается, если в ней будет глобальная переменная с информационным значением, которое может меняться (например, имя пользователя). Гораздо хуже, если изменится значение глобальной переменной, которая влияет непосредственно на результаты выполнения самой программы или на её работу.

Правило: Вместо глобальных переменных используйте локальные (когда это целесообразно).

В чём плюсы использования (неконстантных) глобальных переменных?

Их немного. Зачастую можно обойтись без использования неконстантных глобальных переменных. Но в некоторых случаях их разумное использование поможет уменьшить сложность программы, и, иногда, может быть даже лучшим вариантом для решения проблемы, чем альтернативные способы.

Например, если ваша программа использует базу данных для чтения и записи данных, то имеет смысл определить базу данных глобально, поскольку доступ к ней может понадобиться с любого места. Аналогично, если в вашей программе есть журнал ошибок (или журнал отладки), в котором вы можете читать/записывать информацию об ошибках (или об отладке), то имеет смысл определить его глобально. Звуковая библиотека может быть еще одним хорошим примером: вам, вероятно, не захочется открывать доступ к ней для каждой функции, которая делает запрос. Поскольку у вас будет только одна звуковая библиотека, управляющая всеми звуками, логично будет объявить её глобально, инициализировать при запуске программы, а затем использовать только в режиме чтения.

Как защититься от «глобального разрушения»?

Если у вас возникнет ситуация, где полезнее будет использовать неконстантные глобальные переменные, вместо локальных, то вот вам несколько полезных советов, которые помогут свести к

минимуму количество потенциальных проблем, с которыми вы можете столкнуться при использовании подобных переменных.

Во-первых, добавляйте префикс `g_` ко всем вашим глобальным переменным и/или размещайте их в пространстве имен, дабы уменьшить вероятность возникновения **конфликтов имен**.

Например, вместо следующего:

```
1 #include <iostream>
2
3 double gravity (9.8); // по имени переменной непонятно, глобальная ли это переменная
4
5 int main()
6 {
7     return 0;
8 }
```

Сделайте следующее:

```
1 #include <iostream>
2
3 double g_gravity (9.8); // теперь понятно, что это глобальная переменная
4
5 int main()
6 {
7     return 0;
8 }
```

Во-вторых, вместо разрешения прямого доступа к глобальным переменным, лучше их «инкапсулировать». Сначала добавьте ключевое слово `static`, чтобы доступ к ним был возможен только из файла, в котором они объявлены. Затем напишите внешние глобальные «функции доступа» для работы с переменными. Эти функции помогут обеспечить надлежащее использование переменных (например, при проверке ввода, проверке допустимого диапазона значений и т.д.). Кроме того, если вы когда-либо решите изменить первоначальную реализацию программы (например, перейти из одной базы данных в другую), то вам нужно будет обновить только *функции доступа* вместо каждого фрагмента кода, который напрямую использует глобальные переменные.

Например, вместо следующего:

```
1 double g_gravity (9.8); // можно экспортировать и использовать напрямую в любом файле
```

Сделайте следующее:

```
1 static double g_gravity (9.8); // ограничиваем доступ к переменной только на этот файл
2
3 double getGravity() // эта функция может быть экспортирована в другие файлы для доступа
4 {
5     return g_gravity;
6 }
```

В-третьих, при написании автономной функции, использующей глобальные переменные, не используйте их непосредственно в теле функции. Передавайте их в качестве параметров. Таким

образом, если в вашей функции нужно будет когда-либо использовать другое значение, то вы сможете просто изменить параметр. Это улучшит модульность вашей программы.

Вместо следующего:

```
1 // Эта функция полезна только для расчета мгновенной скорости на основе глобальной гравитации.
2 double instantVelocity(int time)
3 {
4     return g_gravity * time;
5 }
```

Сделайте следующее:

```
1 // Эта функция вычисляет мгновенную скорость для любого значения гравитации.
2 // Передайте возвращаемое значение из getGravity() в параметр gravity, если хотите использовать его.
3 double instantVelocity(int time, double gravity)
4 {
5     return gravity * time;
6 }
```

Наконец, изменение значений глобальных переменных — это прямой путь к проблемам. Структурируйте ваш код в соответствии с тем, что ваши глобальные переменные могут измениться. Постарайтесь свести к минимуму количество случаев, где они могут изменять свои значения — обращайтесь с ними исключительно как с *доступными только для чтения* (насколько позволяет ситуация). Если вы можете инициализировать значение глобальной переменной при запуске программы, а затем не изменять его в ходе выполнения, то, таким образом, вы снизите вероятность возникновения непредвиденных проблем.

Шутка

Какой наилучший префикс для глобальных переменных?

Ответ: //.

Заключение

Избегайте использования неконстантных глобальных переменных, насколько это возможно! Если же используете, то используйте их максимально разумно и осторожно.

Оценить статью:

★★★★★ (283 оценок, среднее: 4,94 из 5)



← [Урок №49. Глобальные переменные](#)



Комментариев: 16



1. *Codder:*

[31 мая 2020 в 15:17](#)

```
1 // Программист по-прежнему ожидает, что g_mode будет 1
2 // Но функция doSomething() изменила значение этой переменной на 2
```

Вот щас посмеялся. Это не программист, а ребенок, если не понимает как работает элементарный синтаксис.

Я даже не знаю на чем может базироваться его ожидание.

Если программист вызывает функцию для изменения глобальной переменной — значит так и было задумано при условии если он конечно дружит с головой и знает что делает.

[Ответить](#)



1. *Александр:*

[31 июля 2020 в 14:36](#)

Это же пример, брат.

Реальный код больше и в нем проще запутаться.

[Ответить](#)



2. *Kinonik:*

[8 декабря 2019 в 19:20](#)

Я бы не сказал, что глобалки прям таки зло. Я программирую не много, около 7-9 лет на C++. Единственной потенциальной угрозой я считаю потокобезопасность. К глобалкам доступ быстрее для методов, т.к. им не нужно ничего возвращать, они напрямую могут работать с файлами.

Самое нелепое, т.ч. способы, которые дают возможность обойтись без глобалок делают локалки глобальными в каком-то роде 😊

Вы передаёте в функцию какую-то переменную и он, представляете, МОЖЕТ ЕЁ МЕНЯТЬ! А если функция в другом потоке... ух...

[Ответить](#)



3. *ФывФыв:*

[15 июля 2019 в 12:01](#)

А в чём смысл шутки? Что-то я не догнал...

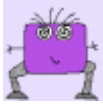
[Ответить](#)



1. *Юрий:*
[15 июля 2019 в 13:35](#)

Для шутки нужно степень учёную иметь. Не для вас походу)

[Ответить](#)



2. *Слава:*
[12 августа 2019 в 16:36](#)

// — превращает строку в комментарий.

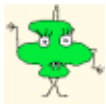
[Ответить](#)



3. *Юрец:*
[20 ноября 2019 в 19:49](#)

Это что-то на подобие "хороший черт — мертвый черт. То есть, самое лучшее применение глобальных переменных — это вообще не применять.

[Ответить](#)



4. *Алексей:*
[9 июля 2019 в 17:47](#)

Вердикт — осторожно использовать глобальность, или наломать дров.

Я думаю глобальность можно заюзать только после написания кода, крайне осторожно и только в целях оптимизации.

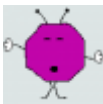
[Ответить](#)



5. *Евгений:*
[30 июля 2018 в 18:57](#)

Хотел спросить совета. Я вот хочу потренироваться путём написания текстового квеста (я раньше в Космических рейнджерах любил их проходить). Допустим, такие понятия как здоровье или количество денег объявить как глобальные переменные, или всё же лучше как статические, например в той же `main`? Так же предполагается, что квест не будет линейным: можно будет пойти из точки А в точку В, из неё в С, затем в D, а из неё обратно в В. При этом здоровье или деньги будут меняться.

[Ответить](#)



1. *Александр:*
[1 февраля 2019 в 15:41](#)

на том количестве знаний, что Вы уже успели прочитать в этих статьях, лучше сделать `static` переменные и функции доступа к ним (например — `getHealth()` возвращает здоровье,

setHealth(int h) устанавливает здоровье на фиксированное значение, charWounded(int c) — "ранит" персонажа по заданным правилам, зависящим от c)

Другим хорошим вариантом будет передача всех этих параметров в функции, но тогда заголовки этих функций окажутся просто чудовищными

Ну и самым хорошим вариантом будет создание класса "персонаж" 😊

[Ответить](#)



1. *Вячеслав:*

[11 февраля 2019 в 23:43](#)

Александр хотел задать вопрос собираюсь писать бота на криптобиржу что лучше сделать написать функцию или создать переменную, которая будет принимать с биржи данные через API ключи?

[Ответить](#)



2. *Codder:*

[31 мая 2020 в 15:21](#)

Конкретно в этом случае не вижу ничего плохого в глобальных переменных. Ведь здоровье может уменьшиться по 1000 различным причинам. Это такой параметр, к которому должен быть доступ у многих функций.

[Ответить](#)



6. *STM32_Den_Od:*

[22 октября 2017 в 11:33](#)

Добрый день. Спасибо за ваши статьи много нового узнаю. Програмирую на Си под микроконтроллеры. А как же быть с флагами?

Или семафорами которые должны быть глобальными, потому что они поднимаются в разных местах программы.

[Ответить](#)



1. *Юрий:*

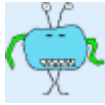
[22 октября 2017 в 12:30](#)

Привет. Не буду врать, не особо знаком с низкоуровневым программированием, могу ошибаться, но вот, что думаю. Во-первых, набор флагов ведь и так объявляется в качестве константных значений ([урок 46](#)). С семафорами, скорее всего, есть два варианта:

1. Либо объявить данные локально и вызывать их через функцию (используя [структуры](#) и классы (еще не рассматривали) для хранения и передавая эти значения по константной ссылке ([урок 89](#)) — тогда вы сможете гарантировать то, что эти значения не изменяться в программе).
2. Либо всё равно использовать глобальную переменную (и попробовать её encapsулировать, т.е. защитить от возможных изменений в программе).

Ведь у вас вопрос по защите этих данных от изменений или эти данные у вас изменяются на протяжении выполнения программы? Если изменяются и так и нужно, то `const` использовать нет надобности — просто объявите данные локально и вызывайте их через функцию (вариант 1), либо же, если значения не изменяются, то и проблемы не использовать `const` нет.

[Ответить](#)



2. *AleksandrM:*

[5 мая 2018 в 19:18](#)

Никак. Ну или почти никак. Самый простой вариант: объявить глобально. Только надо не просто объявлять(`int Flag;`), а объявлять и инициализировать(`int Flag=0;`), иначе есть шанс "выстрелить себе в ногу" при обращении к данной переменной.

[Ответить](#)



3. *Андрей:*

[25 июня 2018 в 13:15](#)

Бро давай свяжемся с тобой т.к. я тоже програмирую микроконтроллеры но на c++

[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 «SameGame» на C++/MFC](#)
-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

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