Ravesli Ravesli

- Уроки по C++
- OpenGL
- SFML
- Qt5
- RegExp
- Ассемблер
- Купить .PDF

# Урок №51. Статические переменные

Юрий |

- Уроки C++

|

Обновл. 22 Сен 2020 |

55545

| 10

Ключевое слово static является одним из самых запутанных в языке C++. Оно имеет разные значения в разных ситуациях.

На уроке о **глобальных переменных** мы узнали, что, добавляя static к переменной, объявленной вне блока, мы определяем её как внутреннюю, то есть такую, которую можно использовать только в файле, в котором она определена.

Ключевое слово static можно применять и к переменным внутри блока, но тогда его значение будет другим. На уроке о **локальных переменных** мы узнали, что локальные переменные имеют автоматическую продолжительность жизни, т.е. создаются, когда блок начинается, и уничтожаются при выходе из него.

Использование **ключевого слова static** с локальными переменными изменяет их свойство продолжительности жизни с автоматического на статическое (или *«фиксированное»*). **Статическая переменная** (или ***«переменная со статической продолжительностью жизни»***) сохраняет свое значение даже после выхода из блока, в котором она определена. То есть она создается (и инициализируется) только один раз, а затем сохраняется на протяжении выполнения всей программы.

Рассмотрим разницу между переменными с автоматической и статической продолжительностями жизни.

Автоматическая продолжительность жизни (по умолчанию):

```
1 #include <iostream>
2
3 void incrementAndPrint()
4 {
5     int value = 1; // автоматическая продолжительность жизни (по умолчанию)
6     ++value;
```

```
    std::cout << value << std::endl;
} // переменная value уничтожается здесь

int main()
{
    incrementAndPrint();
    incrementAndPrint();
    incrementAndPrint();
}
```

Каждый раз, при вызове функции incrementAndPrint(), создается переменная `value`, которой присваивается значение `1`. Функция incrementAndPrint() увеличивает значение переменной до `2`, а затем выводит его. Когда incrementAndPrint() завершает свое выполнение, переменная выходит из области видимости и уничтожается.

Следовательно, результат выполнения программы:

```
2
2
2
```

Теперь рассмотрим статическую версию. Единственная разница между этими двумя программами только в добавлении ключевого слова static к переменной.

Статическая продолжительность жизни:

```
#include <iostream>

void incrementAndPrint()
{
    static int s_value = 1; // переменная s_value является статической
    ++s_value;
    std::cout << s_value << std::endl;
} // переменная s_value не уничтожается здесь, но становится недоступной

int main()
{
    incrementAndPrint();
    incrementAndPrint();
    incrementAndPrint();
}
```

Поскольку переменная `s_value` объявлена статической (с помощью ключевого слова static), то она создается и инициализируется только один раз. Кроме того, выходя из области видимости, она не уничтожается. Каждый раз, при вызове функции incrementAndPrint(), значение `s_value` увеличивается.

Результат выполнения программы:

```
2
3
4
```

Так же, как мы используем префикс `g_` с глобальными переменными, префикс `s_` принято использовать со статическими переменными. Обратите внимание, внутренние глобальные переменные (которые объявлены с использованием static) остаются с префиксом `g_`, а не с префиксом `s_`.

Зачем нужны статические локальные переменные? Одним из наиболее распространенных применений является генерация уникальных идентификаторов. При работе с большим количеством одинаковых объектов внутри программы часто бывает полезно присвоить каждому объекту отдельный уникальный идентификационный номер. Это легко осуществить, используя одну статическую локальную переменную:

```
int generateID()
{
    static int s_itemID = 0;
    return s_itemID++;
}
```

При первом вызове функции возвращается `0`. Во второй раз возвращается `1`. Затем `2` и каждый последующий вызов будет увеличивать эту переменную на единицу. Хороший способ генерации уникальных идентификаторов для похожих объектов? Хороший! Поскольку `s_itemID` — это локальная переменная, то она не может быть «изменена» другими функциями.

Статические переменные имеют некоторые преимущества глобальных переменных (они не уничтожаются до завершения программы), сохраняя при этом локальную область видимости. Таким образом, они намного безопаснее для использования, нежели глобальные переменные.

## Тест

Какой эффект от добавления ключевого слова static к глобальной переменной? Какое влияние оно имеет на локальную переменную?

**Ответ**

Добавляя ключевое слово static к глобальной переменной, мы определяем её как внутреннюю, то есть такую, которую нельзя экспортировать и использовать в других файлах.

В случае с локальной переменной, добавление static определяет её как статическую, то есть она создается и инициализируется только один раз, и не уничтожается до самого конца программы.



Урок №50. Почему глобальные переменные – зло?

Урок №52. Связи, область видимости и продолжительность жизни

# Комментариев: 10

1.  *Misha_81*:
[17 октября 2020 в 22:36]

Статические локальные переменные в функции очень напомнили термин "Замыкание" в javascrip.

Ответить

2.  *Slava*:
[18 сентября 2020 в 22:42]

Очень хорошие уроки, начал читать затянулся. Сколько искал, гуглил, чтобы выучить язык, смотрел видео. Но данный сайт топ. Спасибо вам за вашу роботу Юрий.

Ответить

3.  *Олег*:
[18 июня 2020 в 20:02]

Весьма толково. Уже написал прогу высчитывания разверток…. Удобно. Очень Вам благодарен….

Ответить

4.  *Андрец*:
[8 февраля 2020 в 19:54]

Юрий, спасибо большое за эти уроки! Ну просто огромнейшее спасибо!

Ответить

    1.  *Юрий*:
    [8 февраля 2020 в 21:33]

    Большое пожалуйста 😉

    Ответить

5.  *Ильдар*:
[29 января 2020 в 14:45]

Поддерживаю одобрения об этом сайте и вашей работе.

По поводу урока: локальная переменная будет "жить" до конца, но и в тоже время изменять её можно только в блоке, в котором она была создана, верно?

Ответить

6.  *Sergey*:
[12 декабря 2019 в 06:58]

Полезные уроки.
Можно быстро найти ответ на интересующий вопрос, не вникая в талмуды.
Спасибо.

Ответить

1.  *Юрий*:
[12 декабря 2019 в 09:27]

Пожалуйста)

Ответить

7.  *Malkin*:
[25 сентября 2019 в 08:08]

Честно скажу, это лучшие уроки вообще в рунете из всего, что я гуглил много лет. Поскольку занимаюсь МК и ассемблером, важно было узнать, что происходит с памятью в процессе Си кода — стек, куча, все дела. Везде пишут, лишь как правильно писать код, а здесь автор объясняет всю поднаготную, особенно в дальнейших уроках. Бесконечно благодарен!

Ответить

1.  *Юрий*:
[25 сентября 2019 в 10:32]

Пожалуйста, мне очень приятно))

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.

Отправить комментарий

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)