

**ВЫ ТОЛЬКО ПОСМОТРИТЕ
НА ВСЕ ЭТИ ФИЧИ**



КОТОРЫЕ ВЫ НЕ ИСПОЛЬЗУЕТЕ

ООП/C++: Лекция 6

Инициализация, static-и, перегрузка операторов, строки

ака «Повседневные нюансы: новые фишки, совсем не как в Си»

О чём лекция сегодня

1. Инициализация
2. Статические поля и методы
3. Перегрузка операторов
4. Строки
5. (*) Загадка

<https://github.com/avasyukov/oop-2nd-term/tree/master/2019/lection06>



http://judge2.vdi.mipt.ru/cgi-bin/new-client?contest_id=911142



Инициализация

Инициализация

- В стандарте C++11 привнесен ряд новых способов инициализации переменных
- Учить их наизусть не надо, но многие из них удобные
- Некоторые посмотрим, больше, например, здесь:
 - <http://scrutator.me/post/2012/11/16/new-ctors-p1.aspx>
 - <http://scrutator.me/post/2012/12/18/new-ctors-p2.aspx>

Инициализация: примеры

Разбираем пример

- 01_initializers.cpp

Статические поля и методы

Что значит «статические»?

Статическое поле класса:

- Привязано «ко всем экземплярам класса сразу», никому из них лично не принадлежит
- При изменении (любым экземпляром класса или «просто так») меняется для всех объектов сразу

Статический метод класса:

- Привязан «ко всем экземплярам класса сразу», вызывается вне контекста конкретного экземпляра (грубо говоря, у него нет `this`)
- Может работать только с локальными переменными и статическими полями класса

Статические поля и методы: примеры

Разбираем пример

- 02_static_fields_and_methods.cpp

Ну и зачем это надо?

- Иногда хочется «выставить параметры» для всех экземпляров класса (причём это именно «параметры», а не константы, но параметры для всех экземпляров сразу)
- Фундамент для некоторых паттернов проектирования (например, Singleton)

Перегрузка операторов

Перегрузка операторов

- Можно описать, например, что для данного класса значат операторы '+' '-' '*' '/' '=' '==' '!=' и т.д.
- Можно описать, как класс читается из потока ввода и как пишется в поток вывода
- Это всё ради вызывающего кода, чтобы там можно было сказать, например, $c = a + b$ (при том что a и b , например, матрицы $N \times N$)
- Довольно редкая фишка среди языков программирования (C++ умеет в полном объёме)

Перегрузка операторов: примеры

Разбираем примеры

- 03_operator_overload_intro.cpp
- 04_operator_overload_cout_not_working.cpp
- 05_operator_overload_cout.cpp
- 06_operator_overload_sum.cpp
- 07_operator_overload_cin.cpp

Строки

Строки в С и в С++

- Строка в С – массив `char`-ов со спецсимволом на конце
- Строка в С++ – класс `string` (у которого внутри почти то же самое):
 - Внутри класса лежит массив символов
 - У класса есть методы – аналоги функций С для работы со строками (некоторые посмотрим в примерах)

Строки в С и в С++: как с ними жить

- Если есть массив `char`-ов, а нужна С++-ная строка – создайте класс `string`, отдав массив в его конструктор
- Если есть С++-ная строка, а нужен простой массив – попросите у класса `string` отдать хранимые символы (метод `c_str()`)
- Мешать в одном коде строки С и С++ не стоит, используйте или одно, или другое (иначе, как минимум, это будет страшно читаться)

Строки в С и в С++: примеры

Разбираем примеры

- 08_strings_basics.cpp
- 09_strings_io.cpp

(*) Загадка

Есть вот такой код

```
class Person
{
protected:
    string name;
public:
    Person(string _name) : name(_name) { }
    ~Person() { }

    string getName() const {
        return name;
    }
};

std::ostream& operator<<(std::ostream& os, const Person& p) {
    os << "(" << p.getName() << ")";
}

int main()
{
    Person p("Alice");
    cout << p << endl;
    return 0;
}
```

Это пример 10_challenge_operator_overload_error.cpp

Который вот так падает

```
(gdb) run
```

```
Starting program: /home/anganar/git/mipt/oop-2nd-term/2019/lection06/10  
_challenge_operator_overload_error
```

```
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.26-30.  
fc27.x86_64
```

```
Program received signal SIGSEGV, Segmentation fault.
```

```
0x00007ffff7b64d19 in std::basic_ostream<char, std::char_traits<char> >&  
    std::endl<char, std::char_traits<char> >(std::basic_ostream<char,  
    std::char_traits<char> >&) () from /lib64/libstdc++.so.6
```

```
Missing separate debuginfos, use: dnf debuginfo-install libgcc-7.3.1-6.  
fc27.x86_64 libstdc++-7.3.1-5.fc27.x86_64
```

```
(gdb) backtrace
```

```
#0 0x00007ffff7b64d19 in std::basic_ostream<char, std::char_traits<char>  
    > >& std::endl<char, std::char_traits<char> >(std::basic_ostream<  
    char, std::char_traits<char> >&) () from /lib64/libstdc++.so.6
```

```
#1 0x0000000000400c4d in main () at 10  
    _challenge_operator_overload_error.cpp:24
```

Вопрос: что это было?

Подобное бывает часто



mcc

@mcclure111

Follow



In C++ we don't say "Missing asterisk" we say "error C2664: 'void std::vector<block,std::allocator<_Ty>>::push_back(const block &)': cannot convert argument 1 from 'std::_Vector_iterator<std::_Vector_val<std::_Simple_types<block>>>' to 'block &&'" and i think that's beautiful

4:30 PM - 1 Jun 2018

292 Retweets 926 Likes



20



292



926



Что стоит запомнить из лекции

- В C++ (начиная с C++11) есть разные удобные способы инициализации переменных
- Есть static-и (однажды они пригодятся, зафиксируйте ключевое слово)
- В C++ можно перегрузить вообще любые операторы для своего класса, упаковав логику обращения с ним внутрь него самого (инкапсуляция, однако)
- Строки в C и C++ логически об одном, но технически вообще разные

В C++ есть много способов творчески сломать свой код, получив совершенно непонятные сообщения об ошибках. Это плата за гибкость и силу языка.

Что стоит запомнить из лекции

- В C++ (начиная с C++11) есть разные удобные способы инициализации переменных
- Есть static-и (однажды они пригодятся, зафиксируйте ключевое слово)
- В C++ можно перегрузить вообще любые операторы для своего класса, упаковав логику обращения с ним внутрь него самого (инкапсуляция, однако)
- Строки в C и C++ логически об одном, но технически вообще разные

В C++ есть много способов творчески сломать свой код, получив совершенно непонятные сообщения об ошибках. Это плата за гибкость и силу языка.

<https://tinyurl.com/y35to9vc>



OVERLOAD



ALL THE OPERATORS!