

EXCEPTIONS



EXCEPTIONS EVERYWHERE

ООП/С++: Лекция 10

Exceptions && smart pointers

ака «Как писать меньше спагетти-кода»

О чём лекция сегодня

1. Exceptions vs return codes
2. Использование exception-ов
3. Smart pointers

<https://github.com/avasyukov/oop-2nd-term/tree/master/2019/lection10>



http://judge2.vdi.mipt.ru/cgi-bin/new-client?contest_id=911147



Exceptions vs return codes

Традиционный путь – вернуть из функции код завершения («успешно» или код ошибки)

- Просто и понятно в первый момент
- Приводит к тому, что в реальном коде обработка ошибок занимает огромное количество строк и снижает общую читаемость
- Огромная вероятность того, что какой-то случай всё равно останется не обработанным

Что такое exception

Exception – логический «сигнал», который можно сгенерировать («бросить») в одном произвольном месте кода и обработать («поймать») в произвольном другом месте, находящемся выше по цепочке вызовов

Exceptions: базовые примеры

Смотрим примеры:

- `01_exceptions/01_simple_error_handling.cpp`
- `01_exceptions/02_hello_exception.cpp`
- `01_exceptions/03_real_life_error_handling.cpp`
- `01_exceptions/04_exceptions_for_the_rescue.cpp`

Использование exception-ов

Использование exception-ов (1/3)

Не надо использовать exception как замену return!

- Exception – фатальная проблема, с которой нельзя справиться, а не вариант штатной работы
- Если условный 1% вызовов приводит к exception, то это уже повод внимательно проверить логическую архитектуру
- Обработка реально выброшенного exception – достаточно дорогая операция

Использование exception-ов (2/3)

Не надо использовать exception в ситуациях, когда ваш код явно технически некорректно вызван

- Если ваш код некорректно использован – надо падать, чтобы заставить вызвавшего разобраться и починить код на своей стороне
- Exception – это когда корректный код столкнулся с фатально проблемными условиями работы

Использование exception-ов (3/3)

- Можно и нужно задавать классы exception-ов. Общее правило: класс описывает логическую ситуацию, а не место возникновения проблемы.
- Ловить exception-ы нужно только в том случае, если знаете, что делать с пойманным.
- Стоит помнить, что вообще любая строчка кода потенциально выбрасывает exception.

Использование exception-ов: примеры

Смотрим примеры:

- `01_exceptions/05_custom_exceptions.cpp`
- `01_exceptions/06_bad_exception_handling.cpp`
- `01_exceptions/07_still_bad_exception_handling.cpp`
- `01_exceptions/08_exception_in_constructor.cpp`

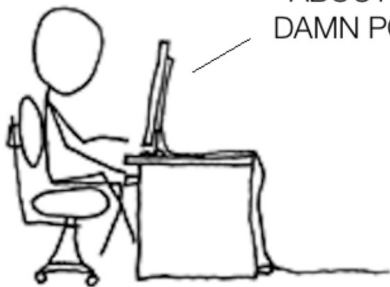
Exception-ы в реальной жизни

- Позволяют «локализовать» обработку проблем – проблема касается только того, кто обнаружил ошибку, и того, кто может её обработать (и не касается всех слоёв между ними).
- Позволяют разделить код штатной работы («happy path») и код обработки ошибок («error path»), сделать программу в целом читаемее.
- Для использования exception-ов нужно перестроить мышление. Проектирование обработки исключений – очень сложная часть реальной разработки.

OKAY HUMAN, LISTEN
UP! WE NEED TO TALK

HUH? ABOUT WHAT?

ABOUT YOUR
DAMN POINTERS ...



Smart pointers

Что такое smart pointer

Smart pointer:

- Специальная «тонкая» и «лёгкая» обёртка над «обычным» указателем
- Следит, насколько ещё жив владелец указателя. Если владелец уже мёртв, освобождает память.

Smart pointers: примеры

Смотрим примеры:

- `02_smart_pointers/01_the_problem.cpp`
- `02_smart_pointers/02_hello_smart_pointer.cpp`

Smart pointer, заметки на полях

Smart pointer, заметки на полях:

- При желании можете в качестве упражнения написать аналог сами. Это не слишком сложно.
- Да, это примерно «garbage collection без garbage collector-а». И это круто.

Виды smart pointer-ов (1/2)

- `unique_ptr`
 - ровно один владелец указателя
 - нельзя копировать, можно сменить владельца
 - используйте по умолчанию его, если не уверены, что именно требуется в конкретном случае
- `shared_ptr`
 - несколько владельцев указателя
 - можно копировать (это добавляет владельца)
 - считает количество ещё живых владельцев
 - освобождает память при смерти последнего

Виды smart pointer-ов (2/2)

- `weak_ptr`
 - используется только совместно с `shared_ptr`
 - даёт доступ к объекту, на который указывает `shared_ptr`, но не участвует в подсчёте ссылок
 - уместен для `observer-ов`
 - используется для борьбы с кольцевыми ссылками между `shared_ptr`
- `auto_ptr`
 - исторически «первый блин комом»
 - опасное поведение при копировании
 - считается устаревшим, не используйте его в новом коде, заменяйте на другие виды

Smart pointers: примеры

Смотрим примеры:

- `02_smart_pointers/03_smart_pointer_issue.cpp`
- `02_smart_pointers/04_shared_pointers.cpp`

Smart pointer + exception

Смотрим последний пример:

- `03_combination/happy_end.cpp`

Что стоит запомнить из лекции

Exceptions

- удобный способ обработки исключительных ситуаций, когда кода уже много
- дают более чистый и читаемый код в итоге
- требуют перестроить мышление

Smart pointers

- способ борьбы с утечками памяти
- выход из некоторых очень неприятных ситуаций

Если захочется больше: ключевое слово RAII

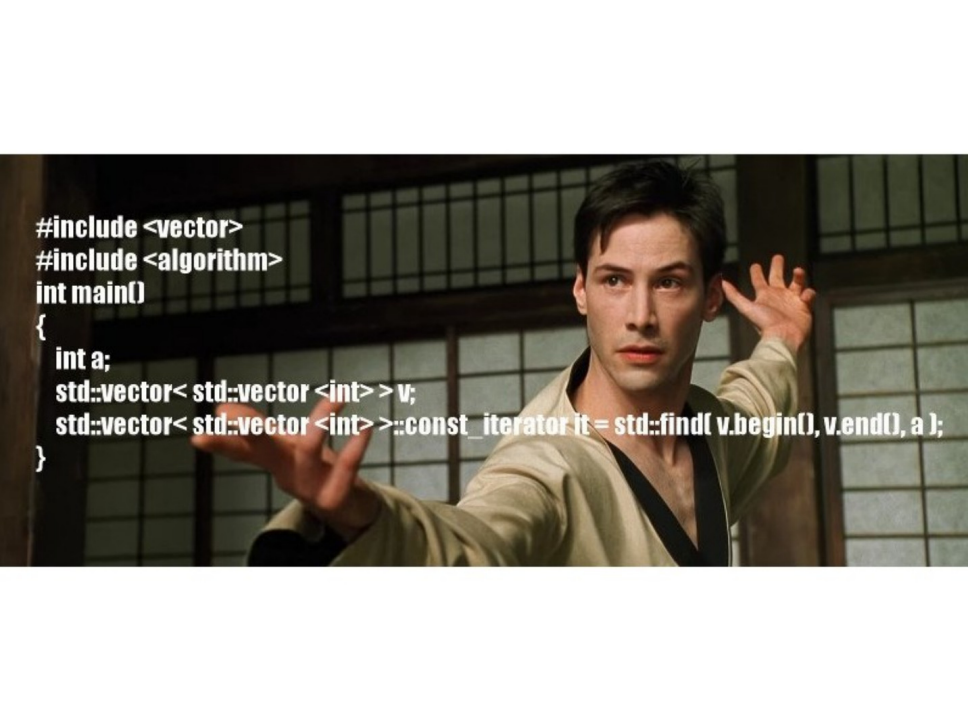
Контра близко



I know C++



Show me

A background image of Keanu Reeves in a martial arts pose, wearing a white gi, with his hands raised in a defensive or offensive stance. The setting appears to be a dojo with a window in the background.

```
#include <vector>  
#include <algorithm>  
int main()  
{  
    int a;  
    std::vector< std::vector<int> > v;  
    std::vector< std::vector<int> >::const_iterator it = std::find( v.begin(), v.end(), a );  
}
```

In file included from /usr/include/c++/4.6/algorithm:63:0,
from error_code.cpp:2:
/usr/include/c++/4.6/bits/stl_algo.h: In function 'std::find(_RandomAccessIterator, _RandomAccessIterator, const_Tp&, std::random_access_iterator_tag) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator*, std::vector >, _Tp = int]':
/usr/include/c++/4.6/bits/stl_algo.h:4403:45: instantiated from 'std::find(_Iter, _Iter, const_Tp&) [with _Iter = __gnu_cxx::__normal_iterator*, std::vector >, _Tp = int]'
error_code.cpp:8:89: instantiated from here
/usr/include/c++/4.6/bits/stl_algo.h:162:4: error: no match for 'operator==' in
'_first, __gnu_cxx::__normal_iterator::operator* [with Iterator = std::vector*, Container = std::vector >, __gnu_cxx::__normal_iterator::reference = std::vector&]() == __val'
/usr/include/c++/4.6/bits/stl_algo.h:162:4: note: candidates are:
/usr/include/c++/4.6/bits/stl_pair.h:201:5: note: template bool std::operator==(const std::pair&, const std::pair&)
/usr/include/c++/4.6/bits/stl_iterator.h:285:5: note: template bool std::operator==(const std::reverse_iterator&, const std::reverse_iterator&)
/usr/include/c++/4.6/bits/stl_iterator.h:335:5: note: template bool std::operator==(const std::reverse_iterator&, const std::reverse_iterator&)
/usr/include/c++/4.6/bits/allocator.h:122:5: note: template bool std::operator==(const std::allocator&, const std::allocator&)
/usr/include/c++/4.6/bits/allocator.h:127:5: note: template bool std::operator==(const std::allocator&, const std::allocator&)
/usr/include/c++/4.6/bits/stl_vector.h:1273:5: note: template bool std::operator==(const std::vector&, const std::vector&)
/usr/include/c++/4.6/ext/new_allocator.h:123:5: note: template bool __gnu_cxx::operator==(const __gnu_cxx::new_allocator&, const __gnu_cxx::new_allocator&)
/usr/include/c++/4.6/bits/stl_iterator.h:805:5: note: template bool __gnu_cxx::operator==(const __gnu_cxx::__normal_iterator&, const __gnu_cxx::__normal_iterator&)
/usr/include/c++/4.6/bits/stl_iterator.h:799:5: note: template bool __gnu_cxx::operator==(const __gnu_cxx::__normal_iterator&, const __gnu_cxx::__normal_iterator&)
/usr/include/c++/4.6/bits/stl_algo.h:4403:45: instantiated from 'std::find(_Iter, _Iter, const_Tp&) [with _Iter = __gnu_cxx::__normal_iterator*, std::vector >, _Tp = int]'
error_code.cpp:8:89: instantiated from here
/usr/include/c++/4.6/bits/stl_algo.h:166:4: error: no match for 'operator==' in
'_first, __gnu_cxx::__normal_iterator::operator* [with Iterator = std::vector*, Container = std::vector >, __gnu_cxx::__normal_iterator::reference = std::vector&]() == __val'
/usr/include/c++/4.6/bits/stl_algo.h:166:4: note: candidates are:
/usr/include/c++/4.6/bits/stl_pair.h:201:5: note: template bool std::operator==(const std::pair&, const std::pair&)
/usr/include/c++/4.6/bits/stl_iterator.h:285:5: note: template bool std::operator==(const std::reverse_iterator&, const std::reverse_iterator&)
/usr/include/c++/4.6/bits/stl_iterator.h:335:5: note: template bool std::operator==(const std::reverse_iterator&, const std::reverse_iterator&)
/usr/include/c++/4.6/bits/allocator.h:122:5: note: template bool std::operator==(const std::allocator&, const std::allocator&)
/usr/include/c++/4.6/bits/allocator.h:127:5: note: template bool std::operator==(const std::allocator&, const std::allocator&)
/usr/include/c++/4.6/bits/stl_vector.h:1273:5: note: template bool std::operator==(const std::vector&, const std::vector&)
/usr/include/c++/4.6/ext/new_allocator.h:123:5: note: template bool __gnu_cxx::operator==(const

<https://tinyurl.com/y3gp2g4f>

