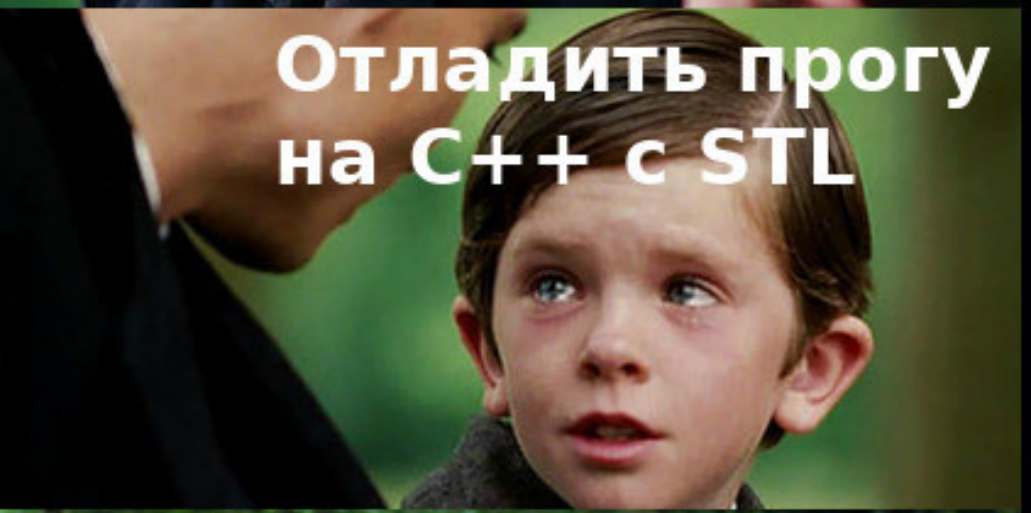
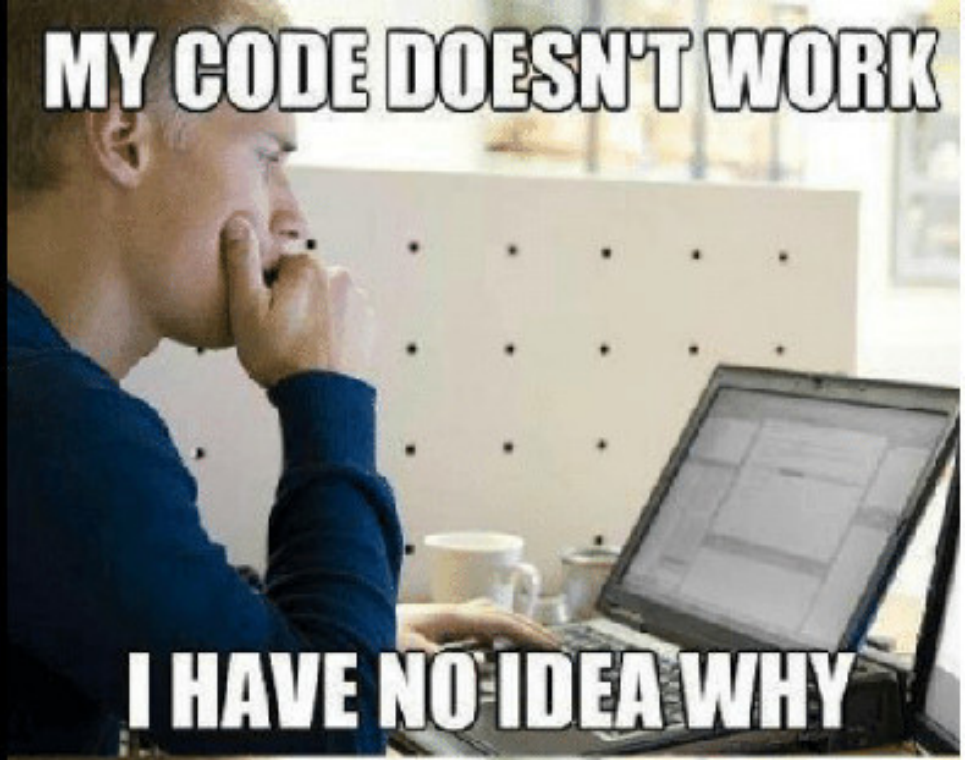
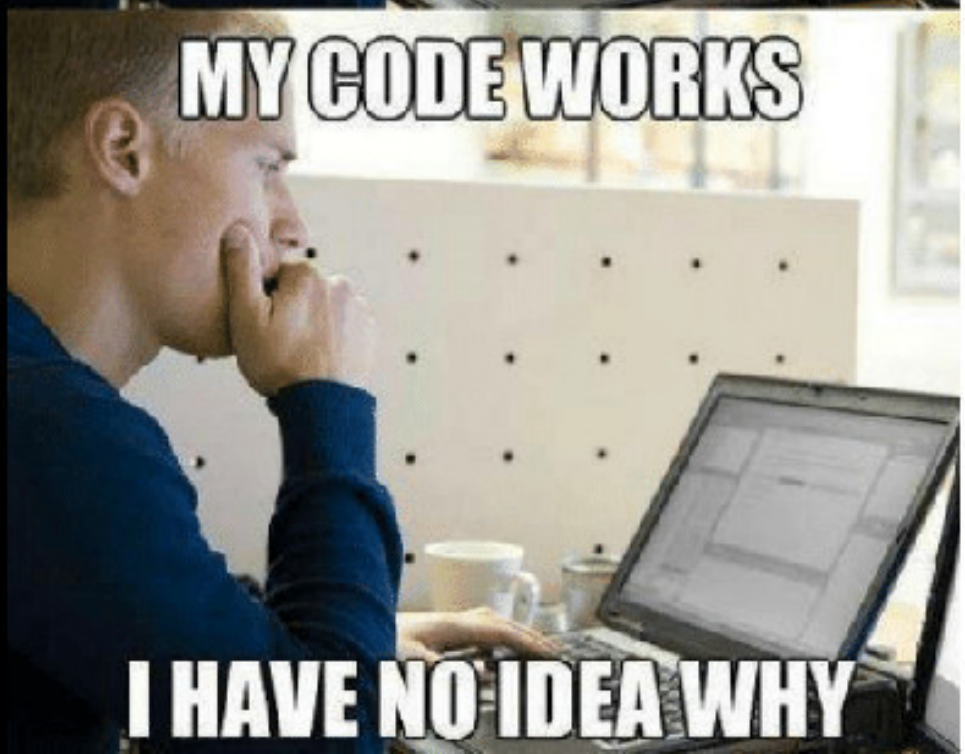

Какие планы?


Отладить прогу
на C++ с STL


memecenter.com


MY CODE DOESN'T WORK
I HAVE NO IDEA WHY


MY CODE WORKS
I HAVE NO IDEA WHY

ООП/C++: Лекция 8

Введение в STL (set, map), немного type deduction и RTTI

ака «Ещё полезные кубики и толстый слой синтаксического сахара»

О чём лекция сегодня

1. Вспоминаем STL и итераторы
2. Контейнеры STL (set, map)
3. Нюансы STL (и не только)
4. Выведение типов

Код на сегодня

https://github.com/avasyukov/oop-2nd-term/tree/master/2019/lection08

Контест к лекции

http://judge2.vdi.mipt.ru/cgi-bin/new-client?contest_id=911144

Вспоминаем STL и итераторы

Что такое STL

STL – библиотека, набор типовых примитивов, которым можно и нужно пользоваться. STL содержит:

- Контейнеры — примитивы для хранения наборов объектов
- Итераторы — средство доступа к данным в контейнерах произвольного вида
- Алгоритмы — наиболее типовые операции над данными в контейнерах
- Адаптеры — обёртки для обеспечения наиболее употребляемых интерфейсов (например, стек и очередь)
- Функторы — конструкции, позволяющие использовать объект как функцию (да, звучит непонятно)

Алгоритмы в составе STL

- В STL реализация алгоритмов отделена от реализации контейнеров данных (и это прекрасно) – в том числе для этого нужны итераторы
- Готовых типовых алгоритмов довольно много – мы сейчас смотрим только примеры, не «все» и даже не «самые главные»
- Если возникла конкретная «низкоуровневая» задача, которая кажется типовой – загляните в документацию, возможно, есть готовый «кубик»

Что такое итератор

Iterator – такая штука, которая немного похожа на указатель, но нужна для единообразного обхода произвольных структур, в том числе не плоских (дерева, хэш-таблицы и т.д.).

- Логически — сущность вида «служебный объект для обхода контейнера с данными»
- В простейшем случае технически равен указателю
- Для сложных контейнеров может быть реализован очень небанально внутри

Позволяет отделить алгоритм от структуры, в которой хранятся данные (затем и придуман)

Итератор: нюансы

- При изменении данных контейнера итератор может стать непригоден для использования, придётся получить новый.
- Бывают разные виды итераторов (обычный/const, normal/reverse).
- Проход контейнера в разных направлениях:
  - begin() – итератор на первый элемент
  - end() – итератор на позицию «после последнего»
  - rbegin() – итератор на последний элемент
  - rend() – итератор на позицию «перед первым»

iterator: примеры

Контрольно вспомним итератор:

- 01_hello_again_iterator.cpp

Контейнеры STL (set, map)

Контейнеры STL

- Последовательные:
  - вектор (vector)
  - двусвязный список (list)
  - дэк (deque)
- Ассоциативные:
  - множества (set и multiset)
  - хэш-таблицы (map и multimap)
- Псевдоконтейнеры:
  - битовые маски (bitset)
  - строки (string и wstring)
  - массивы (valarray)

Контейнер set и типовые функции

- Хранилище ключей:
  - Ключи – значения, для которых применима операция сравнения
  - Каждый ключ может встретиться в контейнере только один раз
  - Быстрый поиск по ключам (хранятся внутри контейнера в «правильном» виде для поиска)
- Применение – обработать потоки уникальных записей, выполнить типовые операции над множествами (пересечение, объединение, ...)

set: примеры

Смотрим пример:

- 02_set_example.cpp

Контейнер map и типовые функции

- Хранилище пар «ключ-значение»:
  - Ключи аналогичны set
  - Значения могут быть любые (операция сравнения для значений может быть не определена)
- Вариация map – ключ должен быть уникален, обращение по ключу возвращает строго одно значение (используется часто)
- Вариация multimap – ключ не обязан быть уникален, обращение по ключу возвращает набор элементов (используется редко)

map: примеры

Смотрим примеры:

- 03_map_example.cpp
- 04_multimap_example.cpp

Нюансы STL (и не только)

Нюансы STL (и не только)

Давайте посмотрим:

- Как логическая стройность системы типов (и отдельно STL) превращается в боль при написании конкретного кода
- Как эту боль можно полечить (и где границы применимости рецепта)

Нюансы STL

Пример из только что разобранного кода:

```
pair<
    multimap<int,string>::iterator,
    multimap<int,string>::iterator
> range = tags.equal_range(111);
```

Логически всё понятно. Но читать и писать такое немного больно.

Разные сюрпризы: примеры

Смотрим примеры:

- 05_nightmare_example.cpp
- 06_nightmare_example.cpp
- 07_nightmare_example.cpp

Идея

А давайте как-нибудь избавимся от таких страшных длинных типов?

C++11: А давайте!

Идея

А давайте как-нибудь избавимся от таких страшных длинных типов?

C++11: А давайте!

Выведение типов

Информация о типах в ходе работы

Смотрим пример:

- 08_hello_c++11_auto.cpp

Disclaimer: далее почти весь материал специфичен для C++11

Пара фич C++11

Ряд конструкций, которые не вносят ничего принципиально нового, но упрощают жизнь:

- auto – «пусть компилятор как-нибудь сам разберётся с этими страшными типами»
- range-based loop – «постараемся итераторы вообще спрятать куда-нибудь поглубже»

Внимание! Использование имеет ряд очень неочевидных ограничений – примеры в дополнительных материалах к лекции.

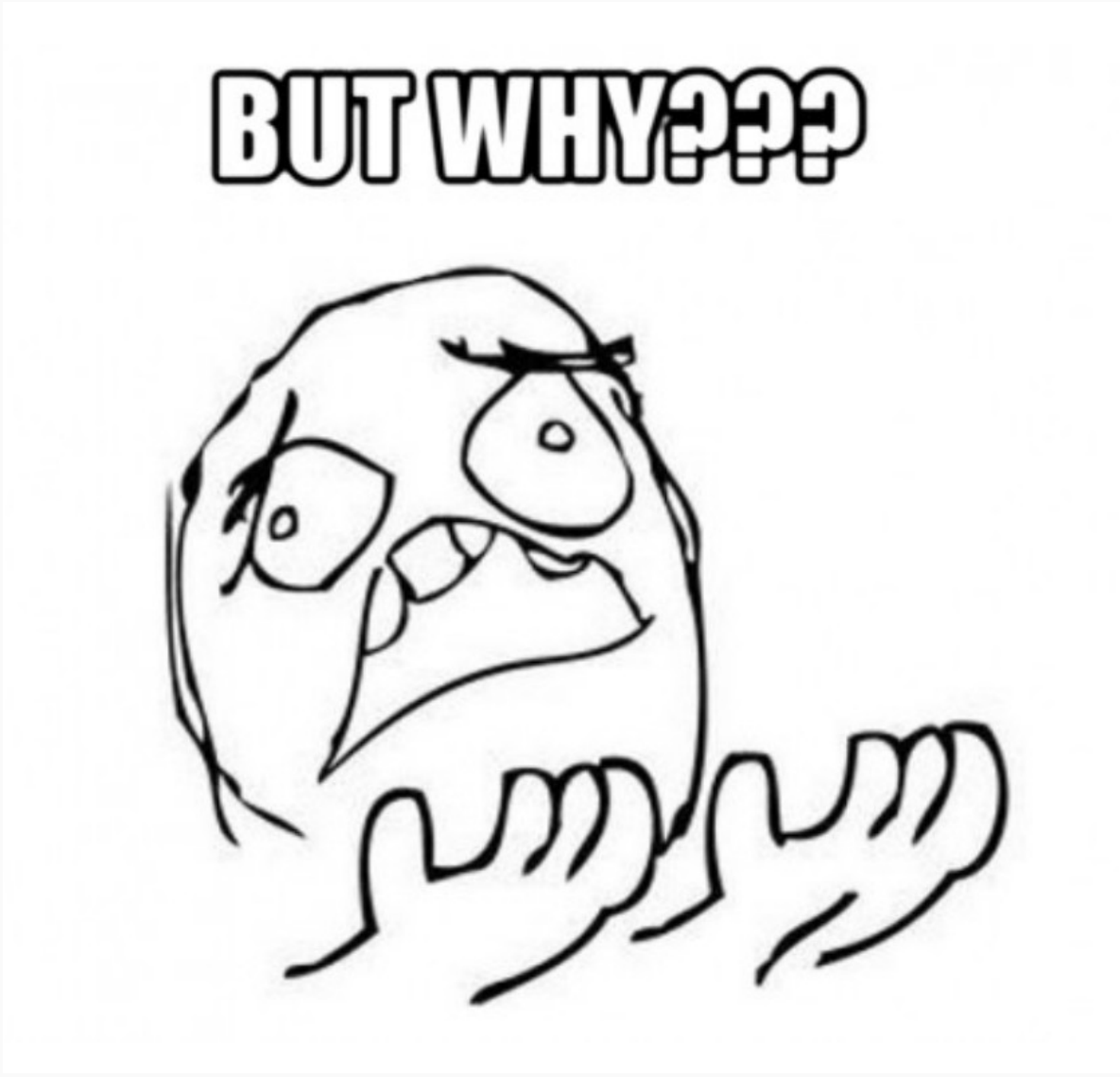
Пара фич C++11: примеры

Смотрим примеры:

- 09_syntax_with_auto.cpp
- 10_more_c++11.cpp
- 11_more_examples.cpp
- 12_auto_limitations.cpp

Пара фич C++11

Так зачем было рассказывать страшный синтаксис?!



Пара фич C++11

Чтобы не чувствовать себя глупо, когда что-то идёт не так, и компилятор выдаёт стену ошибок.

```
 2
 3
 4   Machine,
 5
 6   Pls make website,
 7
 8   all responsive like,
 9   w/ BIG pictures ooo,
10   use my fav fonts,
11   also fancy menus with whooosh on,
12   load fast pls
13
14   Thanks,
15   Human
16
17   PS no bugs :)
18
19
```

Холивар

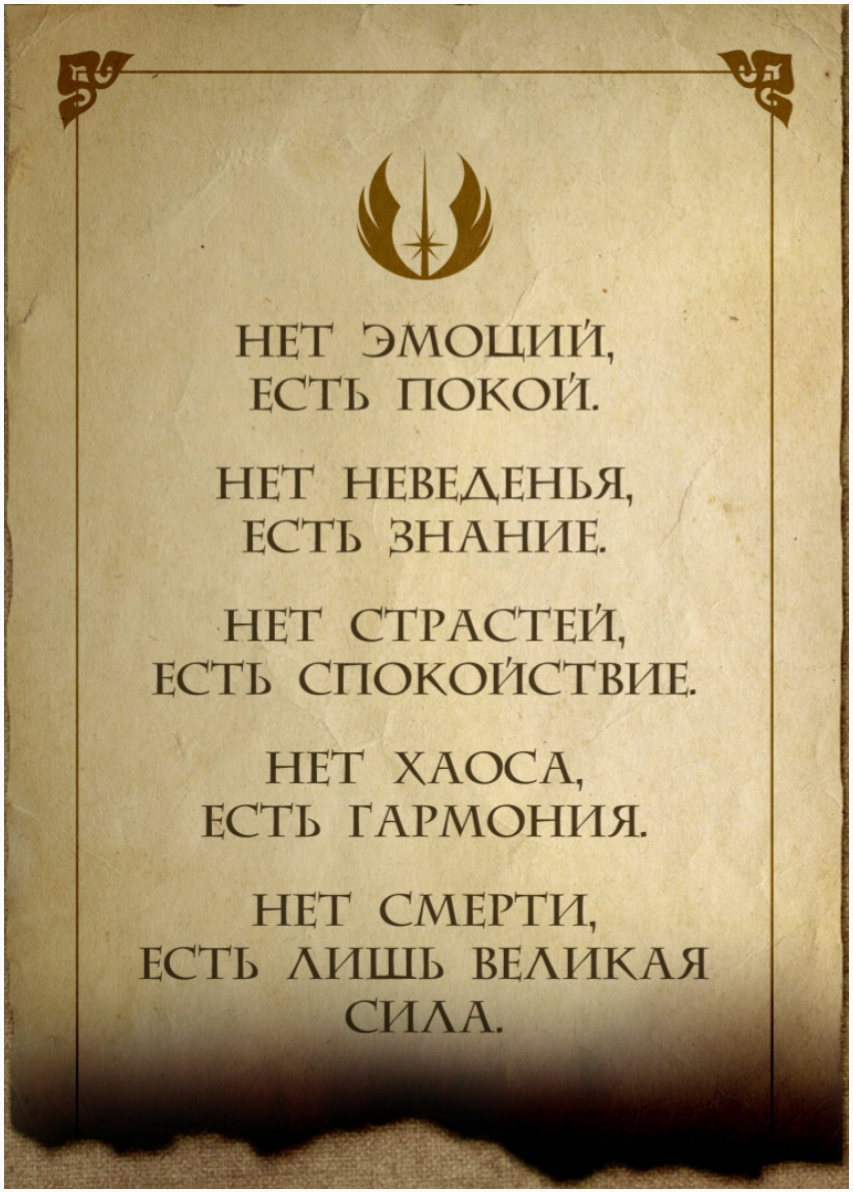
- Традиционный синтаксис портит читаемость кода, всё слишком длинно и неинтуитивно.
- Современный C++ должен быть удобным, auto обеспечивает это.

VS

- Использование auto портит читаемость кода, приходится гадать, какой там тип на самом деле.
- Использование auto портит чистоту C++ как строго типизированного языка.

Так что же правильно

- Слушайте внутреннего джедая
- Сохраняйте спокойствие
- Не поддавайтесь на холиварные провокации
- Использование auto удобно и хорошо, но оно не должно быть самоцелью



Что стоит запомнить из лекции

- В STL кроме массивов (array, vector) существуют удобные и полезные set и map. Плюс готовые функции для типовых задач над множествами.
- Итераторы таки придётся понять, без них почти все описанные вкусности недоступны.
- Начиная с C++11 появились auto и range-based loop-ы, но ими нужно пользоваться с пониманием, что это такое.
- Всё-таки не нужно использовать auto в параметрах и возвращаемых значениях функций.

Мнение о лекции

https://tinyurl.com/yxznwo6a


ARRGH! MY MAP OF LISTS OF MAPS TO STRINGS IS TOO HARD TO ITERATE THROUGH! I'LL JUST ASSIGN EVERYTHING A NUMBER AND USE A *!#*!@ ARRAY