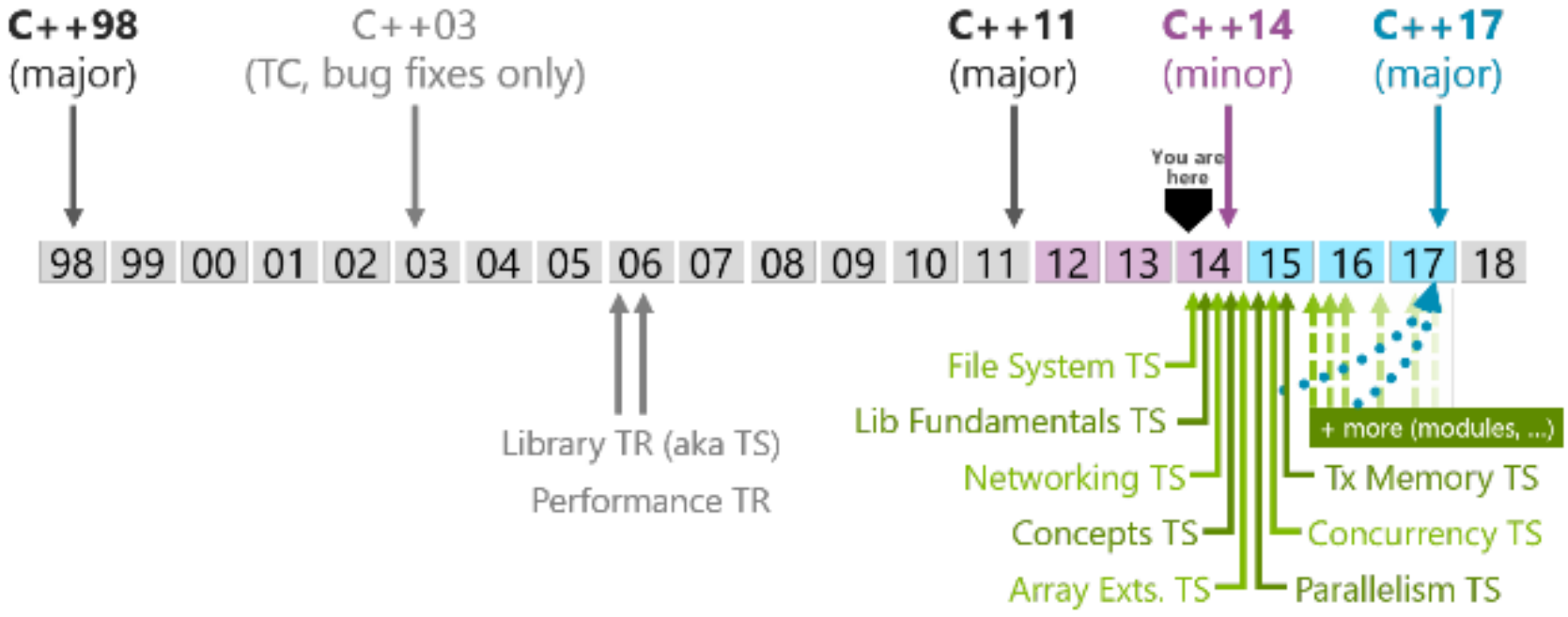
Стандарт C++ 11

Александр Смаль

Академический университет
17 апреля 2014
Санкт-Петербург

Стандарты C++


C++98
(major)
C++03
(TC, bug fixes only)
C++11
(major)
C++14
(minor)
C++17
(major)
You are here
98 99 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18
Library TR (aka TS)
Performance TR
File System TS
Lib Fundamentals TS
Networking TS
Concepts TS
Array Exts. TS
Tx Memory TS
Concurrency TS
Parallelism TS
+ more (modules, ...)

Стандарт 2011 года

- поддержка стабильности и обеспечение совместимости с C++98;
- предпочитается введение новых возможностей через стандартную библиотеку, а не через ядро языка;
- предпочитаются изменения, которые улучшают технику программирования;
- совершенствовать C++ с точки зрения системного и библиотечного дизайна, вместо введения новых возможностей, полезных для отдельных приложений;
- увеличивать типобезопасность для обеспечения безопасной альтернативы для нынешних опасных подходов;
- увеличивать производительность и возможности работать напрямую с аппаратной частью;
- обеспечивать решение реальных, широко распространённых проблем;
- реализовать принцип «не платить за то, что не используешь»;
- сделать C++ проще для изучения без удаления возможностей, используемых программистами-экспертами.

Rvalue Reference/Move semantics

Новый стандарт позволяет реализовывать семантику переноса.

```
template<class T> class vector
{
    vector (const vector & v)
        : size_(v.size()), ...
    {...}

    vector (vector && v) : size_(0), ... {
        v.swap(*this);
    }
    vector & operator = (const vector & v) {
        vector(v).swap(*this);
    }
    vector & operator = (vector && v) {
        v.swap(*this);
    }
};
```

std::forward

По умолчанию работает для временных объектов.

Для переменных нужно использовать std::move:

```
v.push_back(std::move(str));
```

v.push_back(str);

Простые типы вместо POD

Тривиальный класс — это класс, который:

1. содержит тривиальный конструктор по умолчанию и тривиальный деструктор.
2. не содержит нетривиальных копирующих конструкторов, перемещающих конструкторов, копирующих операторов присваивания, перемещающих операторов присваивания,

Класс со стандартным размещением — это класс, который:

1. не содержит нестатических членов-данных, имеющих тип класса с нестандартным размещением или ссылок.
2. не содержит виртуальных функций и виртуальных базовых классов,
3. имеет один и тот модификатор доступа для всех нестатических членов-данных,
4. не имеет базовых классов с нестандартным размещением,
5. не является классом, одновременно содержащим унаследованные и неунаследованные нестатические члены-данные, или содержащим нестатические члены-данные, унаследованные сразу от нескольких базовых классов,
6. не имеет базовых классов того же типа, что и у первого нестатического члена-данного (если таковой есть).

Константные выражения и внешние шаблоны

Константные выражения

```
constexpr int GiveFive() { return 5;}
int some_value[GiveFive() + 7];
```

```
constexpr double accelerationOfGravity = 9.8;
constexpr double moonGravity = accelerationOfGravity / 6;
```

Внешние шаблоны

```
//instantiate here
template class std::vector<MyClass>;

//don't instantiate here
extern template class std::vector<MyClass>;
```

11 2003

2011

Списки инициализации

```
// constructors
struct SequenceClass {
      SequenceClass(std::initializer_list<int> list);
};

SequenceClass someVar = {1, 4, 5, 6};

// functions
void FunctionName(std::initializer_list<float> list);

FunctionName({1.0f, -3.45f, -0.4f});

// containers
vector<string> v = { "xyzzy", "plugh", "abracadabra" };
vector<string> v{ "xyzzy", "plugh", "abracadabra" };
```

Объекты `std::initializer_list<>` могут быть созданы только статически с использованием синтаксиса со скобками {}, объекты являются неизменяемыми.

Универсальная инициализация

```
struct BasicStruct {
    int x;
    double y;
};

struct AltStruct {
    AltStruct(int x, double y) : x_(x), y_(y) {}

    int x_;
    double y_;
};

BasicStruct var1{5, 3.2};
AltStruct   var2{2, 4.3};

BasicStruct GetString() { return {6, 4.2}; }

std::vector<int> theVec{4}; // [4]
```

Вывод типов

```
// auto
for (vector<int>::const_iterator i = myvec.cbegin();
                                 i != myvec.cend(); ++i) {
    vector<int>::const_iterator j = i;
}

for (auto i = myvec.cbegin(); i != myvec.cend(); ++i) {
    decltype(i) j = i;
}

// comparison
  const std::vector<int> v(1);
  auto a = v[0];          // a - int
  decltype(v[0]) b = 1; // b - const int&
  auto c = 0;             // c - int
  auto d = c;             // d - int
  decltype(c) e;          // e - int
  decltype((c)) f = c;    // f - int&
  decltype(0) g;          // g - int
```

Тип переменной

тип выражения

Лямбда-выражения

```
// (\x y -> x + y)
[](int x, int y) { return x + y; }

[](int x, int y) -> int { int z = x + y; return z; }

vector<int> lst = {1,2,3,4,5};
int total = 0;
for_each(lst.begin(), lst.end(), [&total](int x) {
  total += x;
});

int value = 5;
[&total, value](int x) { total += (x * value); } (1);

auto lambdaFun = [this]() { this->privateMethod(); };
```

Различные виды замыканий: `[]`, `[x, &y]`, `[&]`, `[=]`, `[&, x]`, `[=, &z]`

Альтернативный синтаксис функций

```
// RETURN_TYPE = ?
template <typename A, typename B>
    RETURN_TYPE Plus(const A &a, const B &b)
{
    return a + b;
}
// long
template <typename A, typename B>
    decltype(std::declval<const A&>() + std::declval<const B&>())
        Plus(const A &a, const B &b)
{
    return a + b;
}
// wrong
template <typename A, typename B>
    decltype(a + b) Plus(const A &a, const B &b)
{
    return a + b;
}
// nice =)
template <typename A, typename B>
    auto Plus(const A &a, const B &b) -> decltype(a + b)
{
    return a + b;
}
```