Лекция 3. Беглый обзор стандарта C++17

ИУ8

September 22, 2016

if constexpr

В C++17 появилась возможность на этапе компиляции выполнять if.

```
template <std::size_t I, class F, class S>
auto rget(const std::pair<F, S>& p) {
    if constexpr (I == 0) {
        return p.second;
    } else {
        return p.first;
    }
}
```

Заметим, что разные ветки возвращают разные типы.

if (init; condition)

C++11

```
1  void foo() {
2      // ...
3      {
4          std::lock_guard<std::
               mutex> lock(m);
5          if (!container.empty()) {
6              // do something
7          }
8      } // the destruction of the
           lock_guard
9      // ...
10 }
```

C++17

```
1  void foo() {
2      // ...
3      if (std::lock_guard lock(m);
           !container.empty()) {
4          // do something
5      } else {
6      // var 'lock' is visible here
7      } // the destruction of the
           lock_guard
8      // ...
9  }
```

Автоматическое определение шаблонных параметров для классов

Простые шаблонные классы, конструктор которых явно использует шаблонный параметр, теперь автоматически определяют свой тип.

C++11

```
1 std::pair<int, double> p(17, 42.0);
2
3 std::lock_guard<std::shared_timed_mutex> lck(mut_);
4
5 std::unique_ptr<std::vector<int>> ptr(new std::vector<int>{1, 2, 3});
```

C++17

```
1 std::pair p(17, 42.0);
2
3 std::lock_guard lck(mut_);
4
5 std::unique_ptr ptr(new std::vector<int>{1, 2, 3});
```

Теперь нет необходимости делать make Функции для упрощения вывода типов

std::optianal<T>

Объекст типа std::optianal управляет опциональным значением, т.е. объект может содержать значение, а может и не содержать.

```
std::optional<int> getConfigParam(std::string name); // return either an
    int or a ‘not-an-int‘

int main()
{
  auto oi = getConfigParam("MaxValue");
    if (oi)     // did I get a real int?
      runWithMax(*oi); // use my int
    else
      runWithNoMax();
}
```

```
struct S {
  std::optional<std::string> _data;
  std::string getData() {
    if(!_data)
      _data.emplace(readFromFile());
    return *_data;
  }
};
```

std::variant<T...>

The variant class template is a safe, generic, stack-based discriminated union container, offering a simple solution for manipulating an object from a heterogeneous set of types in a uniform manner.

Класс std::variant представляет из себя объединение безопасное с точки зрения типов. std::variant может содержать один из альтернативных типов, объявленных при определении, или может не содержать значения вовсе (valueless_by_exception).

Дизайн основан на boost::variant, но при этом убраны все известные недочёты последнего:

- std::variant никогда не аллоцирует память для собственных нужд;
- множество методов std::variant являются constexpr, так что его можно использовать в constexpr выражениях;
- std::variant умеет делать emplace;
- к хранимому значению можно обращаться по индексу или по типу — кому как больше нравится;
- std::variant не умеет рекурсивно держать в себе себя

Пример

```
std::variant<int, std::string> v;
v = "Hello word";
assert(std::get<std::string>(v) == "Hello word");
v = 17 * 42;
assert(std::get<0>(v) == 17 * 42);
```

```
int main() {
    std::variant<int, float> v;
    v = 12; // v contains int
    int i = std::get<int>(v);

// std::get<double>(v); // error: no double in [int, float]
// std::get<3>(v); // error: valid index values are 0 and 1

    try {
      std::get<float>(v); // w contains int, not float: will throw
    } catch (std::bad_variant_access&) {}

    std::variant<std::string> v("abc"); // converting constructors work
        when unambiguous
    v = "def"; // converting assignment also works when unambiguous
}
```

std::any

Объекты типа std::any можно использовать для выполнения операций над объектами различных типов.

Функция any_cast<T> обеспечивает доступ к объекту, содержащийся в std::any.

Методы std::any

- **operator=** присвоение любого значения
- **emplace** изменение хранимого значения, конструирование происходит непосредственно в emplace
- **reset** разрушается хранимый в std::any объект
- **swap** swap он и в Африке swap
- **has_value** проверка, хранится ли объект в std::any
- **type** возращает typeid хранимого объекта

```cpp
any x(5); // x holds int
assert(any_cast<int>(x) == 5); // cast to value
any_cast<int&>(x) = 10; // cast to reference
assert(any_cast<int>(x) == 10);

x = string("Meow"); // x holds string
string s, s2("Jane");
s = move(any_cast<string&>(x)); // move from any
assert(s == "Meow");
any_cast<string&>(x) = move(s2); // move to any
```

std::string_view

std::string_view — это класс, не владеющий строкой, но хранящий указатель на начало строки и её размер.

```cpp
#include <string>
// allocate memory, if pass big array of char
void get_vendor_from_id(const std::string& id) {

    // allocate memory to creating substring
    std::cout << id.substr(0, id.find_last_of(':'));
}
```

```cpp
#include <string_view>

// doesn't allocate memory
// working with 'const char*', 'char*', 'const std::string&', etc.
void get_vendor_from_id(std::string_view id) {

    // doesn't allocate memory to creating substring
    std::cout << id.substr(0, id.find_last_of(':'));
}
```

- используйте единственную функцию, принимающую string_view, вместо перегруженных функций, принимающих const std::string&, const char* и т.д.;
- передавайте string_view по копии (нет необходимости писать 'const string_view& id')

Изменение в std::string

Появился метод std::string::data() возвращающий неконстантный указатель **char ***

Многопоточные алгоритмы

Большинство алгоритмов были продублированы в виде версий, принимающих параметр ExecutionPolicy. Теперь можно выполнять алгоритмы многопоточно.

```
std::vector<int> v;
v.reserve(100500 * 1024);
some_function_that_fills_vector(v);

// multi-threaded sorting
std::sort(std::execution::par, v.begin(), v.end());
```

Если внутри алгоритма, принимающего ExecutionPolicy, вы кидаете исключение и не ловите его, то программа завершится с вызовом std::terminate()

Доступ к нодам контейнера

В C++17 многие контейнеры обзавелись возможностью передавать свои внутренние структуры для хранения данных наружу, обмениваться ими друг с другом без дополнительных копирований и аллокаций.

Таким образом в C++17 будет существовать возможность реализовывать алгоритмы:

- более производительными — за счёт уменьшения количества динамических аллокаций и уменьшения времени, которое программа проводит в критической секции;
- более безопасными — за счёт уменьшения количества мест, кидающих исключения, и за счет меньшего количества аллокаций;
- менее требовательными к памяти.

T& container::emplace_back(Args&&...)

Метод emplace_back возвращает ссылку на созданный объект

C++11

```
1 some_vector.emplace_back();
2 some_vector
3   .back()
4   .do_something();
```

C++17

```
1 some_vector
2   .emplace_back()
3   .do_something();
```

Структурное связывание

Structured bindings

```
std::pair<bool, string> getNameDevice();

auto foo() {
  auto [ok, info] = getNameDevice();
  if(!ok)
    return _defaultName;
  return info;
}
```

Структурное связывание работает не только с std::pair или std::tuple, а с любыми структурами

```
struct my_struct { std::string s; int i; };
my_struct my_function() {
  return my_struct{ "some string", 42};
}
// ...

auto [str, integer] = my_function();
assert(str == "some string");
assert(integer == 42);
```

Ура, товарищи!

- Класс std::path
- Работа с файлами (удаление, копирование, проверка на существование и т.д.)
- Работа с различными типами файлов (symlink, socket, hard link, directory, etc)
- Класс std::directory_iterator для обхода директории
- Определение прав доступа
- др.

Все перечисленное и многое другое

В 2017 во всех компиляторах страны

The end

Самостоятельное изучение

- constexpr лямбды
- примеры работы с файлами
- реализовать std::any
- реализовать std::optional
- реализовать std::variant

Список литературы

- Working draft C++17
- Антон Полухин. C++17
- Последние новости о развитии C++
- Boost
- CppCon 2016