Программирование на языке C++
Лекция 8

Ещё о нововведениях C++11/C++14

Александр Смаль

Кортежи

```
std::tuple<std::string, int, int> getUnitInfo(int id) {
    if (id == 0) return std::make_tuple("Elf",   60, 9);
    if (id == 1) return std::make_tuple("Dwarf", 80, 6);
    if (id == 2) return std::make_tuple("Orc",   90, 3);
    //...
}
int main() {
    auto ui0 = getUnitInfo(0);
    std::cout << "race: "<< std::get<0>(ui0) << ", "
              << "hp: "  << std::get<1>(ui0) << ", "
              << "iq: "  << std::get<2>(ui0) << "\n";

    std::string race1; int hp1; int iq1;
    std::tie(race1, hp1, iq1) = getUnitInfo(1);
    std::cout << "race: " << race1 << ", "
              << "hp:   " << hp1   << ", "
              << "iq:   " << iq1   << "\n";
}
```

Константные выражения

Для констант и функций времени компиляции.

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

constexpr int pow(int x, int k)
{ return k == 0 ? 1 : x * pow(x, k - 1); }

int data[pow(3, 5)] = {};
```

```
struct Point {
   double x, y;
   constexpr Point(double x = 0, double y = 0)
       : x(x), y(y) {}
   constexpr double getX() const { return x; }
   constexpr double getY() const { return y; }
};
constexpr Point p(moonGravity, accOfGravity);
constexpr auto x = p.getX();
```

Range-based for

Синтаксическая конструкция для работы с последовательностями.

```
int array[] = {1, 4, 9, 16, 25, 36, 49};

int sum = 0;
// по значению
for (int x : array) {
    sum += x;
}

// по ссылке
for (int & x : array) {
    x *= 2;
}
```

Применим к встроенным массивам, спискам инициализации, контейнерам из стандартной библиотеки и любым другим типам, для которых определены функции `begin()` и `end()`, возвращающие итераторы (об этом будет рассказано дальше).

Списки инициализации

Возможность передать в функцию список значений.

```
// в конструкторах массивов и других контейнеров
template<typename T>
struct Array {
    Array(std::initializer_list<T> list);
};

Array<int> primes = {2, 3, 5, 7, 11, 13, 17};
```

```
// в обычных функциях
int sum(std::initializer_list<int> list) {
    int result = 0;
    for (int x : list)
        result += x;
    return result;
}

int s = sum({1, 1, 2, 3, 5, 8, 13, 21});
```

Универсальная инициализация

```
struct CStyleStruct {
    int x;
    double y;
};
struct CPPStyleStruct {
    CPPStyleStruct(int x, double y): x(x), y(y) {}
    int x;
    double y;
};
```

```
// C++03
CStyleStruct    s1 = {19, 72.0};// инициализация
CPPStyleStruct  s2(19, 83.0);   // вызов конструктора
```

```
// C++11
CStyleStruct    s1{19, 72.0}; // инициализация
CPPStyleStruct  s2{19, 83.0}; // вызов конструктора
```

```
// тип не обязателен
CStyleStruct getValue() { return {6, 4.2}; }
```

std::function

Универсальный класс для хранения указателей на функции, указателей на методы и функциональных объектов.

```
int mult (int x, int y) { return x * y; }

struct IntDiv {
    int operator()(int x, int y) const {
        return x / y;
    }
};
```

```
std::function<int (int, int)> op;
if ( OP == '*' )
    op = &mult;
else if ( OP == '/')
    op = IntDiv();
int result = op(7,8);
```

Позволяет работать и с указателями на методы.

Лямбда-выражения

```
std::function<int (int, int)> op =
    [](int x, int y) { return x / y; } // IntDiv

// то же, но с указанием типа возвращаемого значения
op = [](int x, int y) -> int { return x / y; }

// C++14
op = [](auto x, auto y) { return x / y; }
```

Можно захватывать *локальные* переменные.

```
// захват по ссылке
int total = 0;
auto addToTotal = [&total](int x) { total += x; };

// захват по значению
auto subTotal = [total](int & x) { x -= total ; };

// Можно захватывать this
auto callUpdate = [this](){ this->update(); };
```

Различные виды захвата

Могут быть разные типы захвата, в т.ч. смешанные:
`[]`, `[x, &y]`, `[&]`, `[=]`, `[&, x]`, `[=, &z]`

Перемещающий захват `[x = std::move(y)]` (только в C++14).

Не стоит использовать захват по умолчанию `[&]` или `[=]`.

```
std::function<bool(int)> getFilter(Checker const& c) {
    auto d  = c.getModulo();
    // захватывает ссылку на локальную переменную
    return [&] (int i) { return i % d == 0; }
}
```

```
struct Checker {
    std::function<bool(int)> getFilter() const {
        // захватывает this, а не d
        return [=] (int x) { return x % d == 0; }
    }
    int d;
};
```

Новые строковые литералы

```
u8"I'm a UTF-8 string."           // char[]
u"This is a UTF-16 string."       // char_16_t[]
U"This is a UTF-32 string."       // char_32_t[]
L"This is a wide-char string."    // wchar_t[]

u8"This is a Unicode Character: \u2018."
u"This is a bigger Unicode Character: \u2018."
U"This is a Unicode Character: \U00002018."

R"(The String Data \ Stuff " )"
R"delimiter(The String Data \ Stuff " )delimiter"

LR"(Raw wide string literal \t (without a tab))"
u8R"XXX(I'm a "raw UTF-8" string.)XXX"
uR"*(This is a "raw UTF-16" string.)*"
UR"(This is a "raw UTF-32" string.)"
```