Rvalue-reference в D

Речь сегодня пойдёт о передаче в функцию rvalue по ссылке. Сравнивать будем C++ и D.

Как правило, при передаче в функцию структуры желательно избегать лишнего копирования, так как это сказывается на производительности не в лучшую сторону. Для этого используется передача по ссылке. В случае C++

```
#include <iostream>
#include <cmath>

using namespace std;

struct vec {
  double x, y, z;
};

double length(vec& v) {
  return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}

int main() {
  vec a = {3, 4, 12};
  cout << length(a) << endl;
  return 0;
}
```

Для D всё выглядит очень похоже:

```
import std.stdio;
import std.math;
```

```
struct vec {
  double x, y, z;
};

double length(ref vec v) {
  return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}

void main() {
  vec a = {3, 4, 12};
  writeln(length(a));
}
```

Но что, если мы не захотим создавать переменную `a`, а попробуем передать в функцию в качестве аргумента конструктор:

```
length(vec(3,4,12))
```

В этом случае оба компилятора будут ругаться на то, что они не могут получить неконстантную ссылку на rvalue, коим является результат вызова конструктора. Функция `length` не изменяет своего аргумента, поэтому слегка её изменим, сделав аргумент константной ссылкой:

```
double length(const vec& v) {
  return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}
```

На D это выглядит так:

```
double length(ref const vec v) {
  return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}
```

В C++ это решает проблему, а в D нет – он считает, что реальный тип аргумента vec, а ожидаемый `ref const vec`. Дело в том, что D воспринимает rvalue только как значение, но при передаче в функцию лишнего копирования не происходит. Поэтому от нас требуется определить такую функцию, которая сможет одновременно принимать и `ref const vec`, и просто `vec`. И тут не обойтись без шаблона:

```
double length(T: vec)(auto ref const T v) {
  return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}
```

Собирая всё в кучу, имеем

```
import std.stdio;
import std.math;

struct vec {
  double x, y, z;
};

double length(T: vec)(auto ref const T v) {
  return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}

void main() {
  vec a = {3, 4, 12};
  writeln(length(a));
}
```

Можно заметить, что проблемы можно избежать, определив `length` как метод `vec`:

```
import std.stdio;
import std.math;

struct vec {
  double x, y, z;

  double length() {
    return sqrt(this.x * this.x + this.y * this.y + this.z * this.z);
  }
};

void main() {
  vec a = {3, 4, 12};
  writeln(length(a));
```

```
4

}
```

Но если нам понадобится скалярное произведение, то всё равно придётся писать такую вот шаблонную функцию.

В общем, если в D вам потребуется аналог `const T&`, то это `auto ref const T`.