Метапрограммирование на C++

12 июля 2022 г.

Оглавление

Глава 1

Введение в шаблоны

1.1. Базовые понятия

Шаблоном, согласно стандарту, называют отрывок кода специального вида[1], задающий семейство классов, функций или переменных, алиас[2] на семейство типов, либо концепт. Прочитав это определение представить в голове что такое шаблон решительно невозможно, поэтому постараемся распутать его, пожертвовав в процессе точностью в угоду простоте, и выработать ментальную модель, пригодную для использования при написании кода.

Итак, не отступая далеко от «официального» определения, шаблоном мы действительно будем называть отрывок кода, состоящий из двух частей: заголовка и тела. Заголовок обязан выглядеть как `template< ... >`, где на месте `...` перечисляются через запятую так называемые *аргументы шаблона*, подробнее о которых будет ниже. Тело же обязано быть *определением* или *объявлением* некоторой сущности языка C++, например класса, структуры, функции или глобальной переменной. Незамедлительно перейдём к синтаксическим примерам, не обременяя себя семантикой происходящего:

```
// Шаблон структуры от двух аргументов-типов
template<class T, class U>
```

[1] Вид этот определён сложной к прочтению формальной грамматикой.

[2] Алиасом называют имя, заданное посредством ключевого слова `using`.

```
struct Pair
{
    T first;
    U second;
};

// Шаблон структуры от одного аргумента-не-типа
template<int Size>
struct ArrayOfBools
{
    bool values[Size];
};

// Шаблон алиаса
template<class U>
using SamePair = Pair<U, U>;

// Шаблон объявления функции
template<class T>
T square(T value);

// Шаблон определения функции
template<class T>
T square(T value)
{
    return value*value;
};
```

Из примера видно, что в качестве аргумента шаблона может выступать как типы, так и значения (но некоторыми ограничениями). В первом случае используется синтаксис `class ArgumentName` или эквивалентный ему `typename ArgumentName`. Во втором случае синтаксис аналогичен обычным функциям.

Теперь, имея представление о том, как выглядят шаблоны в коде, начнём разбирать их семантику. Для этого нам потребуется переосмыслить понимание системы типов C++.

Что есть тип? Согласно теории типов, это не что иное, как *множество его возможных значений*[3]. Таким образом тип `bool` не отличим от множества $\{\text{true}, \text{false}\}$, а тип `uint32_t` не отличим от $[0, 2^{32}) \cap \mathbb{Z}$. Структуры и

[3] Это не правда

кортежи же не отличимы от декартовых произведений, например

```
struct Person
{
    uint32_t age;
    float height;
};
```

эквивалентно множеству uint32_t $\times$ float. Аналогичное справедливо про `std::pair` и `std::tuple`. Двойственная к произведению конструкция – непересекающееся объединение, обозначаемое как $A \sqcup B$, которому эквивалентны `union` и `std::variant`.

Функции языка C++ в нашей интерпретации эквивалентны обычным математическим функциям, отображающим значения одного типа в значения другого типа. Однако так мы ограничиваемся рассмотрением исключительно *чистых* функций, то есть возвращающих один и тот же результат при одинаковых входных данных. Чистыми не являются, например, функции `std::scanf` и `std::rand`.

Наконец рассмотрим множество Type, содержащее в себе все возможные типы. Тогда шаблон класса/структуры является частичной функцией $Type \to Type$.