**212 – Назаров – Захар**

**ЭССЕ**

**Teamplate**

# Шаблоны (C++)

Шаблоны являются основой для универсального программирования на C++. Как строго типизированный язык C++ требует, чтобы все переменные имели определенный тип, явно объявленный программистом или выведенный компилятором. Однако многие структуры данных и алгоритмы выглядят одинаково независимо от типа, с каким типом они работают. Шаблоны позволяют определять операции класса или функции и позволяют пользователю указать, с какими конкретными типами должны работать эти операции.

## Определение и использование шаблонов

Шаблон — это конструкция, которая создает обычный тип или функцию во время компиляции на основе аргументов, которые пользователь предоставляет для параметров шаблона. Параметру типа можно присвоить любое имя, но по соглашению чаще всего используются отдельные прописные буквы. *T* — параметр шаблона; в **typename** ключевое слово указано, что этот параметр является заполнителем для типа. При вызове функции компилятор заменяет каждый экземпляр на конкретный T аргумент типа, указанный пользователем или выведенный компилятором. Процесс, в котором компилятор создает класс или функцию из шаблона, называется *создание экземпляра шаблона*; minimum<int> — это экземпляр шаблона minimum<T>. В другом месте пользователь может объявить экземпляр шаблона, специализированный для int.

Правила того, как компилятор выполняет выведение типов в шаблонах функций, основаны на правилах для обычных функций.

## Параметры типа

Обратите внимание, что в приведенном minimum выше шаблоне параметр типа *T* не является квалифицированным, пока он не будет использоваться в параметрах вызова функции, где добавляются квалификаторы const и ссылок.

Количество параметров типа практически не ограничено. Разделяйте несколько параметров запятыми. В этом контексте ключевое слово **class** эквивалентно **typename**. Оператор с многоточием (...) можно использовать для определения шаблона, который принимает произвольное число параметров типа (ноль или больше). В качестве аргумента типа можно использовать любой встроенный или определяемый пользователем тип. Например, можно использовать std::vector в стандартной библиотеке для хранения переменных типа **int**, **double**, std::string, MyClass, **const**MyClass*, MyClass&и т. д.

Основное ограничение при использовании шаблонов заключается в том, что аргумент типа должен поддерживать любые операции, применяемые к параметрам типа. Не существует неотъемлемого требования к тому, чтобы аргументы типа для любого конкретного шаблона принадлежали к одной и той же иерархии объектов, хотя можно определить шаблон, который применяет такое ограничение. Объектно-ориентированные методы можно сочетать с шаблонами. Обратите внимание, что аргументы должны быть указателями.

Основные требования, которые std::vector и другие контейнеры стандартных библиотек накладывают на элементы , — это T возможности копирования T и назначения и создания с помощью копирования.

## Параметры, не относящиеся к типу

В отличие от универсальных типов в других языках, таких как C# и Java, шаблоны C++ поддерживают *нетиповые параметры*, также называемые параметрами значений. Обратите внимание на синтаксис в объявлении шаблона. Значение size_t передается в качестве

аргумента шаблона во время компиляции и должно быть **const** или выражением **constexpr**.

Другие типы значений, включая указатели и ссылки, можно передавать в качестве нетиповых параметров. Например, можно передать указатель на функцию или объект функции, чтобы настроить определенную операцию в коде шаблона.

## Вычет типов для параметров шаблона, не относящихся к типу

В Visual Studio 2017 и более поздних версиях, а также в /std:c++17 режиме или более поздней версии компилятор выводит тип аргумента шаблона, не являющегося типом, объявленного с auto помощью.

## Шаблоны в качестве параметров шаблона

Шаблон может быть параметром шаблона. В этом примере MyClass2 имеет два параметра шаблона: параметр typename *T* и параметр шаблона *Arr*.

Так как сам параметр *Arr* не имеет основного текста, имена его параметров не требуются. По этой причине имена параметров типа *Arr* можно опустить.

## Аргументы шаблона по умолчанию

Шаблоны классов и функций могут иметь аргументы по умолчанию. Если в шаблоне есть аргумент по умолчанию, его можно оставить неуказанным при его использовании.

В большинстве случаев приемлем класс std::allocator по умолчанию, поэтому используйте вектор следующим образом:

vector<int> myInts;

Но при необходимости можно указать пользовательский распределитель следующим образом:

vector<int, MyAllocator> ints;

При наличии нескольких аргументов шаблона все аргументы после первого аргумента по умолчанию должны иметь аргументы по умолчанию.

При использовании шаблона, параметры которого используются по умолчанию, используйте пустые угловые скобки.

## Специализация шаблонов

В некоторых случаях шаблон не может или не рекомендуется определять точно такой же код для любого типа. Например, можно определить путь к коду, который будет выполняться, только если аргумент типа является указателем, std::wstring или типом, производным от определенного базового класса. В таких случаях можно определить *специализацию* шаблона для конкретного типа. Когда пользователь создает экземпляр шаблона с этим типом, компилятор использует специализацию для создания класса, а для всех остальных типов компилятор выбирает более общий шаблон. Специализации, в которых все параметры являются специализированными, являются *полными специализациями*. Если только некоторые параметры являются специализированными, это называется *частичной специализацией*.

Шаблон может иметь любое количество специализаций, если каждый параметр специализированного типа уникален. Только шаблоны классов могут быть частично специализированными. Все полные и частичные специализации шаблона должны быть объявлены в том же пространстве имен, что и исходный шаблон.