

## Лекция 12. Шаблоны (Templates)

Функции-шаблоны, классы-шаблоны, обобщённое программирование

---

### 1. Введение

Одним из самых мощных механизмов языка C++ являются **шаблоны (templates)**. Они позволяют создавать обобщённый код, который может работать с различными типами данных без дублирования.

Шаблоны стали основой для создания **стандартной библиотеки шаблонов (STL)**, где контейнеры, итераторы и алгоритмы реализованы универсально — независимо от конкретного типа данных.

---

### 2. Идея обобщённого программирования

**Обобщённое программирование (Generic Programming)** — это парадигма, которая фокусируется на создании алгоритмов и структур данных, независимых от конкретных типов.

Например, алгоритм сортировки должен работать одинаково как с `int`, так и с `double`, `string` или даже пользовательскими типами.

#### Преимущества:

- уменьшение дублирования кода;
  - повышение гибкости;
  - повышение производительности (благодаря генерации конкретного кода при компиляции);
  - упрощение поддержки и расширения программ.
- 

### 3. Функции-шаблоны

Функция-шаблон — это шаблон, из которого компилятор создаёт конкретные функции для разных типов данных.

#### Синтаксис:

```
template <typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}
```

#### Использование:

```
int main() {
    cout << max(5, 10) << endl;    // работает с int
    cout << max(3.5, 2.8) << endl; // работает с double
    cout << max('A', 'Z') << endl; // работает с char
}
```

При вызове `max(5, 10)` компилятор создаёт версию функции `int max(int, int)`.

---

## 4. Параметры шаблонов

Шаблон может иметь несколько параметров:

```
template <typename T, typename U>
void printPair(T a, U b) {
    cout << a << " и " << b << endl;
}
```

**Пример:**

```
printPair(5, "пример"); // T = int, U = const char*
```

---

## 5. Классы-шаблоны

Шаблоны могут применяться не только к функциям, но и к классам. Это особенно полезно для создания контейнеров данных (например, `vector`, `stack`, `map` и т.д.).

**Пример класса-шаблона:**

```
template <typename T>
class Box {
private:
    T value;
public:
    Box(T v) : value(v) {}
    void set(T v) { value = v; }
    T get() const { return value; }
};
```

**Использование:**

```
int main() {
    Box<int> intBox(123);
}
```

```
Box<string> strBox("Привет");

cout << intBox.get() << endl;
cout << strBox.get() << endl;
}
```

---

## 6. Частичная специализация шаблонов

Иногда необходимо изменить поведение шаблона для конкретного типа. Это называется **специализацией шаблонов**.

**Пример полной специализации:**

```
template <>
class Box<string> {
public:
    string value;
    Box(string v) : value(v) {}
    void print() { cout << "Строка: " << value << endl; }
};
```

Теперь при создании Box<string> будет использоваться специализированная версия.

---

## 7. Шаблоны функций-членов

Класс может содержать шаблонные методы, независимо от того, является ли сам класс шаблоном:

```
class Printer {
public:
    template <typename T>
    void print(T value) {
        cout << value << endl;
    }
};
```

---

## 8. Нестандартные параметры шаблонов

Кроме типов, в шаблоны можно передавать **значения**, такие как целые числа:

```
template <typename T, int size>
```

```
class Array {
private:
    T data[size];
public:
    void fill(T value) {
        for (int i = 0; i < size; i++)
            data[i] = value;
    }
};
```

### Пример:

```
Array<int, 5> arr;
arr.fill(10);
```

---

## 9. Автоматическое выведение типов

В некоторых случаях компилятор сам может определить тип шаблона:

```
auto result = max(3.14, 2.71); // T автоматически выведен как double
```

---

## 10. Обобщённое программирование в STL

Шаблоны — основа стандартной библиотеки C++.

Благодаря им возможно создание универсальных структур данных:

```
#include <vector>
#include <algorithm>
```

```
vector<int> nums = {1, 2, 3, 4};
sort(nums.begin(), nums.end()); // шаблон функции sort
```

Здесь `sort()` и `vector<>` — шаблоны, которые адаптируются к типу `int`.

---

## 11. Ограничения шаблонов и понятие "концепты" (C++20)

До стандарта C++20 шаблоны не проверялись до момента их инстанцирования, что делало ошибки сложными для отладки.

С выходом **C++20** появились **concepts** — способ ограничить типы, с которыми может работать шаблон.

### Пример:

```
template <typename T>
concept Number = std::is_arithmetic_v<T>;
```

```
template <Number T>
T add(T a, T b) {
    return a + b;
}
```

Теперь шаблон можно использовать только с числовыми типами.

---

## 12. Заключение

Шаблоны — это сердце C++.

Они позволяют писать эффективный, гибкий и обобщённый код, который компилируется в высокопроизводительные машинные инструкции.

Именно благодаря шаблонам C++ остаётся языком, сочетающим **универсальность, скорость и типовую безопасность**.

---

## 13. Вопросы для самопроверки

1. Что такое шаблон в C++?
  2. Чем отличаются функции-шаблоны и классы-шаблоны?
  3. Для чего используется специализация шаблонов?
  4. Как шаблоны связаны с STL?
  5. Что такое концепты в C++20?
- 

## 14. Рекомендуемая литература

1. Бьерн Страуструп. *Язык программирования C++ (4-е издание)*.
2. Герб Саттер, Андрей Александреску. *Современное проектирование на C++*.
3. Scott Meyers. *Effective Modern C++*.
4. Nicolai M. Josuttis. *The C++ Standard Library: A Tutorial and Reference*.
5. ISO/IEC 14882:2020 — *The C++20 Standard*.