# Шаблоны функций — конспект темы

## Обобщаем функцию

Функция `ComputeTermFreqs` вычисляет частоту строк по данному вектору. Посчитать частоту можно и для числовых данных.

Чтобы узнать количество двуногих и четвероногих обитателей квартиры, напишите две функции с одинаковым названием для типа `int`. По типам аргументов компилятор поймёт, какую функцию вызвать:

```
map<string, int> ComputeTermFreqs(const vector<string>& terms) {
    map<string, int> term_freqs;
    for (const string& term : terms) {
        ++term_freqs[term];
    }
    return term_freqs;
}

map<int, int> ComputeTermFreqs(const vector<int>& terms) {
    map<int, int> term_freqs;
    for (int term : terms) {
        ++term_freqs[term];
    }
    return term_freqs;
}

int main() {
    const vector<int> leg_counts = {4, 2, 4, 4};
    const auto legs_stat = ComputeTermFreqs(leg_counts);
    cout << "Двуногих "s << legs_stat.at(2) << ", "s
         << "четвероногих "s << legs_stat.at(4) << endl;
    // Двуногих 1, четвероногих 3
}
```

Это **перегрузка функций**. Такой копипаст нежелателен. Перебор циклом `for` по значению можно убрать — серьёзно функция не замедлится.

Отличие останется только в типе. Пока неизвестно, с каким типом для слов функция будет работать, но её уже можно написать:

```
// нам интересны Term = string и Term = int, пытаемся обобщить
map<Term, int> ComputeTermFreqs(const vector<Term>& terms) {
    map<Term, int> term_freqs;
    for (const Term& term : terms) {
        ++term_freqs[term];
    }
    return term_freqs;
}
```

Функция зависит не только от конкретного вектора `terms`, но и от типа его элементов. Такая функция называется **шаблонной**:

```
template <typename Term>  // шаблонный параметр-тип с названием Term
map<Term, int> ComputeTermFreqs(const vector<Term>& terms) {
    map<Term, int> term_freqs;
    for (const Term& term : terms) {
        ++term_freqs[term];
    }
    return term_freqs;
}
```

## Как устроены шаблоны

Свойства шаблонных функций:

- `ComputeTermFreqs<int>`, `ComputeTermFreqs<string>` и функции с любыми другими `Term` в угловых скобках — это разные функции. Компилятор копирует их, подставляя нужный тип вместо `Term`. Конкретная `ComputeTermFreqs<Animal>` может не скомпилироваться, но по умолчанию требований к типу нет.
- При вызове шаблонной функции можно указать в угловых скобках значение её шаблонного параметра. А можно не указывать — тогда компилятор постарается вывести шаблонные параметры из типов аргументов.

## Универсальные функции вывода контейнеров

Чтобы вывести содержимое контейнера оператором вывода `<<`, оператор вывода переопределяют в поток для вектора:

```
#include <iostream>
#include <vector>
```

```
using namespace std;

ostream& operator<<(ostream& out, const vector<string>& container) {
    for (const string& element : container) {
        out << element << " "s;
    }
    return out;
}

int main() {
    const vector<string> cats = {"Мурка"s, "Белка"s, "Георгий"s, "Рюрик"s};
    cout << cats << endl;
}
```