

Junior C++ developer

Лекция 8

- Шаблоны

Шаблоны

- Шаблоны позволяют создавать многократно используемый код.
- Используя шаблоны, можно создавать **обобщенные** функции или классы.
- Тип данных задается как **параметр**, поэтому не нужно явным образом создавать реализации для каждого типа данных.

Главная идея шаблонов - написать класс или функцию один раз и они будут работать с любыми типами данных.

Большая часть стандартной библиотеки состоит из шаблонных классов

Коллекции

Сортировка

Поиск

STL - Standard Template
Library

Шаблон функций

Ключевое слово

Шаблонный
параметр **T**

```
template <typename T>
T max(T const& t1, T const &t2)
{
    return t1 < t2 ? t2 : t1;
}
```

Шаблон - это метакод, то есть код, который при компилировании генерирует другой код.

Функция **max** не существует до тех пор пока не будет вызвана в коде

Что написали мы:

Во что это превратил компилятор:

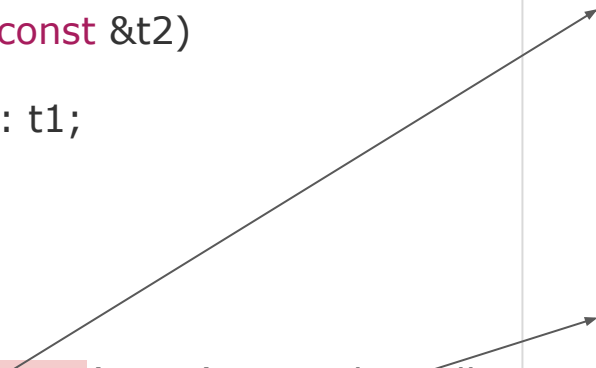
```
#include <iostream>
#include <string>

template <typename T>
T max(T const& t1, T const &t2)
{
    return t1 < t2 ? t2 : t1;
}

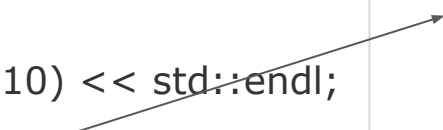
int main()
{
    std::cout << max<int>(5, 10) << std::endl;

    std::cout << max<std::string>("abz", "abk")
                << std::endl;

    return 0;
}
```



```
int max(int const& t1, int const &t2)
{
    return t1 < t2 ? t2 : t1;
}
```



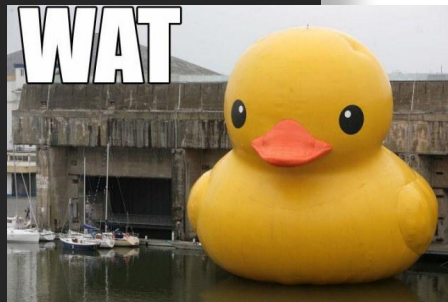
```
std::string max(std::string const& t1,
std::string const &t2)
{
    return t1 < t2 ? t2 : t1;
}
```

Компиляция шаблонов

Параметр шаблона потенциально может быть любым типом данных, но...

```
template <typename T>
T max(T val1, T val2) {
    if (val1 > val2)
        return val1;
    return val2;
}
int main() {
    Duck d1("Donald");
    Duck d2("Scrooge");
    Duck best_duck = max(d1, d2);
    cout << best_duck.getName() << " wins!" << endl;
}
```

Не
скомпилируется.



Компиляция шаблонов

1. Компилятор создает версию для используемого типа данных
2. Проверка на что компиляция для этого типа данных пройдет успешно

```
template <typename T>
T max(T val1, T val2) {
    if (val1 > val2)
        return val1;
    return val2;
}
int main() {
    Duck d1("Donald");
    Duck d2("Scrooge");
    Duck best_duck = max(d1, d2);
    cout << best_duck.getName() << " wins!" << endl;
}
```

Если для типа **Duck** не определен **operator>**, то код не скомпилируется

Шаблон класса

```
template <typename T>
class Accum
{
public:
    Accum(T start):
        total(start) {}
    T operator+=(T const& t)
    { return total = total + t; }
    T getTotal() const
    { return total; }
private:
    T total;
};
```

Класс для работы с
разными типами данных

```
int main()
{
    Accum<int> integers(0);
    Accum<std::string> string("");
    return 0;
}
```

C++17

```
int main()
{
    Accum<int> integers(0);
    Accum<std::string> string("");
    return 0;
}
```

В **C++17** можно явно не указывать тип шаблонного параметра, компилятор может вывести его самостоятельно

```
Accum integers(0);
Accum strings(std::string(""));
```

Специализация шаблонов

```
Person start("", "", 0);  
Accum<Person> people(start);  
Person p1("f1", "s1", 17);  
Person p2("f2", "s2", 19);
```

```
people += p1;  
people += p2;
```

```
std::cout << people.getTotal() << std::endl;
```

Что произойдет при
компиляции?

Специализация шаблонов

```
Person start("", "", 0);  
Accum<Person> people(start);  
Person p1("f1", "s1", 17);  
Person p2("f2", "s2", 19);
```

```
people += p1;  
people += p2;
```

```
std::cout << people.getTotal() << std::endl;
```

Что произойдет при
компиляции?

error: no match for 'operator+' (operand
types are 'Person' and 'const Person')
return total = total + t;

Специализация шаблонов

Когда необходимо изменить поведение шаблона для определенного типа данных используется специализация шаблонов.

```
template <typename T>
class Accum
{
public:
    Accum(T start):
        total(start) {}
    T operator+=(T const& t)
    {
        return total = total + t;
    }

    T getTotal() const
    {
        return total;
    }

private:
    T total;
};
```

```
template <>
class Accum<Person>
{
public:
    Accum(int start):
        total(start) {}
    int operator+=(Person const& t)
    {
        return total = total + t.getId();
    }

    int getTotal() const
    {
        return total;
    }

private:
    int total;
};
```

Специализация
для класса
Person