

Шаблоны

TLDR; Шаблоны

- Шаблоны — инструмент статического полиморфизма
- Благодаря шаблонам существует стандартная библиотека C++
- Шаблоны не поддерживают отдельную компиляцию

Проблема

- В языке C нет развитой стандартной библиотеки.
- За 50 лет существования языка не появилось
- И не появится
- GLib не предлагать

Много ли нам нужно?

- Список
- Строка
- Словарь
- Алгоритмы

В терминологии абстрактных типов данных

Много ли нам нужно?

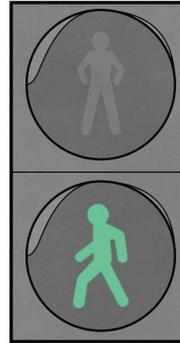
- И расширяемость
- И производительность
- И можно без хлеба (нет)



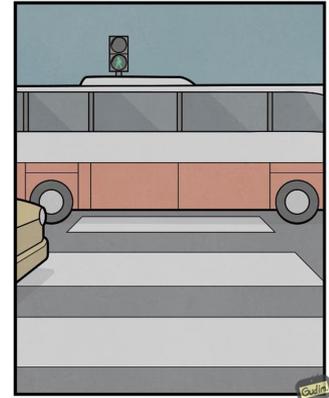
Можно поспорить

- Есть множество функций для работы со строками
strstr, strcat, strcmp, ...
Да, но управление памятью на разработчике
- Вы можете написать свои
Да, но не будете
- Есть даже qsort
Да, но... уже обсуждали

ДА,



НО



Тем временем в C++

- 1983 — C++
- 1991 — Появились шаблоны
- 1993 — Создание STL

Причины

- В С крайне мало выразительных средств
- Очень скудный полиморфизм на уровне языка
- Средства для написания обобщенного кода:
 - Макросы
 - void*

Проблемы макросов

```
#define MAX(a, b) (((b) < (a)) ? (a) : (b))  
int a = 1, b = 2;  
int m = MAX(a++, b++); // m = ?
```

Проблемы макросов

```
#define MAX(a, b) (((b) < (a)) ? (a) : (b))
```

```
int a = 1, b = 2;
```

```
int m = MAX(a++, b++); // m = ?, b = ?
```

Проблемы макросов

```
#define MAX(a, b) (((b) < (a)) ? (a) : (b))  
int a = 1, b = 2;  
int m = MAX(a++, b++); // m = ?, b = ?
```

Проблемы макросов

```
#define MAX(a, b) (((b) < (a)) ? (a) : (b))  
int a = 1, b = 2;  
int m = MAX(a++, b++); // m = 3, b = 4
```

Обобщения функций

// Вынужденно стираем типы

```
int IntLess(const void* lhs, const void* rhs) {  
    const auto lhs_value = *(const int*)lhs;  
    const auto rhs_value = *(const int*)rhs;  
    return lhs_value - rhs_value;  
}
```

(Не)обобщенный код

// Отдельные функции под каждый тип

```
double fmax(double x, double y);
```

```
float fmaxf(float x, float y);
```

```
long double fmaxl(long double x, long double y);
```

Generic Selection

Как вам такое решение?

```
#define print_number(X) \  
    _Generic((X), int: print_int, \  
            long: print_long, \  
            double: print_double)(X)
```

Generic Selection

Первым аргументом может быть только одно выражение

`_Generic((X), ...)`

Как реализовать MAX?

Generic Selection

```
#define MAX(X, Y) \  
    _Generic(X, char: max_char, \  
             int: max_int)(X, Y)
```

```
#define MAX(X, Y) \  
    _Generic(X + Y, char: max_char, \  
             int: max_int)(X, Y)
```

Какая функция будет вызвана?

```
#define MAX(X, Y) _Generic(X, ...)
```

```
MAX(1, 2);
```

```
MAX(1, 'a');
```

```
MAX('a', 'b');
```

```
char a = 'a', b = 'b';
```

```
MAX(a, b);
```

```
MAX(a, 256);
```

Какая функция будет вызвана?

```
#define MAX(X, Y) _Generic(X, ...)  
  
MAX(1, 2);           // max_int  
MAX(1, 'a');        // max_int  
MAX('a', 'b');     // max_int  
  
char a = 'a', b = 'b';  
MAX(a, b);         // max_char  
MAX(a, 256);      // max_char + warning
```

Какая функция будет вызвана?

```
#define MAX(X, Y) _Generic(X + Y, ...)
```

```
MAX(1, 2);
```

```
MAX(1, 'a');
```

```
MAX('a', 'b');
```

```
char a = 'a', b = 'b';
```

```
MAX(a, b);
```

```
MAX(a, 256);
```

Какая функция будет вызвана?

```
#define MAX(X, Y) _Generic(X + Y, ...)  
  
MAX(1, 2);           // max_int  
MAX(1, 'a');        // max_int  
MAX('a', 'b');     // max_int  
  
char a = 'a', b = 'b';  
MAX(a, b);          // max_int  
MAX(a, 256);        // max_int
```

Обобщенный код в языке C

Макросы:

- Не контролируют типы
- Небезопасны в отношении побочных эффектов
- Не являются функциями (не могут быть переданы в другие функции)

void*:

- Небезопасен в отношении типов
- Также передача функций по указателю может приводить к падению производительности.

Вернемся в С++

Два основных вида полиморфизма

- Динамический (Полиморфизм подтипов)
- Статический (Шаблоны)

Пример полиморфизма подтипов

```
struct Shape {  
    virtual ~Shape() = default;  
    virtual void draw() const = 0;  
};  
  
void f(const Shape& s) { s.draw(); }  
  
Circle c(...);  
f(c);
```

Пример полиморфизма подтипов

```
void f(const Shape& s) { s.draw(); }
```

Здесь:

- Shape — статический тип объекта s.
- Динамический тип известен во время выполнения
- Адрес виртуального метода draw определяется во время выполнения, вызов через указатель из vtable

Пример шаблона

```
template <typename T>
```

```
T max(T a, T b) {
```

```
    return b < a ? a : b;
```

```
}
```

Использование шаблона

```
int main() {  
    double a = 3.1415, b = 2.7183;  
    std::cout << ::max(a, b) << '\n';  
  
    std::string s1 = "hello", s2 = "world";  
    std::cout << ::max(s1, s2) << '\n';  
}
```

Зачем :: ?

NB

Явно обращаемся к `max` из глобальной области видимости ::

Избегаем неоднозначности с `std::max`

Keyword: ADL

Двухфазная трансляция

Шаблоны компилируются в два этапа:

1. Во время определения: проверка корректности кода без учета **T**
2. Во время инстанцирования: полная проверка.

Инстанцирование = генерация кода

```
double max<double>(double, double):
```

```
    push    rbp
    mov     rbp, rsp
```

```
    ...
```

```
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>
max<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>>
(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
 std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >):
```

```
    push    rbp
    mov     rbp, rsp
    sub    rsp, 32
```

```
    ...
```

Упражнение

```
// Пусть в текущем namespace нет функции f()  
// Результат компиляции кода?
```

```
template <typename T>  
T id(T value) {  
    f();  
    return value;  
}
```

Упражнение

```
// Пусть в текущем неймспейсе нет функции f()  
// Результат компиляции кода?
```

```
template <typename T>  
T id(T value) {  
    f();  
    return value;  
}
```

Ошибка компиляции при определении

Упражнение

```
// Пусть в текущем неймспейсе нет функции f()  
// Результат компиляции кода?
```

```
template <typename T>  
T id(T value) {  
    f(value);  
    return value;  
}
```

Ответ

- На этапе определения ошибки нет
- Поиск f в глобальном пространстве имен
- ...и в пространстве имен конкретного T (ADL)

Выводы

1. Шаблоны особенно чувствительны к наличию тестов
2. Компилируется при определении != скомпилируется при
инстанцировании

В шаблонах классов мы еще увидим ленивую механику
инстанцирования.

Следствие двухфазной трансляции

На момент инстанцирование должно быть доступно тело функции.

Раздельная компиляция (hpp/cpp) с шаблонами не работает.

Отделить определение можно, но...

```
#pragma once
```

```
template <typename T>  
T max(T a, T b);
```

```
#include "max-inl.cpp"
```

Цена универсальности

1. Увеличение времени компиляции

(все реализации в заголовочных файлах, многократный парсинг)

2. Увеличение объема кода

(каждый тип порождает новый инстанс)

ВЫВОД ТИПОВ

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

Пусть есть вызов `max(1, 2);`

Выводится два типа:

1. `T = int`
2. `a, b = const int&`

Преобразования типов

Во время вывода типов автоматические преобразования ограничены:

- При передаче по ссылке типы должны совпадать точно.
- При передаче по значению допускаются decay преобразования
 - Отбрасывание const/volatile
 - Отбрасывание ссылок
 - Функции/массивы преобразуются в указатели

Decay преобразования

```
template <typename T> T max(T a, T b) { ... }
```

```
int i = 13;  
const int c = 42;
```

```
max(i, c);    // T = ?  
max(c, c);    // T = ?
```

```
int& ir = i;  
max(i, ir);    // T = ?
```

```
int arr[4];  
max(&i, arr); // T = ?
```

Decay преобразования

```
template <typename T> T max(T a, T b) { ... }
```

```
int i = 13;  
const int c = 42;
```

```
max(i, c);    // T = int  
max(c, c);    // T = int
```

```
int& ir = i;  
max(i, ir);   // T = int
```

```
int arr[4];  
max(&i, arr); // T = int*
```

Стандартные преобразования запрещены

```
template <typename T> T max(T a, T b) { ... }
```

```
// deduced conflicting types for parameter 'T'  
// ('int' vs. 'double')
```

```
max(42, 3.1415);
```

```
// deduced conflicting types for parameter 'T'  
// ('const char*' vs. 'std::basic_string<char>')  
::max("hello", std::string("world"));
```

Способы обхода таких ошибок

1. Явное преобразование аргументов

```
max(static_cast<double>(42), 3.1415);
```

2. Явное указание типа T

```
max<double>(42, 3.1415);
```

3. Указание разных type-переменных для разных аргументов

```
template <typename T, typename U>  
? max(T a, U b) {...}
```

Способы обхода таких ошибок

1. Явное преобразование аргументов

```
max(static_cast<double>(42), 3.1415);
```

2. Явное указание типа T

```
max<double>(42, 3.1415);
```

3. Указание разных type-переменных для разных аргументов

```
template <typename T, typename U>  
auto max(T a, U b) {...}
```

Просмотр выведенного типа

```
template <typename T>  
class TD;
```

```
TD<decltype(x)> a;
```

Шаблоны классов

```
template <typename T>
class Wrapper {
public:
    const T& value() const {
        return value_;
    }

    void f() { value_.f(); }

private:
    T value_{};
};
```

```
int main() {
    Wrapper<int> w;
    std::cout << w.value();
}
```

Какие проблемы вы видите в коде?

Шаблоны классов

```
template <typename T>
class Wrapper {
public:
    const T& value() const {
        return value_;
    }

    void f() { value_.f(); }

private:
    T value_{};
};
```

```
int main() {
    Wrapper<int> w;
    std::cout << w.value();
}
```

*Если метод не вызван,
он не будет инстанцирован*

Non-type parameters

```
template <typename T, std::size_t Capacity>
class Stack {
    private:
        T items_[Capacity];
        std::size_t size_ = 0;
};
```

Вместо заключения

- Правил много, нужны примеры
- Основная идея: программирование по статическому контракту

Q&A