

Вариативные шаблоны и элементы метапрограммирования

2019

Вспомним рекурсивные функции

```
void intersperse_v1(std::list<float> x) {  
    if (x.empty()) {  
        return;  
    }  
    std::cout << x.front();  
    x.pop_front();  
    for (float f : x) {  
        std::cout << ', ' << f;  
    }  
}
```

```
intersperse_v1({1,2,3}); // 1,2,3
```

Вспомним рекурсивные функции

```
void intersperse_v2(std::list<float> x) {  
    if (x.empty()) {  
        return; // остановка рекурсии  
    } else if (x.size() == 1) {  
        std::cout << x.front();  
    } else {  
        std::cout << x.front() << ', ';  
    }  
    x.pop_front();  
    intersperse_v2(x); // рекурсия  
}
```

```
intersperse_v2({1,2,3}); // 1,2,3
```

Вспомним рекурсивные функции

```
template <class Head>
void intersperse_v3(Head head) {
    std::cout << head; // остановка рекурсии
}

template <class Head, class ... Tail>
void intersperse_v3(Head head, Tail ... tail) {
    std::cout << head << ',';
    intersperse_v3(tail...); // рекурсия
}

intersperse_v3(1, 2, 3); // 1,2,3
intersperse_v3("a", 2, 3.33); // a,2,3.33
```

Вызов конструктора

```
template <class T, class ... Args>
T make(Args&& ... args) {
    return T(std::forward<Args>(args)...);
}
```

```
make<std::string>(3, 'a'); // "aaa"
```

```
make<std::vector<char>>(3, 'a'); // {'a', 'a', 'a'}
```

Пример из стандартной библиотеки:

```
std::vector<std::pair<std::string, float>> x;
x.emplace_back("hello", 13);
```

Привязка аргументов функции

```
float add(float x, float y) { return x + y; }  
  
using namespace std::placeholders;  
auto add2 = std::bind(add, _1, 2.0f);  
float sum = add2(8); // 10
```

Заглушки в стандартной библиотеке C++:

```
namespace std {  
    template <int N> struct _Placeholder {};  
    namespace placeholders {  
        extern const _Placeholder<1> _1;  
        extern const _Placeholder<2> _2;  
        // ...  
    }  
}
```

Кортежи

```
template <class ... Args> class tuple;  
  
template <class Head, class ... Tail>  
class tuple<Head,Tail...>: public tuple<Tail...> {  
    // ...  
};
```

Кортежи

```
template <class ... Args> class tuple;  
  
template <class Head, class ... Tail>  
class tuple<Head,Tail...>: public tuple<Tail...> {  
    // ...  
};
```

```
template <class ... Args>  
tuple<Args...> make_tuple(Args&& ... args) {  
    return tuple<Args...>(std::forward<Args>(args)...);  
}
```


Кортежи

```
std::vector<std::tuple<std::string, float, int>> x;  
x.emplace_back("hello", 13, 27);
```

Кортежи

```
std::vector<std::tuple<std::string, float, int>> x;  
x.emplace_back("hello", 13, 27);
```

```
std::get<0>(x.front()); // "hello"  
std::get<2>(x.front()); // 27
```

Кортежи

```
std::vector<std::tuple<std::string, float, int>> x;  
x.emplace_back("hello", 13, 27);
```

```
std::get<0>(x.front()); // "hello"  
std::get<2>(x.front()); // 27
```

```
std::string s;  
float f;  
std::tie(s, f, std::ignore) = x.front(); // s="hello" f=13
```

Кортежи

```
struct Person {  
  
    std::string firstName;  
    std::string lastName;  
  
    bool operator<(const Person& p) {  
        return std::tie(lastName, firstName) <  
            std::tie(p.lastName, p.firstName);  
    }  
  
};
```

Векторное произведение

```
template <class T> std::vector<T>
cross(std::vector<T> x, std::vector<T> y) {
    return {
        x[1]*y[2] - y[1]*x[2],
        y[0]*x[2] - x[0]*y[2],
        x[0]*y[1] - y[0]*x[1]
    };
}
```

Векторное произведение

```
template<class T1, class T2>
```

```
???
```

```
cross(const ETBase<T1>& d1, const ETBase<T2>& d2) {  
    return sum(  
        sum(  
            LeviCivita() *  
            d1.unwrap()(tensor::j) *  
            d2.unwrap()(tensor::k),  
            tensor::k  
        ),  
        tensor::j  
    );  
}
```

```

ArrayExpr<
  ArrayExprReduce<
    ArrayExpr<
      ArrayExprReduce<
        typename BinaryExprResult<
          Multiply,
          typename BinaryExprResult<Multiply,ArrayExpr<LeviCivita>,
            ArrayExpr<ArrayIndexMapping<typename asExpr<T1>::T_expr,1,0,0,0,0,0,0,0,0,0,0>>::T_result,
            ArrayExpr<ArrayIndexMapping<typename asExpr<T2>::T_expr,2,0,0,0,0,0,0,0,0,0,0>>::T_result,
              2,
              ReduceSum<
                typename BinaryExprResult<Multiply,
                typename BinaryExprResult<Multiply,ArrayExpr<LeviCivita>,
                  ArrayExpr<ArrayIndexMapping<typename asExpr<T1>::T_expr,1,0,0,0,0,0,0,0,0,0,0>>::T_result,
                  ArrayExpr<
                    ArrayIndexMapping<typename asExpr<T2>::T_expr,2,0,0,0,0,0,0,0,0,0,0>>::T_result::T_numtype,
                    BZ_SUMTYPE(bzCC(typename BinaryExprResult<
                      Multiply,
                      typename BinaryExprResult<Multiply,ArrayExpr<LeviCivita>,
                        ArrayExpr<ArrayIndexMapping<typename asExpr<T1>::T_expr,1,0,0,0,0,0,0,0,0,0,0>>::T_result,
                        ArrayExpr<
                          ArrayIndexMapping<typename asExpr<T2>::T_expr,2,0,0,0,0,0,0,0,0,0,0>>::T_result::T_numtype)))>>>,
                      1,
                      ReduceSum<
                        BZ_SUMTYPE(bzCC(typename BinaryExprResult<Multiply,
                        typename BinaryExprResult<Multiply,ArrayExpr<LeviCivita>,
                          ArrayExpr<ArrayIndexMapping<typename asExpr<T1>::T_expr,1,0,0,0,0,0,0,0,0,0,0>>::T_result,
                          ArrayExpr<
                            ArrayIndexMapping<typename asExpr<T2>::T_expr,2,0,0,0,0,0,0,0,0,0,0>>::T_result::T_numtype)))>>>

```

*[мне] потребовалось в 10 раз больше времени,
чтобы понять, как написать возвращаемый
тип, чем сделать все остальное...*

автор Blitz++

Векторное произведение

```
template<class T1, class T2>
auto
cross(const ETBase<T1>& d1, const ETBase<T2>& d2)
-> decltype(/* sum(sum(...)) */) {
    return sum(
        sum(
            LeviCivita() *
            d1.unwrap()(tensor::j) *
            d2.unwrap()(tensor::k),
            tensor::k
        ),
        tensor::j
    );
}
```


Векторное произведение

```
template<class T1, class T2>
auto // c++14
cross(const ETBase<T1>& d1, const ETBase<T2>& d2) {
    return sum(
        sum(
            LeviCivita() *
            d1.unwrap()(tensor::j) *
            d2.unwrap()(tensor::k),
            tensor::k
        ),
        tensor::j
    );
}
```

Нормализация типов

```
// объявление
template <class T> struct remove_reference;

// определение
template <class T> struct remove_reference {typedef T type;};

// специализация для lvalue-ссылок
template <class T> struct remove_reference<T&> {typedef T type;};

// специализация для rvalue-ссылок
template <class T> struct remove_reference<T&&> {typedef T type;};
```

Сравнение типов

```
struct true_type { static const bool value = true; };  
struct false_type { static const bool value = false; };  
  
// объявление  
template <class T>  
struct is_same;  
  
// определение  
template <class T, class U>  
struct is_same: public false_type {};  
  
// специализация для одинаковых типов  
template <class T>  
struct is_same<T,T>: public true_type {};
```

Инварианты

```
// вектор только для чисел с плавающей точкой
template <class T>
class floating_point_vector {
    static_assert(
        typename std::is_floating_point<T>::value, "bad T");
};

// преобразование Фурье только для комплексных чисел
// одинарной точности и только для 1,2,3 измерений
template <class T, int N>
class Fourier_transform {
    static_assert(
        std::is_same<std::complex<float>,T>::value, "bad T");
    static_assert(0<N && N<=3, "bad N");
};
```

Другие операции

```
#include <type_traits> // is_arithmetic, is_integral, is_unsigned

namespace sys {
    class uint128_t; // беззнаковое целое 128 бит (16 байт)
}

namespace std {

    template<>
    struct is_arithmetic<sys::uint128_t>: public true_type {};

    template<>
    struct is_integral<sys::uint128_t>: public true_type {};

    template<>
    struct is_unsigned<sys::uint128_t>: public true_type {};

}
```

Ссылки

- ▶ [A Brief Introduction to Variadic Templates \(N2087\).](#)