Проблема “одинаковых классов”

```
struct ArrayInt {
explicit ArrayInt(size_t size)
    : data_(new int[size])
    , size_(size) {}

~ArrayInt() {delete [] data_;}

size_t size() const
{ return size_; }

int operator[](size_t i) const
{ return data_[i]; }

int & operator[](size_t i)
{ return data_[i]; }
...
private:
    int *   data_;
    size_t  size_;
};
```

```
struct ArrayFlt {
explicit ArrayFlt(size_t size)
    : data_(new float[size])
    , size_(size) {}

~ArrayFlt() {delete [] data_;}

size_t size() const
{ return size_; }

float operator[](size_t i) const
{ return data_[i]; }

float & operator[](size_t i)
{ return data_[i]; }
...
private:
    float  * data_;
    size_t   size_;
};
```

Решение в стиле C: макросы

```
#define DEFINE_ARRAY(Name, Type)\
struct Name {                   \
explicit Name(size_t size)      \
    : data_(new Type[size])     \
    , size_(size) {}            \
~Name() { delete [] data_; }    \
                                \
size_t size() const             \
{ return size_; }               \
                                \
Type operator[](size_t i) const \
{ return data_[i]; }            \
Type & operator[](size_t i)     \
{ return data_[i]; }            \
...                             \
private:                        \
    Type *  data_;              \
    size_t  size_;              \
}
```

```
DEFINE_ARRAY(ArrayInt, int);
DEFINE_ARRAY(ArrayFlt, float);

int main()
{
    ArrayInt ai(10);
    ArrayFlt af(20);
    ...
    return 0;
}
```

Решение в стиле C++: шаблоны классов

```
template <class Type>
struct Array {
explicit Array(size_t size)
    : data_(new Type[size])
    , size_(size) {}
~Array() { delete [] data_; }

size_t size() const
{ return size_; }

Type operator[](size_t i) const
{ return data_[i]; }
Type & operator[](size_t i)
{ return data_[i]; }
...
private:
    Type *  data_;
    size_t  size_;
};
```

```
int main()
{
    Array<int> ai(10);
    Array<float> af(20);
    ...
    return 0;
}
```

Шаблоны классов с несколькими параметрами

```
template <class Type,
          class SizeT = size_t,
          class CRet = Type>
struct Array {
explicit Array(SizeT size)
    : data_(new Type[size])
    , size_(size) {}
~Array() {delete [] data_;}

SizeT size() const {return size_;}
CRet operator[](SizeT i) const
{ return data_[i]; }
Type & operator[](SizeT i)
{ return data_[i]; }
...
private:
    Type *  data_;
    SizeT   size_;
};
```

```
void foo()
{
    Array<int> ai(10);
    Array<float> af(20);
    Array<Array<int>,
          size_t,
          Array<int> const&>
        da(30);
    ...
}

typedef Array<int> Ints;
typedef Array<Ints, size_t,
    Ints const &> IInts;

void bar()
{
    IInts da(30);
}
```

Шаблоны функций: возведение в квадрат

```
// C
int   squarei(int   x)   { return x * x; }
float squaref(float x)   { return x * x; }

// C++
int   square(int   x)  { return x * x; }
float square(float x)  { return x * x; }

// C++ + OOP
struct INumber {
    virtual  INumber * multiply(INumber * x) const = 0;
};
struct Int   : INumber { ... };
struct Float : INumber { ... };
INumber * square(INumber * x) { return x->multiply(x); }

// C++ + templates
template <typename Num>
Num square(Num x) { return x * x; }
```

Шаблоны функций: сортировка

```
// C
void qsort(void * base, size_t nitems, size_t size, /*function*/);

// C++
void sort(int    * p,   int      * q);
void sort(double * p,   double   * q);

// C++ + OOP
struct IComparable {
    virtual  int compare(IComparable * comp) const = 0;
    virtual ~IComparable() {}
};
void sort(IComparable ** p, IComparable ** q);

// C++ + templates
template <typename Type>
void sort(Type * p, Type * q);
```

NB: у шаблонных функций нет параметров по умолчанию.

Вывод аргументов (deduce)

```
template <typename Num>
Num square(Num n) { return n * n; }

template <typename Type>
void sort(Type * p, Type * q);

template <typename Type>
void sort(Array<Type> & ar);

void foo() {
    int a = square<int>(3);
    int b = square(a) + square(4); // square<int>(..)
    float * m = new float[10];
    sort(m, m + 10);     // sort<float>(m, m + 10)
    sort(m, &a);         // error: sort<float> vs. sort<int>
    Array<double> ad(100);
    sort(ad);            // sort<double>(ad)
}
```

Шаблоны методов

```
template <class Type>
struct Array {
    template<class Other>
    Array( Array<Other> const& other )
        : data_(new Type[other.size()])
        , size_(other.size()) {
        for(size_t i = 0; i != size_; ++i)
            data_[i] = other[i];
    }

    template<class Other>
    Array & operator=(Array<Other> const& other);
    ...
};

template<class Type>
template<class Other>
Array<Type> & Array<Type>::operator=(Array<Other> const& other)
{ ... return *this; }
```

Функции для вывода параметров

```cpp
template<class First, class Second>
struct Pair {
    Pair(First const& first, Second const& second)
        : first(first), second(second) {}
    First first;
    Second second;
};

template<class First, class Second>
Pair<First, Second> makePair(First const& f, Second const& s) {
    return Pair<First, Second>(f, s);
}

void foo(Pair<int, double> const& p);

void bar() {
    foo(Pair<int, double>(3, 4.5));
    foo(makePair(3, 4.5));
}
```