

# Программирование на языке C++

## Лекция 6

### Шаблоны классов

Александр Смаль

# 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> typename
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,  
         ArrayInt<int> const&>  
        da(30);  
    ...  
}  
  
typedef Array<int> Ints;  
typedef Array<Ints, size_t,  
             Ints const &> IInts;  
  
void bar()  
{  
    IInts da(30);  
}
```

*Handwritten red annotations:*

- Red arrow pointing to `class Type` in the first template parameter.
- Red arrow pointing to `class SizeT = size_t` in the second template parameter.
- Red arrow pointing to `class CRet = Type` in the third template parameter.
- Red arrow pointing to `Array<int>` in the first argument of `Array` in `foo()`.
- Red arrow pointing to `size_t` in the second argument of `Array` in `foo()`.
- Red arrow pointing to `ArrayInt<int> const&` in the third argument of `Array` in `foo()`.
- Red arrow pointing to `Ints` in the first argument of `Array` in the `typedef` for `IInts`.
- Red arrow pointing to `size_t` in the second argument of `Array` in the `typedef` for `IInts`.
- Red arrow pointing to `Ints const &` in the third argument of `Array` in the `typedef` for `IInts`.