Templates (24.11.17)

Полиморфизм

- Runtime
  - виртуальные функции
- Compile-time
  - перегрузка функций
  - шаблоны

Синтаксис

- Шаблонные функции
  - template<*parameter-list*> *function-declaration*
  - template<typename T> T std::abs(T) {}
- Шаблонные классы
  - template<*parameter-list*> *class-declaration*
  - template<class T, class Allocator=std::allocator<T>>
    class vector {}

Шаблонные параметры

- Non-type параметры
  - Целочисленные типы (float, double нельзя)
  - Enum
  - Lvalue-ссылки (константы времени компиляции)
  - Указатели (константы времени компиляции)

Пример:

```
std::bitset<256> b;
```

Шаблонные параметры

- template параметры
  - std::vector<int> v;
- template template параметры
  - template<typename K, typename V,
    template<typename> typename C>
    class CMap {
    C<K> keys;
    C<V> value;
    };

Специализация

```
template<typename T1, typename T2, int n> class A {...}
```

- Частичная
  - `template<typename T, int n> class A<T, T*, n> {...}`
  - `template<typename T, int n> class A<T, int, n> {...}`
- Полная
  - `template<> class A<double, int, 5> {...}`

Неявные интерфейсы

```
// какие требования к Message ?
template<typename Message>
void processMessage( Message& m )
{
    if( w.length() < 10 && w != ErrorMessage ) {
        Message copy(m);
        copy.send();
    }
}
```

Typename

- Нет разницы в использовании “в шапке”
  - Замечание про TTP (C++17)
- Использование typename в теле функции/класса

Typename в теле функции/класса

```
template<typename C>
void print2nd( const C& container )
{
    // C::const iterator - поле
    C::const_iterator * x;
    // C::const iterator - тип
    typename C::const_iterator* x;
```

Typename. Исключение для базовых классов

- Базовый класс:
  - template<typename T>
    class Derived: public Base<T>::Nested {
        …
    }
- Аналогично для базовых классов в списках инициализации

Наследование

```
template<typename Company>
class LoggingMsgSender: public MsgSender<Company> {
public:
    void sendClearMsg(const MsgInfo& info)
    {
        sendClear(info); // функция из MsgSender
                         // ошибка компиляции
    }
};
```

Наследование

- Причина: даже если у базового класса такой метод есть, то в специализации базового класса этого метода может не быть
  (принцип раннего обнаружения ошибки)
- Решение:
  - using
  - this->
  - BaseClass<T>::method()

Code bloat

- Важно помнить, что для каждой инстанциации генерируется код, следовательно можно вынести код, не зависящий от параметра, из шаблона
- В противном случае возможно существенное увеличение размера объектных файлов

Conversions

```
class A {...};
class B : public A {...};

A* ptr = new B();

//как написать:
SmartPtr<A> ptr = SmartPtr<B>(new B());
```

Conversions

```
template<typename T>
class SmartPtr {
    public:
        template<typename U>
        SmartPtr( const SmartPtr<U>& other ) :
            ptr( other.get() ) {}
    private:
        T* ptr;
};
```

Литература

- Scott Mayers “Effective C++ (55)”
- Scott Mayers “Effective Modern C++ (42)”

Метапрограммирование

Метапрограммирование

```
template <int N>
struct Factorial
{
	enum { value = N * Factorial<N - 1>::value };
};

template <>
struct Factorial<0>
{
	enum { value = 1 };
};
```