Обработка ошибок: гарантии безопасности исключений

Александр Смаль

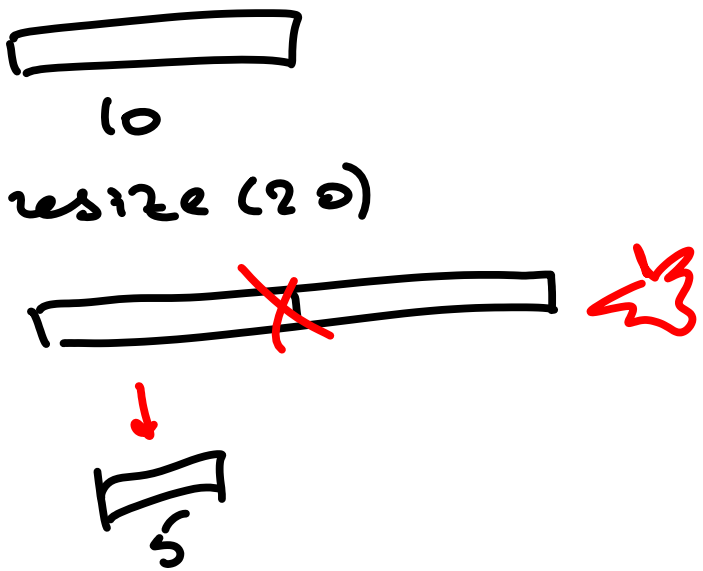
Академический университет
20 марта 2014
Санкт-Петербург

Гарантии безопасности исключений

- **Гарантия отсутствия исключений**
  “Ни при каких обстоятельствах функция не будет генерировать исключения”.

Гарантии безопасности исключений

- **Гарантия отсутствия исключений**
  “Ни при каких обстоятельствах функция не будет генерировать исключения”.
- **Базовая гарантия**
  “При возникновении любого исключения в некотором методе, состояние программы должно оставаться согласованным”.



Гарантии безопасности исключений

Деструкторы
Освобождение
ресурсов
swap

- **Гарантия отсутствия исключений**
  "Ни при каких обстоятельствах функция не будет генерировать исключения".
- **Базовая гарантия**
  "При возникновении любого исключения в некотором методе, состояние программы должно оставаться согласованным".
- **Строгая гарантия**
  "Если при выполнении операции возникает исключение, то программа должна остаться в состоянии, которое было до начала выполнения операции".

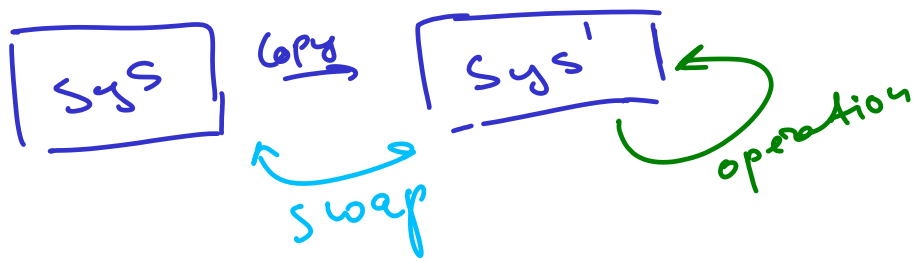
Транзакционность

Строгая гарантия исключений

- В каком случае мы не можем обеспечить строгую гарантию исключений?

Если есть взаимодействие с внешним миром.

- Как обеспечить строгую гарантию в остальных случаях?



- Когда можно обеспечить строгую гарантию *эффективно*?

В чём сложность?

```
template<class T>
struct Array {
    void resize(size_t n) {
        T * ndata = new T[n]; // bad_alloc
        for (size_t i = 0; i != n && i != size_; ++i)
            ndata[i] = data_[i]; // оператор =

        delete [] data_;
        data = ndata;
        size_ = n;
    }

T *     data_;
size_t  size_;
};
```

никакой гарантии

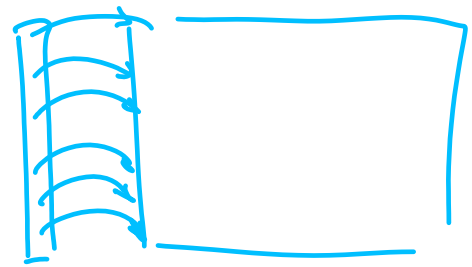
В чём сложность?

```
template<class T>
struct Array {
    void resize(size_t n) {
        T * ndata = 0;
        try {
            ndata = new T[n];

            for (size_t i = 0; i != n && i != size_; ++i)
                ndata[i] = data_[i];
        } catch (...) {
            delete [] ndata;
            throw;
        }
        delete [] data_;
        data = ndata;
        size_ = n;
    }

T *     data_;
size_t  size_;
};
```

// 1

// 2

//



Использование RAII

```
template<class T>
struct Array {
    void resize(size_t n) {
        shared_array<T> ndata = new T[n];

        for (size_t i = 0; i != n && i != size_; ++i)
            ndata.get()[i] = data_.get()[i];

        data_ = ndata;
        size_ = n;
    }

shared_array<T> data_;
size_t          size_;
};
```

Использование swap

```
template<class T>
struct Array {
    void resize(size_t n) {
        Array t(n);                    //
        for (size_t i = 0; i != n && i != size_; ++i)
            t[i] = data_[i];                                    //

        t.swap(*this);  // no exception
    }

T       * data_;
size_t    size_;
};
```

Проектирование с учётом исключений

Рассмотрим традиционный интерфейс стека:

```
template<class T>
struct Stack {
    void push(T const& t)
    {
        data.push_back(t);
    }
    void pop() {
        // T tmp = data_.back();
        data_.pop_back();
        // return tmp;
    }

    std::vector<T> data_;
};
```

T& top(){return v.back();}

Базовая гарантия исключений

Проектирование с учётом исключений

Рассмотрим традиционный интерфейс стека:

```
template<class T>
struct Stack {
    void push(T const& t)
    {
        data.push_back(t);
    }

    void pop(T & res) {
        res = data_.back(); // *
        data_.pop_back();
    }

    std::vector<T> data_;
};
```

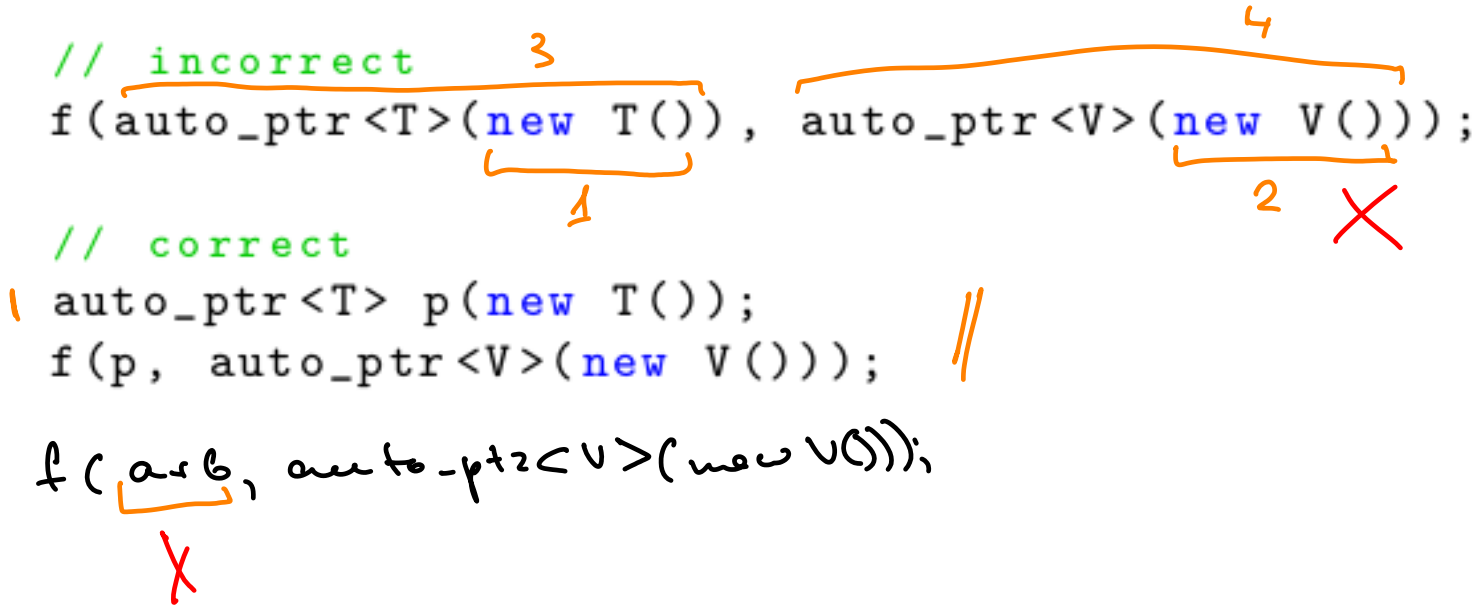
Использование auto_ptr

```
template<class T>
struct Stack {
    void push(T const& t)
    {
        data.push_back(t);
    }

    auto_ptr<T> pop() {
        auto_ptr<T> tmp = new T(data_.back());
        data_.pop_back();
        return tmp;
    }

    std::vector<T> data_;
};
```

Проблемы с RAII

Следите за порядком операций:

```
void f(auto_ptr<T> p, auto_ptr<V> q);

// incorrect
f(auto_ptr<T>(new T()), auto_ptr<V>(new V()));

// correct
auto_ptr<T> p(new T());
f(p, auto_ptr<V>(new V()));
```



Исключения в стандартной библиотеке

- `vector`, `deque`, `string`, `bitset` кидают `std::out_of_range` (фукция `at`).
- Оператор `new T` кидает `std::bad_alloc`.
  Версия оператора `new (std::nothrow) T` в случае ошибки возвращает 0.
- Потоки ввода-вывода.

```
std::ifstream file;
file.exceptions (std::ifstream::failbit |
                 std::ifstream::badbit );
try {
    file.open ("test.txt");
    while (!file.eof()) file.get();
    file.close();
}
catch (std::ifstream::failure const& e) {
    std::cerr << "Exception opening/reading/closing file\n";
}
```