Copy & Swap

Проблема

```
void f(std::size_t n) {
  int* items = new int[n]();
  g();
  delete[] items;
}
```

Объясните потенциальную проблему этого кода

Проблема

```
void f(std::size_t n) {
  int* items = new int[n]();
  g();  // 💥 throw
  delete[] items;
}
```

Утечка памяти при исключении из g()

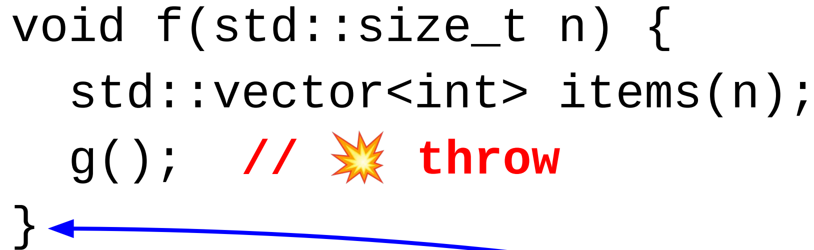
Предложите варианты решения

Решение 1: явная обработка исключений

```
void f(std::size_t n) {
  int* items = new int[n]();
  try {
    g();
  } catch (...) {      // обрабатываем все исключения
    delete[] items;    // освобождаем ресурс
    throw;             // бросаем пойманное исключение
  }
  delete[] items;
}
```

И так везде

Решение 2: RAII



```
void f(std::size_t n) {
  std::vector<int> items(n);
  g();  // 💥 throw
}
```

Выделенная память освободится в деструкторе std::vector

Функция f нейтральна к исключениям

Решение 3: не использовать исключения

```
int f() {
  int* items = new int[42]();
  int status = g(); // TODO: std::expected
  delete[] items;
  return status;
}
```

Это требует более глубокой проработки стратегии обработки ошибок

Реализуем специальные методы для класса

```
template <typename T>
class Vector {
public:
  ~Vector();
  Vector(const Vector& other);
  Vector(Vector&& other);
  Vector& operator=(const Vector& other);
  Vector& operator=(Vector&& other);
private:
  T* begin_ = nullptr;
  T* end_ = nullptr;
  T* capacity_ = nullptr;
};
```

Конструктор копирования: попытка 1

```
template <typename T>
Vector<T>::Vector(const Vector& other)
    : begin_(new T[other.size()]),
      end_(begin_ + other.size()),
      capacity_(end_) {
  for (std::size_t i = 0; i < other.size(); ++i) {
    begin_[i] = other.begin_[i];
  }
}
```

Покритикуйте это решение

Конструктор копирования: попытка 1

```
template <typename T>
Vector<T>::Vector(const Vector& other)
    : begin_(new T[other.size()]),
      end_(begin_ + other.size()),
      capacity_(end_) {
  for (std::size_t i = 0; i < other.size(); ++i) {
    begin_[i] = other.begin_[i];
  }
}
```

default ctors

copy assignment

https://godbolt.org/z/qox17hfhv

Конструктор копирования: попытка 2

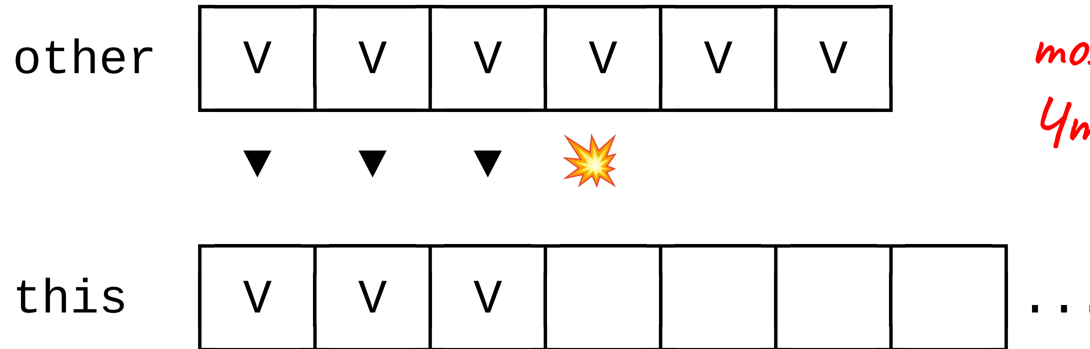
```
template <typename T>
Vector<T>::Vector(const Vector& other)
    : begin_(static_cast<T*>(operator new(sizeof(T) * other.size()))),
      end_(begin_ + other.size()),
      capacity_(end_) {
  for (std::size_t i = 0; i < other.size(); ++i) {
    new (begin_ + i) T(other.begin_[i]);
  }
}
```

Выделение памяти

Конструирование объектов

Покритикуйте решение с точки зрения безопасности исключений

Иллюстрация копирования



Смогли скопировать только часть. Что делать?

Разрушаем объекты и освобождаем память

```
template <typename T>
Vector<T>::Vector(const Vector& other)
    : begin_(static_cast<T*>(operator new(sizeof(T) * other.size()))),
      end_(begin_ + other.size()), capacity_(end_) {
  std::size_t current = 0;
  try {
    for (current = 0; current < other.size(); ++current) {
      new (begin_ + current) T(other.begin_[current]);
    }
  } catch (...) {
    for (std::size_t i = 0; i < current; ++i) {
      begin_[i].~T();
    }
    operator delete(begin_);
    throw;
  }
}
```


https://godbolt.org/z/v4aa81M6s

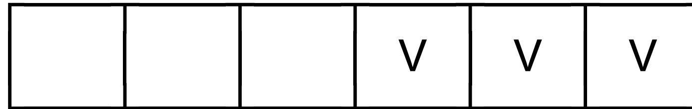
Конструктор перемещения: попытка 1

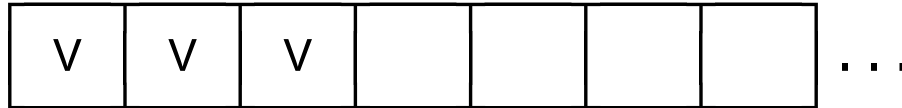
```
template <typename T>
Vector<T>::Vector(Vector&& other)
    : begin_(static_cast<T*>(operator new(sizeof(T) * other.size()))),
      end_(begin_ + other.size()), capacity_(end_) {
  for (std::size_t i = 0; i < other.size(); ++i) {
    new (begin_ + i) T(std::move(other.begin_[i]));
  }
}
```

Какая потенциальная проблема?

https://godbolt.org/z/zT7evv3o1

Иллюстрация перемещения





. . .

Смогли переместить только часть.

Что делать?

Конструктор перемещения: попытка 1

```
template <typename T>
Vector<T>::Vector(Vector&& other)
    : begin_(static_cast<T*>(operator new(sizeof(T) * other.size()))),
      end_(begin_ + other.size()), capacity_(end_) {
  for (std::size_t i = 0; i < other.size(); ++i) {
    new (begin_ + i) T(std::move(other.begin_[i]));
  }
}
```

Какая потенциальная проблема?

Конструктор перемещения: попытка 2

```
template <typename T>
Vector<T>::Vector(Vector&& other)
    : begin_(other.begin_),
      end_(other.end_),
      capacity_(other.capacity_) noexcept {
  other.begin_ = other.end_ = other.capacity_ = nullptr;
}
```


https://godbolt.org/z/qe5T415qM

Иллюстрация перемещения

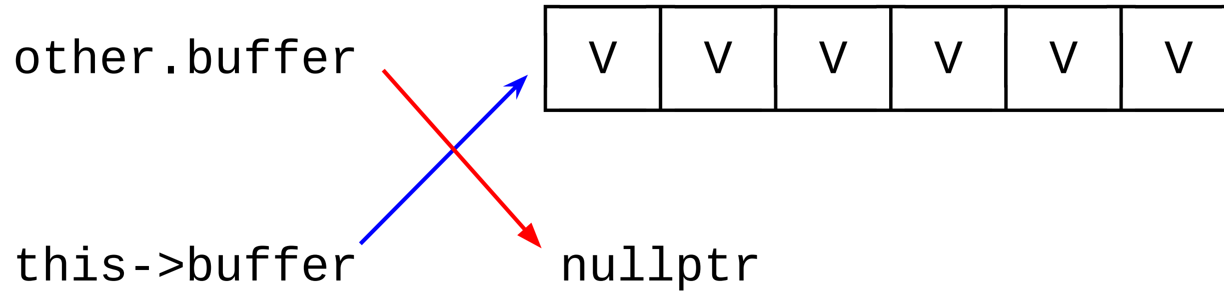
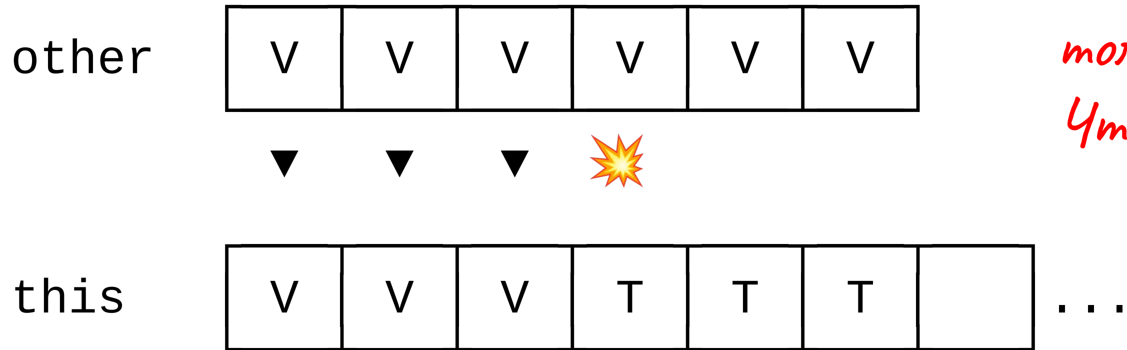


Иллюстрация копирующего присваивания



Смогли присвоить только часть. Что делать?

Copy And Swap

```
template <typename T>
Vector<T>& Vector<T>::operator=(const Vector& other) {
  Vector tmp(other);  // Не нарушает инвариант класса
  tmp.swap(*this);    // Не генерирует исключений
  return *this;
}
```

Самая простая, но не самая оптимальная реализация

Move And Swap

```
template <typename T>
Vector<T>& Vector<T>::operator=(Vector&& other) {
  Vector tmp(std::move(other));
  tmp.swap(*this);
  return *this;
}
```


https://godbolt.org/z/n5W4K3W1b

Q&A