5.15 Гарантии безопасности контейнеров относительно исключений. Проблемы с гарантиями exception-safety у map и unordered_map. Реализация exception safety на примере vector (с произвольным аллокатором): реализация безопасных конструкторов, операторов присваивания, метода push_back.

Гарантии безопасности относительно исключений

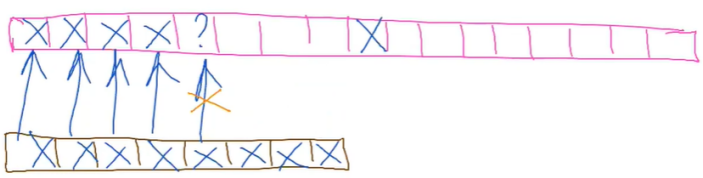
Рассмотрим пример работы вектора

```
void f() {
    Dangerous s(0);
    std::cout << s.x;
    g();
}

struct S {
    int x = 0;
    S(int x): x(x) {}
    S(const S& s): x(s.x) {
        if (x == 8)
            throw 1;
    }
};

int main() {

    std::vector<S> v;
    for (int i = 0; i < 100; ++i) {
        v.push_back(S(i));
    }


    try {
        f();
    } catch (...) {
        std::cout << "caught\n";
    }
}
```

Как мы уже знаем, при заполнении вектора полностью. он расширяется в два раза. Давайте представим, что мы положили какой-то объект, который при копировании бросаем исключение. Чт делать в этот момент вектору? Тут две возможности

1 Внетор может столкнуться с этим исключением, когда он кладет в себя объект

2 А может быть еще хуже. На восьмом шаге у него будет происходить следующее: он начнет перекладывать все преедыдущие объекты в новое место, и если в этот момент кто-то из тех, кого он перекладывает, выброст исключение, случится беда. Он уже часть нового storage выделил, поэтому если вектор реализован плохо, контейнер окажется в разломанном состоянии. Чтобы такого не было, нужно уничтожить все элементы, которые мы успели переложить, освободить память и вернуть все, как было



Функции могут давать или не давать гарантию безопасности относительно исключений. Контейнеры могут перестать работать корректно вследствие вызова исключений и вызвать UB.

Базовая (basic) гарантия: объект останется в валидном состоянии после вызова исключений

Сильная (strong) гарантия: объект останется в исходном состоянии после выхова исключений.

Почти все STL-библиотечные функции дают сильную гарантию безопасности.(vecor дает такую гарантию. Он устроен так, что если во время push_back вылетело исключение, он все вернет аккуратно, как было, и ничего не сломает)

Vector

```
void reserve(size_t n) {
    if (n <= cap) return;

    //T* newarr = alloc.allocate(n); // WHY??
    T* newarr = AllocTraits::allocate(Alloc, n);

    size_t i = 0;
    try {
        for (; i < sz; ++i) {
            //AllocTraits::construct(alloc, newarr + i, arr[i]);
            AllocTraits::construct(alloc, newarr + i, std::move(arr[i]));
        }
    } catch(...) {
        for (size_t j = 0; j < i; ++j){
            AllocTraits::destroy(alloc, newarr + j);
        }
        AllocTraits::deallocate(newarr, n);
        throw;
    }

    for (size_t i = 0; i < sz; ++i) {
        AllocTraits::destroy(alloc, arr + i);
    }
    AllocTraits::deallocate(arr, n);
    arr = newarr;
    cap = n;
}
```

```
Vector (const Vector& other){
    reserve(other.cap);
    sz = other.sz;

    size_t i = 0;
    try {
        for (; i < sz; ++i) {
            AllocTraits::construct(alloc, arr + i, other.arr[i]);
        }
    } catch(...) {
        for (size_t j = 0; j < i; ++j){            AllocTraits::destroy(alloc,
   arr + j);
        }
        AllocTraits::deallocate(arr, cap);
        throw;
    }
}
```

```
Vector (Vector&& other){
    reserve(other.cap);
    sz = other.sz;

    other.cap = 0;
    other.sz = 0;

    size_t i = 0;
    try {
        for (; i < sz; ++i) {
            AllocTraits::construct(alloc, arr + i, std::move(other.arr[i]));
        }
    } catch(...) {
        for (size_t j = 0; j < i; ++j){
            AllocTraits::destroy(alloc, arr + j);
        }
        AllocTraits::deallocate(arr, cap);
        throw;
    }
}
```

```
Vector& operator = (const Vector& other){

    if(this != addressof(other)){
        reserve(other.cap);
        sz = other.sz;

        size_t i = 0;
        try {
            for (; i < sz; ++i) {
                AllocTraits::construct(alloc, arr + i, other.arr[i]);
            }
        } catch(...) {
            for (size_t j = 0; j < i; ++j)
                AllocTraits::destroy(alloc, arr + j);
            AllocTraits::deallocate(arr, cap);
            throw;
        }
    }
    return *this;
}
```

```
Vector& operator = (Vector&& other){

    if(this != addressof(other)){
        reserve(other.cap);
        sz = other.sz;

        other.cap = 0;
        other.sz = 0;

        size_t i = 0;
        try {
            for (; i < sz; ++i) {
                AllocTraits::construct(alloc, arr + i, std::move(other.arr[i]));
            }
        } catch(...) {
            for (size_t j = 0; j < i; ++j)
                AllocTraits::destroy(alloc, arr + j);
            AllocTraits::deallocate(arr, cap);
            throw;
        }
    }
    return *this;
}
```

```
    void push_back(const T& value) {
        if (sz == cap)
            reserve(2 * cap);
        //new(arr + sz) T(value);
        AllocTraits::construct(alloc, arr + sz, value);
        ++sz;
    }
```

Проблема с гарантиями безопасности в map/unordered_map

Допустим, мы начали перестраивать дерево/хэш таблицу, и у нас выскочило исключение. В такой ситуации контейнер не может вернуть все в первоначальное состояние из-за того, что это весьма затруднительное мероприятие в силу внутреннего устройства этих контейнеров