Exercice sur les allocateurs pour la STL

Créer un allocateur

```
template<typename T>
struct Alloc {
    using value_type = T;

    Alloc() noexcept {};
    template<class U> Alloc(const Alloc<U>&) noexcept {}

    template<class U> auto operator==(const Alloc<U>&) const noexcept { return true; }
    template<class U> auto operator!= (const Alloc<U>&) const noexcept { return false; }
```

Qui possède les méthodes d'allocation et désallocation :

```
auto allocate(const size_t n) const { ... }

auto deallocate(T* const p, size_t t) const { ... }
```

Implémentées comme vous le souhaitez mais par exemple :

```
auto allocate(const size_t n) const -> T* {
    if (n == 0) return nullptr;
    if (n > std::numeric_limits<size_t>::max() / sizeof(T)) {
        throw std::bad_array_new_length{};
    }
    void *  const pv = malloc(n * sizeof(T));
    if (pv == nullptr) {
        throw std::bad_alloc{};
    }
    std::cout << " allocate " << n << std::endl;
    return static_cast<T*>(pv);
}

auto deallocate(T* const p, size_t t) const noexcept -> void {
    free(p);
    std::cout << "deallocate " << t << std::endl;
}
```

Utiliser l'allocateur sur par exemple un vecteur :

```
auto main() -> int {
    std::vector<int,Alloc<int>> v;
    v.push_back(42);
```