4.17 Понятие аллокатора. Зачем нужны аллокаторы? Реализация основных методов std::allocator. Оператор placement new, его синтаксис использования и отличие от обычного new. Структура std::allocator_traits, ее предназначение. Структура rebind, ее предназначение.

Оператор placement new, его синтаксис использования и отличие от обычного new

Напоминание: Если выделен какой-то кусок памяти под S, но конструктор на нем еще не было вызван, то есть синтаксис чтобы направить конснhуктор на уже выделенную память по указателю

```
    S* p = reinterpret_cast<S*>(operator new(sizeof(S))); // сырая память
    S* p1 = reinterpret_cast<S*>(operator new(sizeof(S)));// сырая память
    new(p) S(); //default construct will be called
    new (p1) S(value); //construct from value



    void* operator new(size_t , S* p){ // new for placement new
        return p;
    }
```

Замечание: если в структуре переопределен оператор new, то placement new для него сам по себе не генерируется

В чем отличие?

Обычный new выделает память, а потом вызывает на этой памяти конструктор, а placement new не выделяет память, а просто вызывает конструктор
Замечание: placement delete не существует

Понятие аллокатора. Зачем нужны аллокаторы? Реализация основных методов std::allocator

Оператор new - довольно низкоуровневая абстракция, поэтому чтобы работать на более высоком уровне, на уровне языка программы, а не на уровне операционной системы, придумали выделять и владеть памятью в виде класса. Аллокатор "стоит" между контейнером и оператором new.

Стандартный аллокатор

```
    template<typename  T>
    struct allocator{
        T* allocate(size_t n) {
            return::operator_new(n * sizeof(T));
        }

        void deallocate(T* ptr, size_t n) {
            ::operator delete(ptr);
        }


        template<typename... Args>
        void construct(T* ptr, const Args&... args){
            new(ptr) T(args...);
        }
```

```
        void destroy(T* ptr){
            ptr->~T();
        }
    }
```

allocate - выделяет нужное число байт
deallocate - очищает память по указателю
construct - конструирует объект(вызывает конструктор) на том месте, куда указывает указатель(просто вызывается placement new)
destroy - вызывает деструктор у объекта, лежащего по указателю

Структура std::allocator_traits, ее предназначение

std::allocator_traits создан для того, чтобы некоторые вещи доопределить за аллокатор, так как некоторые методы в практически всех аллокаторах делают одно и то же. Например с методом construct: если в аллокаторе он определен, то вызывается он, иначе вызовется метод, определенный в allocator_traits. Структура со статическими методами - ссылка на аллокатор и то что нужно передать аллокатору (из-за того что там только методы, то нельзя создать объект этого класса)

Структура rebind, ее предназначение.

Если тип того что надо выделять на алокаторе совпадает с типом шаблонного параметра, то проблем нет. Однако в таком контейнере как лист, это не выполняется. С C++17 определен в allocator_traits. По сути, подменяет один шаблонный параметр на другой

```
template<typename T, typename Alloc = std::allocator<T>>
class list{
    class Node{};
    typename std::allocator_traits<Alloc>::template rebind_alloc<Node> alloc;
public:
    list(const Alloc& alloc = Alloc()) : alloc(alloc) {};
};

// realization
template< class U >
struct rebind {
    typedef allocator<U> other;
};
```