

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 был вызван, то есть синтаксис чтобы направить конструктор на уже выделенную память по указателю

```
1  S* p = reinterpret_cast<S*>(operator new(sizeof(S))); // сырая память
2  S* p1 = reinterpret_cast<S*>(operator new(sizeof(S))); // сырая память
3  new(p) S(); //default construct will be called
4  new (p1) S(value); //construct from value
5
6
7
8  void* operator new(size_t, S* p){ // new for placement new
9      return p;
10 }
```

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

В чем отличие?

Обычный new выделяет 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

```
1  template<typename T>
2  struct allocator{
3      T* allocate(size_t n) {
4          return::operator_new(n * sizeof(T));
5      }
6
7      void deallocate(T* ptr, size_t n) {
8          ::operator delete(ptr);
9      }
10
11
12     template<typename... Args>
13     void construct(T* ptr, const Args&... args){
14         new(ptr) T(args...);
15     }
```

```

16         void destroy(T* ptr){
17             ptr->~T();
18         }
19     }

```

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

```

1     template<typename T, typename Alloc = std::allocator<T>>
2     class list{
3         class Node{};
4         typename std::allocator_traits<Alloc>::template rebind_alloc<Node> alloc;
5     public:
6         list(const Alloc& alloc = Alloc()) : alloc(alloc) {};
7     };
8
9     // realization
10    template< class U >
11    struct rebind {
12        typedef allocator<U> other;
13    };

```