extra9. Реализация функции allocate_shared с поддержкой нестандартного аллокатора. Каким образом shared_ptr поддерживает нестандартные аллокаторы, каким образом происходит удаление объекта и контрольного блока в этом случае? Как поддержка нестандартного аллокатора может одновременно сочетаться с поддержкой нестандартного Deleter?

Постановка проблемы: мы хотим, чтобы `shared_ptr` работал с кастомным аллокатором (да и вообще, обращение к new - плохой кодстайл).

Поэтому, нам нужно научиться этот аллокатор где-то хранить. Очевидно, это нельзя делать внутри полей. Идея - та же что и с нестандартным делитером - нам нужно использовать type erasure для аллокатора. Но есть небольшая проблема - мы этот аллокатор должны хранить, и одновременно им освобождать выделенную под него память. Кажется, это решается следующим способом: сначала делаем копию аллокатора, а потом этим новым аллокатором делаем `deallocate` всего чего нужно.

```
1
2  struct BaseAllocator {
3    virtual void dealloc(void *ptr) = 0;
4    virtual bool TAllocated() = 0;
5    virtual ~BaseAllocator() = default;
6  };
7
8  template<typename T, typename U = std::allocator<char>>
9  struct AllocatorWithNotCstyle : BaseAllocator {
10   U allocator;
11   AllocatorWithNotCstyle(U allocator) : allocator(allocator) {}
12   void dealloc(void *ptr) override {
13     using right_allocator_type = typename std::allocator_traits<U>::template
       rebind_alloc<char>;
14     right_allocator_type right_allocator = allocator;
15     //тут происходит копия аллокатора
16
17     using right_traits = std::allocator_traits<right_allocator_type>;
18
19     //кажется, тут еще должен быть destroy аллокатора allocator, а именно:
20     //U new = allocator; - копируем аллокатор
21     //typename std::allocator_traits<U>::destroy(new, &allocator);
22
23     right_traits::deallocate(right_allocator,
24                              reinterpret_cast<char *>(ptr),
25                              sizeof(T) + sizeof(BaseDeleter) + sizeof(
       BaseAllocator) + 2 * sizeof(size_t));
26   }
27   bool TAllocated() override {
28     return 1;
29   }
30 };
31
32 template<typename T, typename U = std::allocator<char>>
33 struct AllocatorWithCStyle : BaseAllocator {
34   U allocator;
35   AllocatorWithCStyle(U allocator) : allocator(allocator) {}
```

```
  void dealloc(void *ptr) override {
    using right_allocator_type = typename std::allocator_traits<U>::template
     rebind_alloc<char>;
    right_allocator_type right_allocator = allocator;

    //тут нужно добавить то же что и выше

    using right_traits = std::allocator_traits<right_allocator_type>;
    right_traits::deallocate(right_allocator,
                             reinterpret_cast<char *>(ptr),
                             sizeof(BaseDeleter) + sizeof(BaseAllocator) + 2
    * sizeof(size_t));
  }
  bool TAllocated() override {
    return 0;
  }
};
```

В этом коде, к сожалению, отсутствует Control Block, но что уж есть.

```
template<typename T, typename Allocator, typename... Args>
SharedPtr<T> allocateShared(const Allocator &alloc, Args &&... args) {
  using charAllocatorType = typename std::allocator_traits<Allocator>::
   template rebind_alloc<char>;
  using TAllocatorType = typename std::allocator_traits<Allocator>::template
    rebind_alloc<T>;
  charAllocatorType charAllocator = alloc;
  TAllocatorType TAllocator = alloc;
  char *ControlBlock = std::allocator_traits<charAllocatorType>::allocate(
   charAllocator,

   sizeof(T) + sizeof(BaseDeleter)

     + sizeof(BaseAllocator)

     + 2 * sizeof(size_t));
  std::allocator_traits<TAllocatorType>::construct(TAllocator,
                                                   reinterpret_cast<T *>(
   ControlBlock),
                                                   std::forward<Args>(args)
   ...);
  auto deleter_ = reinterpret_cast<BaseDeleter *>(ControlBlock + sizeof(T));
  auto allocator_ = reinterpret_cast<BaseAllocator *>(ControlBlock + sizeof(
   T) + sizeof(BaseDeleter));
  new(deleter_) DeleterWithAllocator<T, Allocator>(TAllocator);
  new(allocator_) AllocatorWithNotCstyle<T, Allocator>(TAllocator);
  return SharedPtr<T>(ControlBlock);
}
```

Тут просто происходит следущее: для аллокатора мы применям ровно ту же технику, что и для делитера в предыдущем билете. Единственное отличие заключается в том, что мы должны этим же аллокатором себя почистить, для этого просто делаем копию