OTUS

ES.1: Prefer the standard library to other libraries and to ”handcrafted code”

Дмитрий Шебордаев, Ведущий специалист

OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

Дмитрий Шебордаев, Ведущий специалист

Введение

Allocators

Дмитрий Шебордаев

Ведущий специалист

Сегодня

- Умные указатели
- Что такое аллокатор
- Аллокатор для контейнера
- ДЗ

OTUS

unique_ptr

Указатель на объект

```
std::unique_ptr<int> p{new int{42}};
auto p = std::make_unique<int>(12);
```

Handler

```
int *init() {};

done(int *) {};

std::unique_ptr<int, deleter> j{init()};
```


Дмитрий Шебордаев, Ведущий специалист

shared_ptr

- можно копировать
- перемещать дешевле
- всегда хранит два указателя
- std::make_shared лучше
- счетчики потоко безопасны
- можно создать из unique_ptr, но не наоборот

Управление памятью

- куча
- счетчики
- сборка мусора

Куча

```
void* malloc(std::size_t size);
void free(void* ptr);
```


Куча и C++

```
new(p) MyClass();
p->~MyClass();
```


Куча и C++

```
p = new MyClass();
delete p;
```


Аллокатор

- переопределить malloc
- переопределить operator new
- переопределить malloc для контейнера

Аллокатор

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


Аллокатор

```
T* allocate(std::size_t n);
void deallocate(T* p, std::size_t n);
```


Аллокатор

```
T* allocate(std::size_t n)
{
    auto p = std::malloc(n * sizeof(T));
    if (!p)
      throw std::bad_alloc();
    return reinterpret_cast<T *>(p);
};
```


Аллокатор

```
void deallocate(T* p, std::size_t n)
{
    std::free(p);
};
```


Аллокатор

```
template<typename U, typename ... Args >
void construct(T* p, Args&& ... args);
void destroy(T* p);
```


Аллокатор

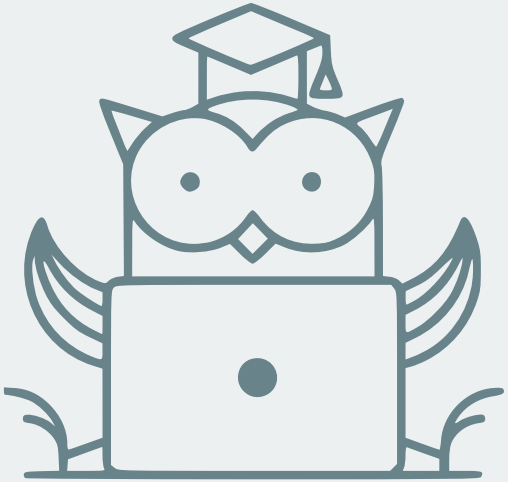
```
template<typename U, typename ... Args>
void construct(U* p, Args&& ... args)
{
    new(p) U(std::forward<Args>(args) ... );
};
```


Аллокатор

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


почитать

- http://en.cppreference.com/w/cpp/memory/allocator
- http://www.ozon.ru/context/detail/id/3829080/



Спасибо за внимание!