Lec 15

В лабе от нас ждут реализацию VLA)

Concept

🔨 Пример — СВОЙ `concept`

```cpp
#include <iostream>
#include <concepts> // std::integral, std::same_as и т.д.

template<typename T>
concept HasSize = requires(T x) {
    { x.size() } -> std::convertible_to<std::size_t>;
};
```

🔍 Это означает: тип `T` должен иметь метод `.size()`, и результат должен быть приводим к `std::size_t`.

Требования контейнера

C++ named requirements: *Container*

A *Container* is an object used to store other objects and taking care of the management of the memory used by the objects it contains.

Requirements

Given the following types and values:

Type	Definition
T	an object type
C	a container class containing objects of type T
Value	**Definition**
`u`, `v`	values of type C or `const C`
`mv`	a value of type C
`cv`	a value of type `const C`
`lhs`, `rhs`	lvalues of type C
`i`, `j`	values of type `C::iterator` or `const C::iterator`

C satisfies the requirements of *Container* if the following types, statements, and expressions are well-formed and have the specified semantics:

Types

Type	Definition	Requirements
`typename C::value_type`	T	T is *CopyConstructible* (until C++11) *Erasable* from C (since C++11).
`typename C::reference`	T&	No explicit requirement
`typename C::const_reference`	`const T&`	
`typename C::iterator`	an iterator type	• `C::iterator` is a *LegacyForwardIterator*, and its value type is T. • `C::iterator` is convertible to `C::const_iterator`.
`typename C::const_iterator`	a constant iterator type	`C::const_iterator` is a *LegacyForwardIterator*, and its value type is T.
`typename C::difference_type`	a signed integer type	`C::difference_type` is the same as the difference type of `C::iterator` and `C::const_iterator`.
`typename C::size_type`	an unsigned integer type	`C::size_type` is large enough to represent all non-negative values of `C::difference_type`.

https://en.cppreference.com/w/cpp/named_req/Container

🤖 Зачем это нужно?

Чтобы алгоритмы из STL и шаблонные функции (например `std::sort`, `std::copy`) могли работать с **любыми контейнерами**, которые **удовлетворяют этим требованиям**, даже если это твой кастомный тип.

например, как `std::vector`, `std::list`, `std::map` и т.п.

Уточняющие требования

https://en.cppreference.com/w/cpp/named_req/SequenceContainer

C

```
sdaf
;Lofdsjkopgfiwmjm
main() {
```

```
int a;
}
```

C++

```
```C
sdaf
;Lofdsjkopgfiwmjm
main() {
int a;
}
```

```
int a = 0;
```

```
#include <concept>
#include <type_traits>
#include <memory> || <cstring>

template <typename T>
    Мы хотим проверить что integer и чётное
concept OddNameConcept = std::is_integral<T>::value && requires (T a
{
    a % 2 == 0; // Проверка на чётноесть
}

template <typename T> requires OddNameConcept <T> // Шаблон соответс
class oddNumberContainer
{
public:
    OddNumberContainer() = deafault; ТО же самое что и OddNumberCont

    // Оператор копирования
    OddNumberContainer(const OddNumverContainer & other) : m_data
    (nullptr), m_size(other.m_size), m_capacity(other.m_capacity)
    {
        T* m_data = new T[m_capacity];
        std::memcpy(m_data, other.m_data, sizeof(T) * m_size);
    }

    // Оператор копирующего присваивания
```
```

```
    OddNumberContainer& operator=(const OddNumverContainer& other)
    {
        //copy and swap
        if (this != other)
        {
            OddNumberContainer tmp(other)
            tmp.swap(*this);
        }
        return *this;
    }

    // Мы явно говорим чтоне будем их реализовывать
    OddNumberContainer(OddNumberContainer&&) = delete;
    OddNumberContainer& operator=(OddNumberContainer&&) = delete;

    void swap(OddNumberContainer& other)
    {
        std::swap(m_data, other.m_data);
        std::swap(m_size, other.m_size);
        std::swap(m_capacity, other.m_capacity);
    }

private:
    T* m_data = nullptr;
    size_t m_size, m_capacity; // m_capacity - макс длинна массива н

    ~OddNumberContainer() noexcept {
        if (m_data != nullptr)
            delete[] m_data;
    }

}
```

GTest

Обязательно:

```
Обязательно:
#include <gtest/gtest.h>

TEST(TestCaseName, TestName) {
    EXPECT_EQ(1, 1);
    EXPECT_TRUE(true);
}
```

```cpp
int main(int argc, char* argv[])
{
    ::testing::InitGoogleTest(&argc, argv); // Запуск тестов
    return RUN_ALL_TESTS();
}
```