25. unique_pointer, shared_pointer, weak_pointer, custom deleters, their usages.

Умные указатели в C++ (smart pointers) предоставляют автоматическое управление памятью, уменьшая вероятность утечек памяти и появления висячих указателей. Наиболее часто используемые умные указатели: `std::unique_ptr`, `std::shared_ptr` и `std::weak_ptr`.

их библиотека - < memory >

std::unique_ptr

Описание:

`std::unique_ptr` — это умный указатель, обеспечивающий **исключительное владение** динамически выделенным объектом. Когда объект `std::unique_ptr` выходит из области видимости, управляемый им объект автоматически удаляется. Но мы не можем создавать его копию, он будет в единственном экземпляре.

Ключевые особенности:

- Гарантирует, что объект принадлежит только одному владельцу.
- Нельзя копировать, можно только перемещать (для передачи владения).

Основные методы:

- `get()`: Возвращает сырой указатель без передачи владения. *правило*: то, что вернуло get() удалять нельзя, тк получим двойное удаление памяти
- `release()`: Освобождает владение и возвращает сырой указатель.
- `reset()`: Удаляет текущий объект и, при необходимости, начинает управлять новым.

```
#include <iostream>
#include <memory>

int main() {
	// так можем создать указатель на массив
	std::unique_ptr<int[]> ptr(new int[5]);
```

```
       // так можем создать указатель на значение
       std::shared_ptr<int> ptr(new (5));

    // так тоже, уже используя make_unique
    std::unique_ptr<int> ptr = std::make_unique<int>(42);
    std::cout << "Значение: " << *ptr << std::endl;

//когда же нужно использовать этот указатель в нескольких местах есть пара вариантов:
       1. использовать функцию get()

int* copy_ptr;
{
       std::unique_ptr<int> ptr(new int(5));
       ptr_copy = ptr.get();
}
std::cout << *ptr_copy;

- Внутри блока создаётся std::unique_ptr<int> ptr, который управляет динамически выделенным объектом int со значением 5.
- Метод ptr.get() возвращает сырой указатель (int*) на объект, которым управляет ptr, и этот указатель сохраняется в copy_ptr.
- После выхода из блока std::unique_ptr<int> ptr уничтожается, а вместе с ним освобождается динамически выделенная память.
- После завершения блока, на который распространяется область видимости ptr, объект, на который указывает copy_ptr, больше не существует, так как std::unique_ptr автоматически уничтожает управляемый объект. Это делает указатель copy_ptr висячим (dangling pointer)

### **Какие могут быть последствия:**
1. Если попытаться использовать `copy_ptr` за пределами блока, это приведёт к неопределённому поведению, так как указатель ссылается на освобождённую память.
2. Это нарушает ключевую концепцию `std::unique_ptr` — безопасное управление памятью без утечек.


!!! правило: то, что вернуло get() удалять нельзя, тк получим двойное удаление памяти
```

```
        2. использовать shared_ptr (об этом ниже)

    // Передача владения
    std::unique_ptr<int> ptr2 = std::move(ptr);
    if (!ptr) {
        std::cout << "ptr теперь null" << std::endl;
    }

    return 0;
}
```

Применение:

- Используйте, когда объект должен иметь только одного владельца.
- Идеально подходит для управления динамически выделенными объектами без разделения владения.
- Можем использовать когда точно знаем когда данные создаются и удаляются, и где начало создания этих данных и где конец.

std::shared_ptr

Описание:

`std::shared_ptr` — это умный указатель, который обеспечивает **разделённое владение** динамически выделенным объектом. Объект удаляется, когда последний `shared_ptr`, владеющий им, уничтожается или сбрасывается, соответственно counter достигает 0.

Ключевые особенности:

- Для управления временем жизни объекта используется подсчёт ссылок, за что мы платим небольшую цену в производительности
- Несколько экземпляров `shared_ptr` могут совместно владеть одним объектом.

Основные методы:

- `use_count()`: Возвращает количество `shared_ptr`, разделяющих владение объектом.
- `reset()`: Освобождает владение, уменьшая счётчик ссылок.

```
#include <iostream>
#include <memory>

int main() {
    std::shared_ptr<int> ptr1 = std::make_shared<int>(42);
    std::shared_ptr<int> ptr2 = ptr1;  // Разделённое владение

    std::cout << "Значение: " << *ptr1 << ", Счётчик ссылок: " <<
ptr1.use_count() << std::endl;

    ptr2.reset();
    std::cout << "После сброса, Счётчик ссылок: " << ptr1.use_count() <<
std::endl;

    return 0;
}
```

Может возникнуть ситуация, когда counter не достигнет 0, например, при наличии циклических зависимостей. Для решения этой проблемы будем использовать weak_ptr

std::weak_ptr

Описание:

`std::weak_ptr` — это умный указатель, который предоставляет **ссылку без владения** на объект, управляемый `std::shared_ptr`. Он не влияет на счётчик ссылок.

Ключевые особенности:

- Помогает избежать циклических ссылок при разделённом владении.
- Может быть преобразован в `shared_ptr`, если объект ещё существует.

Основные методы:

- `lock()`: Создаёт `shared_ptr`, если объект ещё управляется.
- `expired()`: Проверяет, был ли объект уничтожен.

```
#include <iostream>
#include <memory>

int main() {
```

```
    // можно присвоить shared_ptr к weak
    std::shared_ptr<int> sp = std::make_shared<int>(42);
    std::weak_ptr<int> wp = sp;  // Ссылка без владения

    if (auto locked = wp.lock()) {  // Преобразование в shared_ptr
        std::cout << "Значение: " << *locked << std::endl;
    } else {
        std::cout << "Ресурс больше недоступен" << std::endl;
    }

    sp.reset();
    if (wp.expired()) {
        std::cout << "Ресурс был уничтожен" << std::endl;
    }

    return 0;
}
```

Пользовательские удалители

Описание:

Пользовательский удалитель позволяет разработчику определить, как освобождается ресурс при уничтожении умного указателя.

Ключевые особенности:

- Полезно для управления ресурсами, отличными от памяти (например, файловыми дескрипторами, сокетами).
- Удалители передаются как параметры в умные указатели.

Пример с `std::unique_ptr`:

```
#include <iostream>
#include <memory>
#include <cstdio>

int main() {
    auto fileCloser = [](FILE* file) {
        if (file) {
            std::cout << "Закрытие файла" << std::endl;
            fclose(file);
```

```
        }
    };

    std::unique_ptr<FILE, decltype(fileCloser)> file(fopen("example.txt", "w"),
fileCloser);
    if (file) {
        fprintf(file.get(), "Привет, файл!");
    }

    return 0;
}
```

Пример с `std::shared_ptr`:

```
#include <iostream>
#include <memory>

struct Resource {
    ~Resource() { std::cout << "Ресурс уничтожен" << std::endl; }
};

int main() {
    auto deleter = [](Resource* res) {
        std::cout << "Вызов пользовательского удалителя" << std::endl;
        delete res;
    };

    std::shared_ptr<Resource> sp(new Resource, deleter);

    return 0;
}
```

Применение:

- Используйте, если требуется специальная логика очистки для файлов, сокетов или других нестандартных ресурсов.

Сравнительная таблица

Особенность	`std::unique_ptr`	`std::shared_ptr`	`std::weak_ptr`
Владение	Исключительное	Разделённое	Отсутствует
Подсчёт ссылок	Нет	Да	Да (косвенно)
Копирование	Нет	Да	Да
Циклические зависимости	Невозможно	Возможно	Предотвращает
Пользовательские удалители	Поддерживаются	Поддерживаются	Нет