#### Задание 2. Реализация умного указателя

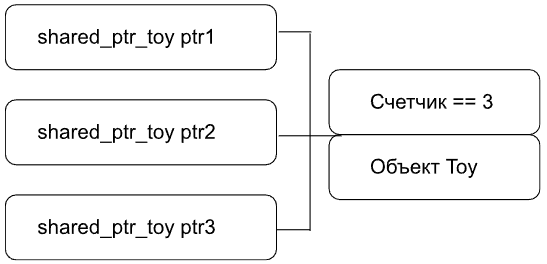
Класс shared\_ptr предоставляет объектам автоматическое управление памятью. Он используется для обеспечения безопасного и эффективного разделения владения объектом между несколькими указателями. Это удобный инструмент, который может решить множество проблем разработчика. Однако, чтобы не совершать ошибок, необходимо отлично знать его устройство.

##### **Что нужно сделать**

1. Реализуйте класс shared\_ptr\_toy — простой аналог умного указателя shared\_ptr, который будет управлять памятью объектов класса Toy, реализованного в предыдущем задании.  
   Указатель должен:
   * иметь все стандартные методы класса,
   * быть функциональной заменой использованию shared\_ptr<Toy>.
2. Реализуйте отдельную сервисную функцию make\_shared\_toy, которая принимает на вход название игрушки и возвращает умный указатель shared\_ptr\_toy на вновь сконструированную игрушку.

##### **Советы и рекомендации**

Чтобы вы правильно выполнили задание, рассмотрим, как должен быть устроен класс shared\_ptr\_toy. Основная идея стандартного shared\_ptr заключается в том, что он подсчитывает количество ссылок на объект и автоматически освобождает память, когда последняя ссылка на объект уничтожается. Это позволяет избежать утечек памяти и ошибок, связанных с некорректным освобождением памяти. Это же поведение нужно реализовать в классе shared\_ptr\_toy.



Изображение: Дмитрий Полухин для Skillbox

Каждый экземпляр shared\_ptr\_toy хранит указатель на объект Toy и указатель на счётчик. За счёт того, что shared\_ptr\_toy хранит указатели, разные экземпляры shared\_ptr\_toy могут ссылаться на одну и ту же игрушку и на один и тот же счётчик.

При конструировании нового объекта shared\_ptr\_toy должен создаваться новый объект Toy и новый счётчик, в который помещается значение 1. Эта функциональность обеспечивается в конструкторе shared\_ptr\_toy. Рекомендуется предусмотреть конструктор с одним параметром «Имя игрушки» для создания экземпляра Toy по имени. Конструктор же по умолчанию (без параметров) создаёт пустой объект shared\_ptr\_toy с нулевыми указателями. Такой объект интересен только в случае присвоения ему другого shared\_ptr\_toy в дальнейшем.

Также требуется реализовать конструктор для конструирования нового объекта shared\_ptr\_toy из другого объекта shared\_ptr\_toy. В этом случае требуется сохранить указатель на Toy и указатель на счётчик, переданный через копируемый shared\_ptr\_toy, а затем увеличить счётчик ссылок на 1. Так мы получим два объекта shared\_ptr\_toy, указывающих на один Toy.

Когда удаляется объект shared\_ptr\_toy, он проверяет количество ссылок на игрушку, и если ссылка всего одна (то есть никто другой не ссылается на текущую игрушку), то он удаляет и сам объект Toy, и счётчик ссылок. Эта функциональность реализуется в деструкторе класса shared\_ptr\_toy.

Помимо этого, нужно обеспечить правильное копирование умных указателей, для чего необходимо переопределить оператор «равно». Алгоритм копирования следующий:

* Если копируемый объект содержит ссылку на тот же объект Toy, что и текущий, то ничего делать не нужно — текущий объект уже содержит ссылку на нужный Toy.
* Если копируемый объект содержит ссылку на другой объект Toy, нужно позаботиться о том, чтобы «освободить» текущую ссылку на Toy, если она есть. Для этого мы выполняем действия, аналогичные действиям в деструкторе, — проверяем количество ссылок на игрушку и, если ссылка всего одна (то есть никто другой не ссылается на текущую игрушку), удаляем объект Toy и счётчик ссылок. Далее нужно сохранить указатель на Toy и указатель на счётчик, переданный через копируемый shared\_ptr\_toy, а затем увеличить счётчик ссылок на 1. В этом случае мы получим два объекта shared\_ptr\_toy, указывающих на один Toy.

В класс shared\_ptr\_toy следует добавить сервисные функции, которые являются обязательной частью стандартного shared\_ptr:

* void reset() — освобождает ссылку на текущий Toy. Функция делает то же самое, что и деструктор, но сам объект shared\_ptr\_toy остаётся доступным. Указатели обнуляются.
* Toy\* get() const — возвращает указатель на Toy, который хранит shared\_ptr\_toy.
* int use\_count() const — возвращает количество ссылок на объект Toy.

Также добавим вспомогательную функцию для проверки нашего класса:

* std::string getToyName() — возвращает имя игрушки, на которую ссылается shared\_ptr\_toy, или слово Nothing, если указатель на Toy пустой.

Функция make\_shared\_toy принимает на вход название игрушки и возвращает умный указатель shared\_ptr\_toy на вновь сконструированную игрушку, используя созданный нами конструктор shared\_ptr\_toy с одним параметром («Имя игрушки»).

Во всех функциях при работе с указателями не забываем занулять указатели и проверять их на nullptr при использовании.

Для проверки созданного вами класса предлагаем следующий код:

int main() {  
   shared\_ptr\_toy toy\_01 = make\_shared\_toy("ball");  
   shared\_ptr\_toy toy\_02(toy\_01);  
   shared\_ptr\_toy toy\_03("duck");  
   std::cout << "=================================================" << std::endl;  
   std::cout << toy\_01.getToyName() << " links:" << toy\_01.use\_count() << "  "  
             << toy\_02.getToyName() << " links:" << toy\_02.use\_count() << "  "  
             << toy\_03.getToyName() << " links:" << toy\_03.use\_count() << std::endl;  
   std::cout << "=================================================" << std::endl;  
   toy\_02 = toy\_03;  
   std::cout << toy\_01.getToyName() << " links:" << toy\_01.use\_count() << "  "  
             << toy\_02.getToyName() << " links:" << toy\_02.use\_count() << "  "  
             << toy\_03.getToyName() << " links:" << toy\_03.use\_count() << std::endl;  
   std::cout << "=================================================" << std::endl;  
   toy\_01.reset();  
   std::cout << toy\_01.getToyName() << " links:" << toy\_01.use\_count() << "  "  
             << toy\_02.getToyName() << " links:" << toy\_02.use\_count() << "  "  
             << toy\_03.getToyName() << " links:" << toy\_03.use\_count() << std::endl;  
   std::cout << "=================================================" << std::endl;  
   return 0;  
}

Если ваш класс реализован правильно, вы получите следующий вывод:

=================================================

ball links:2  ball links:2  duck links:1

=================================================

ball links:1  duck links:2  duck links:2

=================================================

Toy ball was dropped

Nothing links:0  duck links:2  duck links:2

=================================================

Toy duck was dropped