

1) Перечислите все специальные функции-члены класса, включая перемещающие операции

- 1) Конструктор по умолчанию (без аргументов)
- 2) Пользовательские конструкторы (с аргументами)
- 3) Деструктор
- 4) Копирующий конструктор
- 5) Перемещающий конструктор
- 6) Копирующий оператор =
- 7) Перемещающий оператор =

2) Приведите примеры операторов, которые можно, нельзя и не рекомендуется перегружать

Можно: + ; -> ; () ; [] и большинство других, не указанных далее

Не рекомендуется: & ; ,

Нельзя: :: ; . ; *.буквы

3) О каких преобразованиях следует помнить при проектировании операторов?

При проектировании операторов следует помнить о неявных преобразованиях типов. Ключевые приемы: добавление значений по умолчанию для аргументов конструкторов (когда хотим извлечь из них пользу) и прописывание `explicit` (когда хотим избежать их).

4) Опишите классификацию выражений на основе перемещаемости и идентифицируемости

`glvalue` (все идентифицируемые типы) = `lvalue` (идентифицируемый, не перемещаемый) + `xvalue` (идентифицируемый, перемещаемый)

`rvalue` (все перемещаемые типы) = `xvalue` (идентифицируемый, перемещаемый) + `rvalue` (неидентифицируемый, перемещаемый)

5) Зачем нужны `rvalue`-ссылки?

`rvalue`-ссылки нужны для обеспечения перемещения путем их передачи в соответствующие, возможно, перегруженные функции.

6) Почему семантика перемещения лучше копирования?

Если большой набор данных больше не нужен в старом смысле, то гораздо эффективнее передать их во владение другому объекту, нежели выделять новую память, производить копирование и освобождать старые ненужные ресурсы.

7) Что делает функция `std::move` и когда нет необходимости ее вызывать?

`std::move` делает преобразование ссылки к `rvalue`. Практически никогда нет необходимости ее использовать при возврате значений в функциях, поскольку это может только помешать компилятору сделать наиболее выгодные оптимизации.

8) Кем выполняется непосредственная работа по перемещению?

Непосредственную работу по перемещению выполняют операторы / конструкторы / другие функции

9) Когда может потребоваться пользовательская реализация специальных функций-членов класса?

Пользовательская реализация специальных функций-членов класса может потребоваться тогда, когда автоматически генерируемая версия нас не устраивает (например, при копировании массива мы хотим получить не указатель на чужие элементы, а свои собственные элементы с теми же значениями).

10) Для чего нужны ключевые слова `default` и `delete` в объявлении специальных функций-членов класса?

`default` дает команду компилятору реализовать данную функцию по умолчанию. Это может быть важно, если мы заблокировали автоматическую генерацию функций какими-либо своими, но реализация по умолчанию нас вполне устраивает и мы не хотим писать ее с нуля.

`delete` запрещает использование тех или иных функций. Полезно, например, если нужно заблокировать автоматически генерируемые функции или же запретить преобразование типов при вызове функции (вместо преобразования типа и вызова соответствующей новому типу функции для вызова выбирается версия с исходным типом, но ее вызвать не получается по нашей команде).

