Практическое задание

1. Проанализируйте задания предыдущих уроков.
    a. В каких случаях необходима была явная передача указателя в качестве входных параметров и возвращаемых результатов или в качестве приёмника в методах?

    b. В каких случаях мы фактически имеем дело с указателями при передаче параметров, хотя явно их не указываем?

2. Для арифметического умножения и разыменования указателей в Go используется один и тот же символ — оператор (*). Как вы думаете, как компилятор Go понимает, в каких случаях в выражении имеется в виду умножение, а в каких — разыменование указателя?

==========

1a. В общем случае явная передача указателя на объект в какую-либо функцию (аналогично метод с ресивером-указателем) нужна, когда эта функция должна изменить объект. Также передача указателя может служить цели оптимизации программы для меньшего потребления памяти при работе с большими структурами. Плюс на лекции было что-то про работу с БД, когда не очевидно, с какими zero-value данными вернется ответ на запрос.
В предыдущих уроках параметр-указатель явно использовался для организации пользовательского ввода функцией fmt.Scanln. Полагаю, это сделано для того, чтобы сделать единую точку входа с универсальным параметром (интерфейсом). Иначе пришлось бы использовать десяток функций для ввода данных каждого типа: целых чисел, строк, float... Функция strconv.Atoi одним из значений возвращает ошибку типа «указатель».

1.b Неявная передача указателя наблюдается при работе с мапами и функциями. В случае со слайсом передается структура данных, содержащая в себе адрес базового массива. Если модификация слайса не привела к новой аллокации, то изменения в базовом массиве отразятся также на переданном параметре после выхода из функции. Если же внутри функции были добавлены элементы, и для них не хватило емкости, то изменения в слайсе не отразятся на переданном параметре.

2. Решение об интерпретации * принимается в зависимости от контекста.
Компилятор не позволит совершить операцию умножения с переменной типа «указатель» (operator * not defined on pointer). В случае использования умножения и разыменования в одном выражении разыменование имеет более высокий приоритет, поэтому сначала считается значение по указанному адресу, а следующим шагом выполнится умножение:

```
number := 10
pointer := &number
fmt.Println(number * *pointer)   // 100
```