Если не углубляться в тонкости хранения массивов на уровне “железа”, а рассматривать на уровне работы программы, в переменной, “хранящей” массив на самом деле содержится указатель (ссылка) на ячейку памяти, в которой хранится первый элемент и, собственно, физический размер массива (capacity). Действительно, по ссылке на объект мы всегда можем выяснить точное положение этого объекта в памяти, а также количество байт, необходимое для его хранения.

В связи с тем, что элементы хранятся в последовательных ячейках памяти, между индексом произвольного элемента относительно первого элемента существует однозначная линейная связь. Это означает, что можно найти указатель на произвольный объект в памяти через указатель на первый элемент, используя **только операции сложения и умножения на число.** Рассмотрим этот принцип на несколько абстрактном примере. Пусть *M* - указатель на ячейку памяти, в которой хранится первый элемент, а *F* - размер ячейки памяти (в байтах) для одного элемента. Тогда для элемента *N* с индексом *i* (отсчет начинается с нуля) будем иметь указатель на ячейку произвольную *N* ячейку в памяти

N = M + F * i

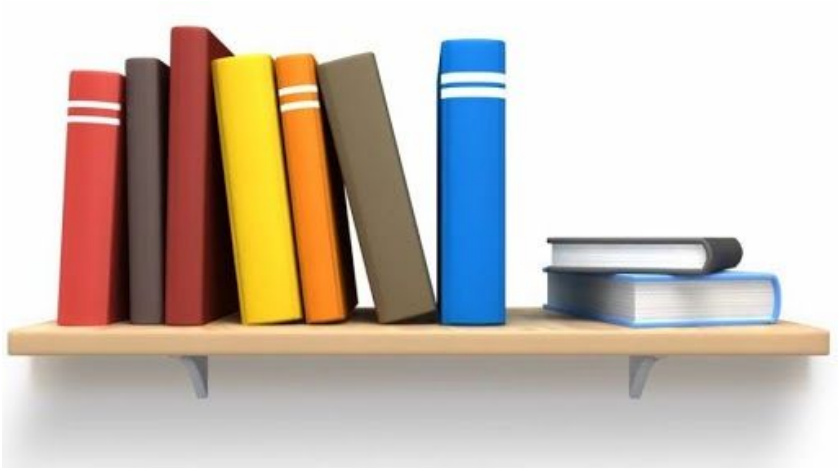
Может быть удивительно, но мы можем складывать указатель и размер ячейки памяти. Это становится понятно, если мы попытаемся заглянуть на эту операцию на уровне “железа”. Указатель - это число (как правило в шестнадцатеричном представлении, например, 0xF510) указывающее порядковый номер байта в памяти, где хранится элемент. Размер ячейки в памяти также представлен в байтах. Положение байта первого элемента “сдвигаем” на индекс ***i***, умноженный на размер ячейки.

Как можно увидеть, используя лишь арифметические операции мы смогли получить указатель на искомую ячейку памяти. С точки зрения алгоритмической сложности, мы можем это сделать за константное время - O(1), ведь нам не требовалось проделывать много операций для нахождения этого указателя - всего лишь две арифметические операции. И более того, количество этих операций никак не зависит от количества элементов в массиве! Точно также и с размером массива - его можно получить за константное время, ведь оно уже хранится в переменной этой структуры данных.

Однако всё становится интереснее, если требуется добавлять или удалять элементы из динамического массива. Рассмотрим несколько возможных ситуаций.

1. **Вставка элемента в конец массива, не выходя за его пределы**. Если мы вставляем элемент и его текущая длина меньше, чем физический размер памяти, то мы просто заполняем выделенную ячейку памяти и делаем это за константное время. Можно представить себе это как книжную полку, в которую

мы просто справа доставляем книги, пока хватает пространства...



2. Если мы хотим поставить книгу на полку, но места уже на ней нет, но перфекционист внутри требует, чтобы книги этой тематики были все в одном месте, мы начинаем их все перемещать на другую полку побольше. Тоже самое происходит, если мы пытаемся **вставить элемент в конец массива, выходя за пределы выделенной памяти.** В памяти происходит тоже самое - все ячейки массива перемещаются в область памяти, которая заведомо больше исходной (например в 2 раза). И для этого нам нужно извлечь каждый элемент и переместить его в памяти, что потребует нескольких действий для каждого элемента. Сложность такого процесса будет - O(n), где n - количество элементов в массиве.
3. Если мы захотим **удалить элемент из конца**, то как и в первом случае всё пройдет гладко - за константное время. А вот если мы захотим **удалить элемент из начала**, то нам придется передвигать все элементы на 1 вправо, что потребует O(n) действий в худшем случае.
4. Аналогично, если нужно **вставить элемент** не в конец, а **в начало или середину** - нам придется “сдвинуть” все элементы вправо, а на освободившееся место вставить новый элемент. И это также потребует O(n) операций.