

Лабораторная работа 07. Динамический массив. Перегрузка операций

1. Построить класс «Вектор» как собственную реализацию типа «динамический массив» для работы с вещественными числами

Для хранения элементов вектора в классе использовать обычный массив.

Обеспечить рациональное использование памяти. Место под массив выделить с некоторым запасом. При добавлении и удалении элементов отслеживать актуальное количество заполненных ячеек. При исчерпании свободного места – переаллоцировать массив с выделением дополнительной памяти (см. справочный материал ниже).

Массив разместить в приватном поле.

Добавить в класс не менее трех конструкторов, переопределить методы `ToString`, `Equals`, `GetHashCode`

В классе обеспечить выполнение операций (через методы, свойства или перегрузку операторов):

- чтение одного элемента по индексу через отдельный `Get`-метод и через индексатор `[]`
- запись элемента по индексу через отдельный `Set`-метод и через индексатор `[]`
- добавление элемента в конец массива
- удаление элемента из конца массива
- вставку элемента по индексу
- удаление элемента по индексу
- поиск элемента по значению (возвращает индекс первого найденного или -1, если элементов нет)
- удаление элемента по значению, удалять все вхождения указанного значения
- возврат максимального значения (индекса)
- возврат актуального количества элементов (метод или свойство)
- очистка всего массива
- сортировка элементов вектора методом `QuickSort` с разделением по схеме Хоара (за `pivot` брать крайний элемент справа)

Для класса «Вектор» перегрузить операторы:

- сравнения на равенство и неравенство
- сложение двух векторов
- инверсия всех значений (через унарный минус)
- умножение двух векторов
- умножение вектора на скаляр
- инкремент и декремент всех значений

В программе

- 1) Протестировать все созданные методы на подходящих примерах.
- 2) Из предварительно подготовленного текстового файла прочитать три набора из нескольких чисел. Разместить их в объектах типа «Вектор».
 - а) Перенести все элементы последнего из них в конец первого.
 - б) Во втором найти максимальный (`max`) и минимальный (`min`) элемент. Удалить из первого вектора все вхождения `max`. Добавить в начало, середину и конец первого `min`.
 - в) Элементы второго вектора отсортировать по возрастанию.
 - г) Сохранить в третьем векторе результат сложения первого и второго.
 - д) Все векторы вывести в новый файл.
- 3) Из предварительно подготовленного текстового файла прочитать матрицу размером (5 x 6) с элементами из [-100, 100].
Получить векторы только с положительными и только с отрицательными элементами этой матрицы. Вывести их на экран и в отдельный текстовый файл.
Инвертировать один из этих векторов и вывести на экран сообщение о том, равны ли они.
- 4) (*доп) Из предварительно подготовленного текстового файла прочитать коэффициенты системы линейных уравнений. В тот же файл добавить последней строкой решение этой системы. Для нахождения корней использовать объекты класса «Вектор» и метод Гаусса.

Справочный материал:

Массив – набор **однотипных элементов**, расположенных в памяти непосредственно друг за другом, доступ к которым осуществляется **по индексу или набору индексов**

Операции

- чтение элемента по индексу за $O(1)$
- запись значения в элемент по индексу за $O(1)$

- добавление элемента за $O(n)$
- удаление элемента за $O(n)$
- прочие (поиск, клонирование, ...)

*определены не для всех типов массивов

Индексы принимают целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона

Размерность массива — количество индексов, необходимое для однозначной адресации элементов в нем (одномерные, двумерные, трёхмерные,...)

- Одномерный массив нестрого соответствует вектору в математике; двумерный - матрице.
- Форма или структура массива — сведения о количестве размерностей и размере (протяжённости) массива по каждой из размерностей; может быть представлена одномерным массивом.

Особенностью массива как структуры данных является **константная вычислительная сложность доступа к элементу массива по индексу** (структура с произвольным доступом)

Фиксированные (статические) массивы и массивы переменного размера

В простейшем случае массив имеет постоянную длину по всем размерностям и хранит элементы данных одного и того же типа (гомогенный), а в качестве индексов выступают целые числа.

0	1	2	3	4	5	6	7
4	2	6	8	1	7	5	3

Типовой способ реализации статического гомогенного массива:

1. Под массив выделяется непрерывный блок памяти объёмом $S * m_1 * m_2 * m_3 ... m_n$, где S — размер одного элемента, а $m_1 ... m_n$ — размеры диапазонов индексов (то есть количество значений, которые может принимать соответствующий индекс).
2. При обращении к элементу массива $A[i_1, i_2, i_3, ..., i_n]$ его адрес вычисляется как

$$B + S * (i_{1p} * m_2 * m_3 * ... * m_n + i_{2p} * m_3 * m_4 * ... * m_n + ... + i_{(n-1)p} * m_n + i_{np}),$$

где B — база (адрес начала блока памяти массива), i_{kp} — значение k -го индекса, приведённое к целому с нулевым начальным смещением.

Преобразование логической структуры массива в физическую (процесс линеаризации)

Порядок следования индексов в формуле вычисления адреса может быть различным. Первый элемент массива, в зависимости от языка программирования, может иметь различный индекс (три основных: с отсчетом от нуля - zero-based, с отсчетом от единицы - one-based и с отсчетом от специфического значения заданного программистом n-based).

Адрес элемента с заданным набором индексов вычисляется так, что время доступа ко всем элементам массива одинаково.

(одинаковость времени доступа следует понимать как отсутствие теоретической зависимости времени доступа от положения элемента и размера массива; в действительности особенности конкретной вычислительной платформы могут дать определённый разброс времени доступа)

Ряд языков поддерживает также **динамические массивы**, длина которых может изменяться по ходу работы программы, и **гетерогенные массивы**, которые могут в разных элементах хранить данные различных типов.

Динамические (саморасширяющиеся) массивы

Динамический массив - это линейная структура данных, которая может увеличиваться и в некоторых реализациях уменьшаться при изменении их размера.

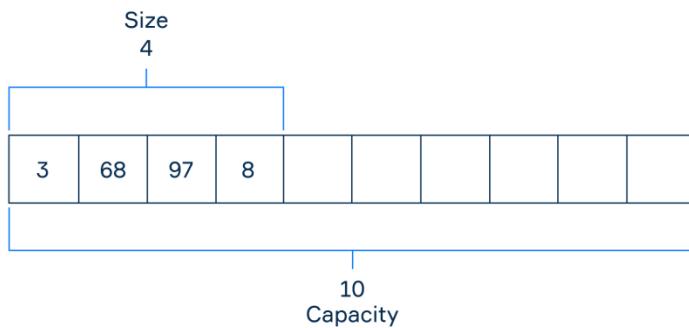
Как правило, он имеет внутренний регулярный массив, который фактически хранит данные и обеспечивает операции над ним.

Динамический массив имеет два важных свойства:

- **размер** (`length`, `size`, `count`) - количество уже сохраненных в нем элементов;
- **емкость** (`capacity`) - возможное количество элементов для хранения, которое соответствует размеру внутреннего регулярного массива.

При первичном создании либо указывают **capacity** для нового динамического массива, либо устанавливают постоянное значение по умолчанию (например, 10).

Пример. Фактический размер 4, а емкость 10 (начальная):



В отличие от базовых массивов, динамические массивы имеют операции для добавления / удаления элементов в любую позицию или из нее.

Коэффициент масштабирования

Если количество элементов превышает **capacity**, все элементы будут скопированы в новый внутренний массив большего размера. Существует ряд различных стратегий масштабирования. Наиболее распространенными являются умножение начальной емкости на 1,5 или 2, реже используются 5/4, 9/8,....

Это компромисс между временем работы и эффективность использования памяти. С большим фактором роста у нас больше вставок, прежде чем нам нужно будет расширять массив, тем самым уменьшая сложность времени.

Также может потребоваться поддержка сжатия внутреннего массива при удалении элементов для уменьшения необходимого размера.

Общие операции

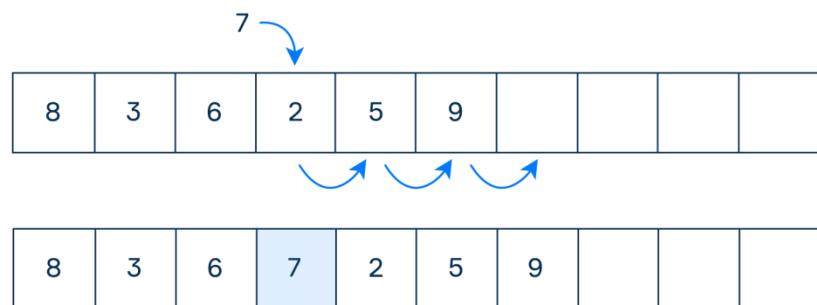
Получить элемент по индексу (чтение). Поскольку динамический массив в основном является обычным массивом, мы можем получить доступ к элементам по их индексу в постоянное время, поэтому сложность $O(1)$.

Запись значения по указанному индексу. Операция заменяет элемент по указанному индексу на элемент. Все это делается за постоянное время, так как это похоже на присваивание в базовом массиве, поэтому сложность $O(1)$.

Добавить элемент. В базовом сценарии, когда мы просто добавляем элемент в конец массива, сложность:

- $O(1)$ - в среднем, поскольку мы просто вставляем элемент в уже выделенную память (меньше емкости);
- $O(n)$ - в худшем случае, когда нам не хватило места и нам нужно выделить новый массив и скопировать в него каждый элемент.

Добавить элемент по указанному индексу. Эта операция используется, когда мы хотим добавить элемент между некоторыми уже размещёнными элементами. Его сложность (как средняя, так и худшая) будет $O(n)$, поскольку при каждой вставке мы должны перемещать элемент по нужному индексу, а затем перемещать каждый элемент на один индекс вправо.



Удалить элемент по значению / индексу. Эти методы либо удаляют первое вхождение указанного элемента, либо элемента с указанным индексом. Оба аналогичны добавлению элемента по указанному индексу в том смысле, что нам нужно было бы переместить некоторые (или все) из оставшихся элементов на один индекс влево; поэтому их сложность также будет $O(n)$.



Очистка массива. Здесь мы просто хотим удалить каждый элемент из массива. Поскольку вставка выполняется с помощью вычислений для текущего размера массива, мы можем просто сбросить размер до нуля и переопределить старые элементы во время следующих вставок. Это привело бы к зависанию элементов в памяти (так что сборщик мусора не сможет их собрать), пока они не будут переопределены. Самая простая форма будет иметь сложности $O(1)$, но правильная $O(n)$.