**Принципы работы контейнера std::vector из библиотеки <vector>**

**Представление в памяти.** Вектор — это динамический массив, который может изменять свой размер при добавлении или удалении дополнительных элементов. У вектора есть методы **size()** – количество реальных объектов (размер), и **capacity()** – количество объектов, под которые зарегистрирована память (ёмкость). capacity всегда больше или равен size, так как вектор резервирует память с запасом.

При создании пустого вектора, память не выделяется (рис.2). Если же вектору задать размер, то он зарезервирует память для указанного size, то есть capacity будет равно size (рис.3).

0

1

2

.size() – 3

.capacity() – 5

Heap

Stack

v

Некоторые методы, предоставляющие доступ к вектору и его элементам:

* **operator[]** – возвращает ссылку на элемент в указанном месте (рис.1).
* **at** – возвращает ссылку на элемент в указанном месте с проверкой границ (рис.3).
* **data** – возвращает указатель на базовый массив, служащий хранилищем элементов (рис.3).
* **front** – возвращает ссылку на первый элемент в контейнере (рис.3).
* **back** – возвращает ссылку на последний элемент в контейнере (рис.3).
* **begin** – возвращает итератор к первому элементу.
* **end** – возвращает итератор за последним элементом.
* **empty** – возвращает true, если вектор пуст (рис.2).
* **reserve** – увеличивает capacity вектора до значения, большего или равного указанного capacity. Если новое значение больше текущего capacity, выделяется новое хранилище, в противном случае функция ничего не делает (рис.4).
* **shrink\_to\_fit** – удаляет неиспользуемую память (рис.4).
* **resize** – изменяет размер контейнера, чтобы он содержал указанное количество элементов (если оно не равно текущему size). Если текущий размер больше указанного, то лишние элементы удаляются, иначе добавляет элементы (по умолчанию или с указанным значением) (рис.5).

**Вставка.** При добавлении элемента если size меньше capacity, он помещается в свободную память. Если size равен capacity, то происходит перераспределение. Используется политика экспоненциального роста, то есть capacity нового блока памяти увеличивается в 2 или 1,5 раза (зависит от компилятора). Старые элементы копируются в новый блок (рис.6). Однако, если размер вектора увеличивается больше, чем в 1,5 раза, память не выделяется с запасом, а capacity равно size (рис.7).

Методы вставки:

* **insert** – вставляет элементы в указанное место контейнера (рис.7).
* **insert\_range** – вставляет в порядке копии элементов из контейнера

перед указанной позицией.

* **emplace** – вставляет новый элемент в контейнер непосредственно перед указанной позицией (в отличие от insert создает объект прямо в векторе).
* **emplace\_back** – добавляет новый элемент в конец контейнера (в отличие от push\_back создает объект прямо в векторе).
* **push\_back** – добавляет заданное значение элемента в конец контейнера (рис.8).

**Удаление.** Удаление элемента не вызывает перераспределение. Удаленный объект будет уничтожен, но память останется принадлежать вектору. Метод shrink\_to\_fit() позволяет освободить лишнюю память. Если удалить все элементы, память будет всё еще занята (рис.8).

Методы удаления:

* **erase** – удаляет указанные элементы из контейнера (в позиции или в диапазоне).
* **pop\_back** – удаляет последний элемент контейнера (рис.8).
* **clear** – удаляет все элементы из контейнера (size будет равен 0) (рис.8).

**Сравнение с TVector.** std::vector использует экспоненциальный рост для минимизации перераспределений, но при вставке в середину может снизиться призводительность, так как нет запаса памяти. В случаях, где он не требуется, отсутствие избыточного выделения памяти будет являться плюсом. TVector же выделяет память с фиксированными запасом, что может привести к частым перераспределениям. Чтобы повысить эффективность TVector можно разработать его различное поведение в зависимости от действий, производимых над вектором (например, при частых перераспределениях памяти увеличивать фиксированный запас памяти).

При удалении в std::vector все элементы сдвигаются, а лишняя память удаляется только с помощью shrink\_to\_fit. В TVector, когда удаляется элемент, его статус помечается “deleted”, перераспределение происходит при превышении порогового значения процента удаленных элементов. Чтобы избежать перераспределения памяти, можно производить сдвиг элементов при вставке в полный массив, таким образом TVector будет работать эффективнее.

Сдвиг элементов в TVector происходит реже, но требуется хранить и проверять статусы элементов во многих операциях. Для оптимизации можно внедрить итераторы, игнорирующие статус “deleted”.

Кроме того, в TVector добавлены методы, позволяющие производить поиск элементов по значению, удобную замену по указателю, перемешивания, а также сортировки, что делает его более функциональным и удобным в специальных проектах.

**Приложение А: проведение эксперимента**

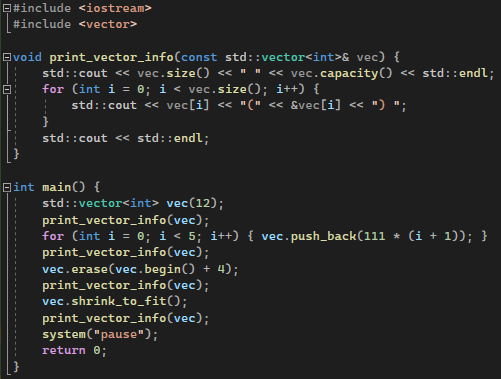
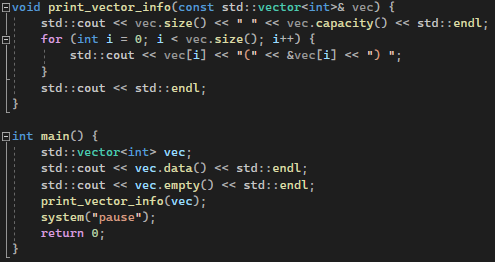




Рис.1. Запуск тестовой программы с выводом адресов

**Приложение Б: проведение эксперимента**



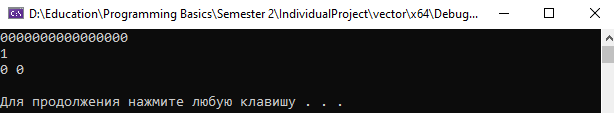
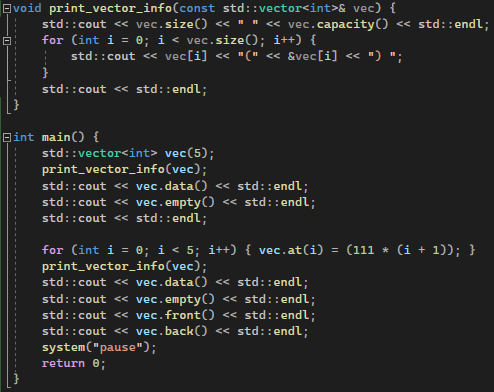


Рис.2. Работа с пустым вектором

**Приложение В: проведение эксперимента**



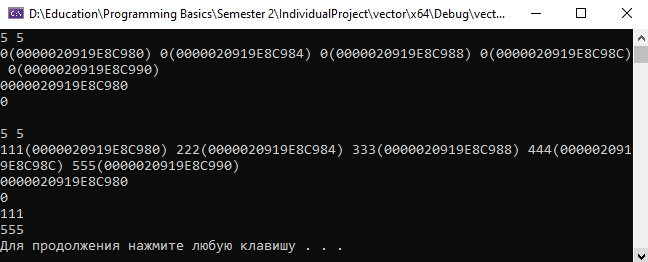
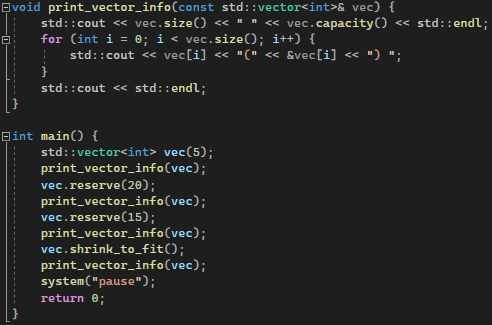


Рис.3. Доступ к элементам вектора

**Приложение Г: проведение эксперимента**



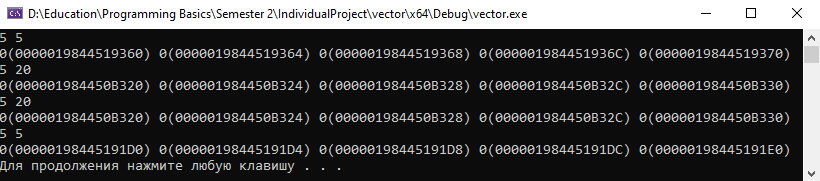
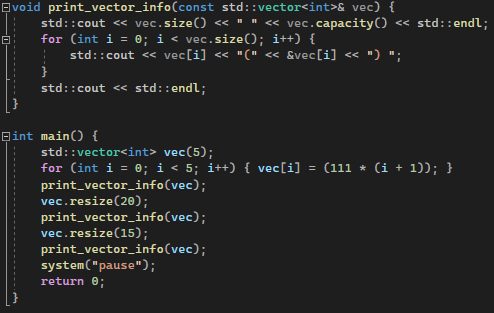


Рис.4. Работа с capacity

**Приложение Д: проведение эксперимента**



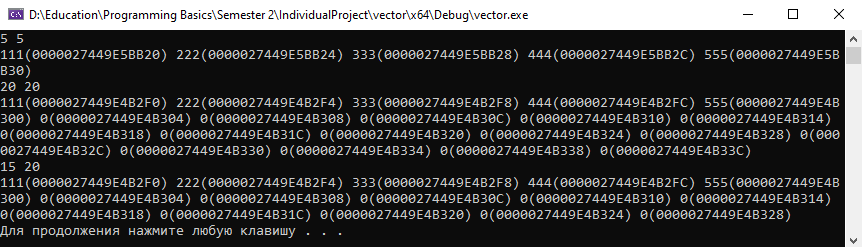


Рис.5. Работа с size

**Приложение Е: проведение эксперимента**



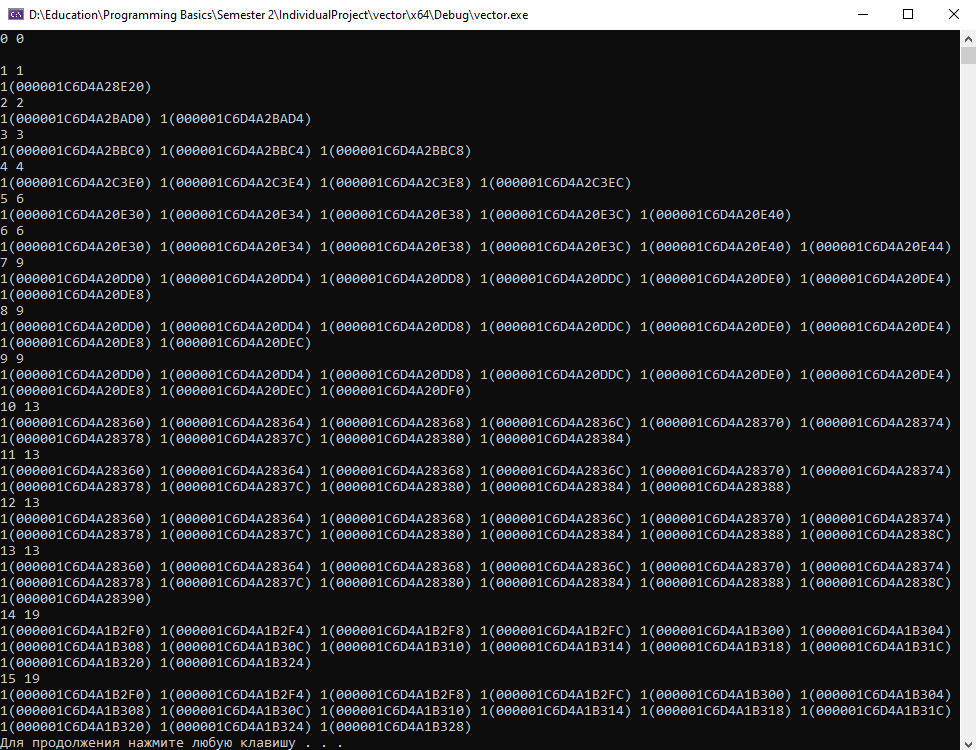
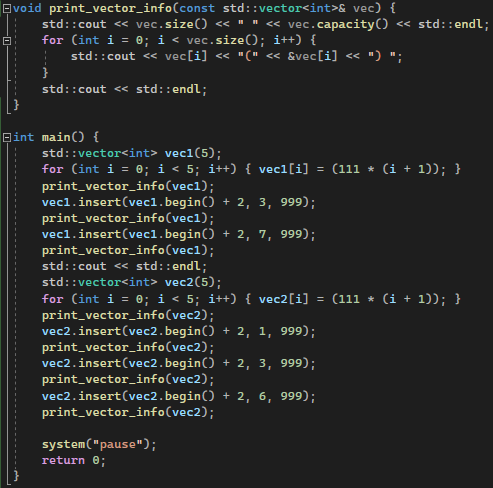


Рис.6. Работа со вставкой

**Приложение Ж: проведение эксперимента**



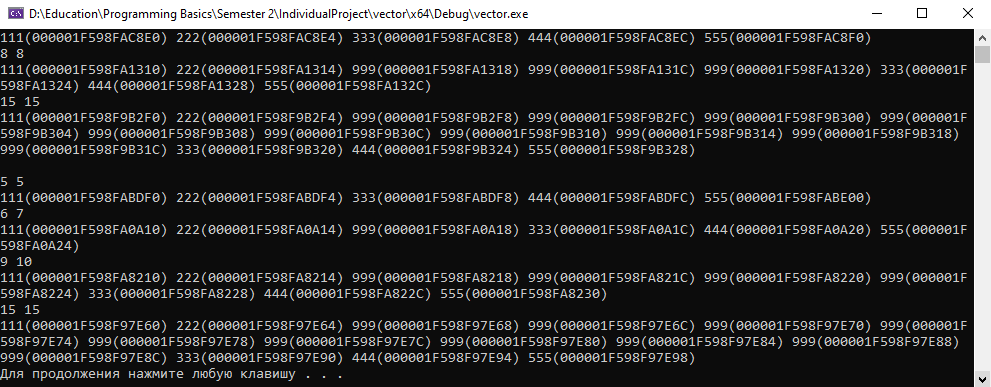
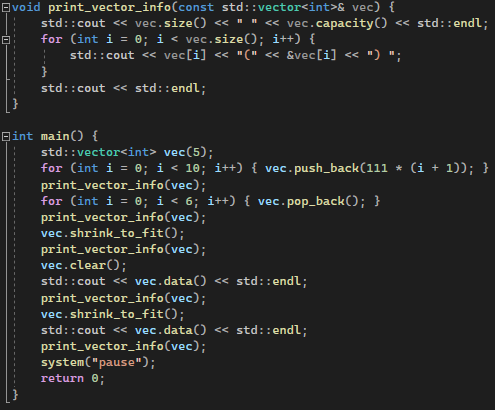


Рис.7. Перераспределение памяти

**Приложение З: проведение эксперимента**



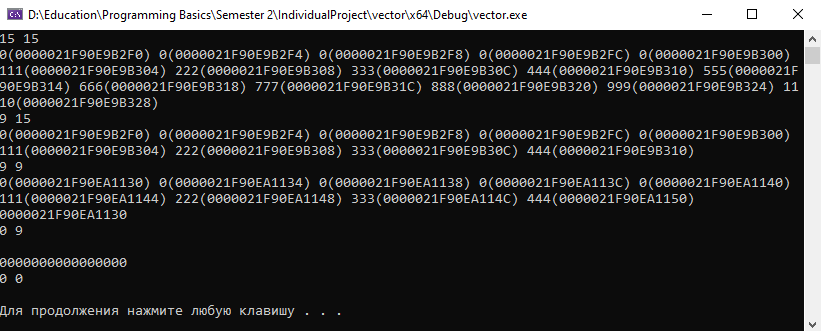


Рис.8. Работа с удалением