**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. Если size равен capacity, то происходит перераспределение. Используется политика экспоненциального роста, то есть capacit нового блока памяти увеличивается в 2 или 1,5 раза (зависит от компилятора). Старые элементы копируются в новый блок.

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.
* **at** – возвращает ссылку на элемент в указанном месте с проверкой границ.
* **data** – возвращает указатель на базовый массив, служащий хранилищем элементов.
* **front** – возвращает ссылку на первый элемент в контейнере.
* **back** – возвращает ссылку на последний элемент в контейнере.
* **begin** – возвращает итератор к первому элементу.
* **end** – возвращает итератор за последним элементом.
* **empty** – возвращает true, если вектор пуст.
* **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.
* **shrink\_to\_fit** – удаляет неиспользуемую память.
* **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).

**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.

Методы вставки:

* **insert** – вставляет элементы в указанное место контейнера.
* **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.

**Удаление.** Удаление элемента не вызывает перераспределение. Удаленный объект будет уничтожен, но память останется принадлежать вектору. Метод shrink\_to\_fit() позволяет освободить лишнюю память.

Методы удаления:

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

**Сравнение с TVector.** std::vector использует экспоненциальный рост для минимизации перераспределений. TVector же выделяет память с фиксированными запасом, что может привести к более частым перераспределениям.

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.

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

**Приложение А: проведение эксперимента**

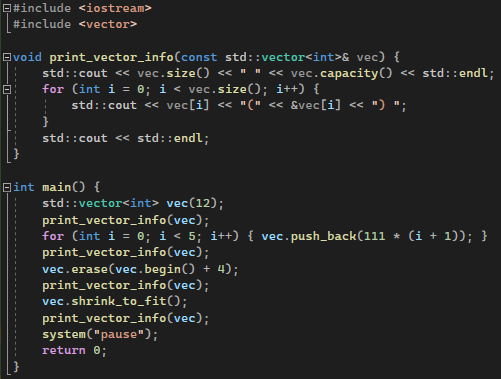




Рис.1. Запуск тестовой программы с выводом адресов