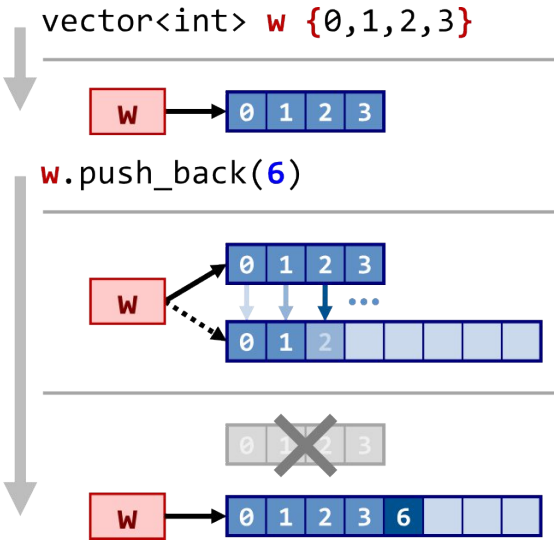
Описание принципов работы std::vector

Вектор — динамический массив, автоматически меняющий свой размер по необходимости.

Представление в памяти:

Allocator выделяет блок памяти capacity для размещения объектов в количестве size. Если памяти недостаточно, запрашивается новый блок, в два раза больше предыдущего, а старый освобождается.

Пример выделения памяти вектором →



Вставка:

Когда элемент вставляется, он копируется в выделенную память и на количество хранимых элементов. Мы можем продолжать вставлять элемент таким образом до тех пор, пока размер не сравняется с емкостью, а это значит,что вектор заполнен. Чтобы вставить больше элементов, необходимо выполнить перераспределение. Сложность (O(1)).

```cpp
#include <iostream>
#include <vector>

void print_vector_info(std::vector<int> vec) {
    std::cout << vec.size() << " " << vec.capacity() << std::endl;
    for (int i = 0; i < vec.size(); i++) {
        std::cout << vec[i] << "(" << &vec[i] << ") ";
    }
    std::cout << std::endl;
}

int main() {
    std::vector<int> vec(6);
    print_vector_info(vec);
    for (int i = 0; i < 5; i++) { vec.push_back(111 * (i + 1)); }
    print_vector_info(vec);
    system("pause");
    return 0;
}
```

Код, представленный выше, выводит размер и ёмкость вектора до и после вставки, а также адреса элементов вектора.

Согласно теории, представленной в документации C++, в результате работы программы size должен быть равен 11, а capacity 12. Однако на деле результаты противоречат документации. Capacity равен size.

```
6 6
0(0000026CD4A7FB30) 0(0000026CD4A7FB34) 0(0000026CD4A7FB38) 0(0000026CD4A7FB3C) 0(0000026CD4A7FB40) 0(0000026CD4A7FB44)
11 11
0(0000026CD4A7DEC0) 0(0000026CD4A7DEC4) 0(0000026CD4A7DEC8) 0(0000026CD4A7DECC) 0(0000026CD4A7DED0) 0(0000026CD4A7DED4)
111(0000026CD4A7DED8) 222(0000026CD4A7DEDC) 333(0000026CD4A7DEE0) 444(0000026CD4A7DEE4) 555(0000026CD4A7DEE8)
Для продолжения нажмите любую клавишу . . .
```

Ситуация не меняется и при бОльших size (capacity должно стать 1000, но оно равно 505).

```
std::vector<int> vec(500);
```

```
505 505
```

Удаление:

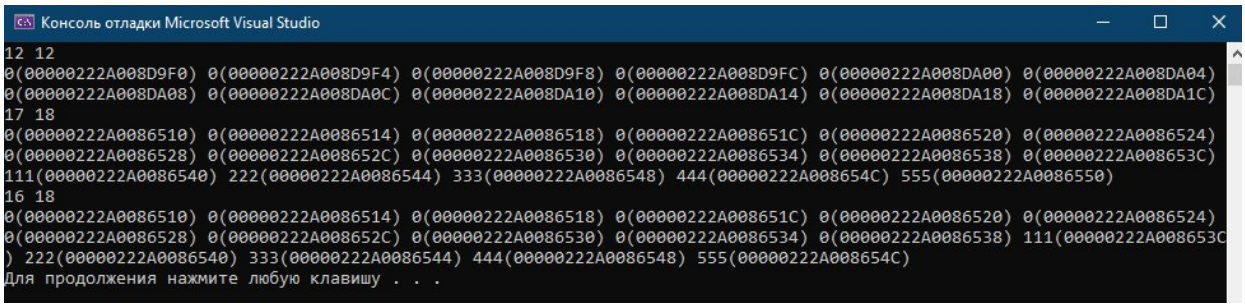
Удаление элементов не приводит к перераспределению. Удаленный объект будет уничтожен, но память останется принадлежать вектору. При удалении элементов возникает та же ошибка. Сложность (O(1)).

```
C:\Users\Ivan\source\repos\Vector_project\x64\Debug\Vector_project.exe
12 12
0(0000025554F3D8A0) 0(0000025554F3D8A4) 0(0000025554F3D8A8) 0(0000025554F3D8AC) 0(0000025554F3D8B0) 0(0000025554F3D8B4)
0(0000025554F3D8B8) 0(0000025554F3D8BC) 0(0000025554F3D8C0) 0(0000025554F3D8C4) 0(0000025554F3D8C8) 0(0000025554F3D8CC)
11 11
0(0000025554F3DC90) 0(0000025554F3DC94) 0(0000025554F3DC98) 0(0000025554F3DC9C) 0(0000025554F3DCA0) 0(0000025554F3DCA4)
0(0000025554F3DCA8) 0(0000025554F3DCAC) 0(0000025554F3DCB0) 0(0000025554F3DCB4) 0(0000025554F3DCB8)
Для продолжения нажмите любую клавишу . . .
```

Однако, если передавать ссылку на оригинал, а не копировать вектор, то удаление работает корректно. Вставка изменяет capacity, но в 1,5 раз вместо 2.

```cpp
void print_vector_info(const std::vector<int> &vec) {
    std::cout << vec.size() << " " << vec.capacity() << std::endl;
    for (int i = 0; i < vec.size(); i++) {
        std::cout << vec[i] << "(" << &vec[i] << ") ";
    }
    std::cout << std::endl;
}

int main() {
    std::vector<int> vec(12);
    print_vector_info(vec);
    for (int i = 0; i < 5; i++) { vec.push_back(111 * (i + 1)); }
    print_vector_info(vec);
    vec.erase(vec.begin() + 4);
    print_vector_info(vec);
    system("pause");
    return 0;
}
```



Сравнение с TVector:
TVector каждый раз по необходимости выделяет новую память с запасом на фиксированную величину. Такой подход позволяет не выделить лишнюю память просто так.
/
std::vector изначально выделяет блок памяти фиксированного размера. Когда вектор становится полным, блок памяти увеличивается в два раза. Однако на практике это не всегда работает правильно.

TVector позволяет применять вставку и удаление с начала и середины вектора. Это значительно расширяет функционал несмотря на возросшее время O(n). Также всё ещё возможно проведение быстрой вставки и удаления из конца. Имеется возможность перевыделения лишней памяти при удалении элементов.
/
std::vector ограничен лишь быстрой вставкой и удалением последнего элемента.

TVector имеет имеет уникальные в данном сравнении функции поиска и сортировки

TVector по итогу имеет внушительный функционал и способен экономить память. Его использование выглядит предпочтительнее.