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

См. код исследования в приложении А

Представление в памяти.

У вектора есть метод .size(), это количество реальных объектов; и .capacity(), это количество объектов, под которые зарезервирована память. Существует функция .empty(), которая проверяет контейнер на наличие элементов. Функция .max_size() возвращает максимально возможное количество элементов. Функция .reserve() позволяет самостоятельно увеличить capacity до необходимых значений при необходимости. Функция .shrink_to_fit() удаляет неиспользуемую capacity в векторе.

По результатам исследования было выяснено, что при функциях вставки или удаления capacity меняется в соответствии с размером вектора (size), то есть в любой момент времени capacity равна size. Такое поведение не соответствует написанному в данной статье: https://www.geeksforgeeks.org/how-does-a-vector-work-in-c/

Экспериментальным путём было установлено, что с некоторым шансом (по приблизительной оценке, с шансом 1/4) после удаления последнего элемента адреса всех ячеек памяти меняется. Также при вставке значений иногда (приблизительно с шансом 1/10) наблюдается такое же поведение. Причина возникновения подобных отклонений мне не понятна.

В общем же случае, при удалении элемента из любого места в векторе удалённая ячейка соответственно заполняется значением ячейки со следующим индексом, проходя и переставляя, таким образом, все значения, которые идут после (включительно) удалённой ячейки.

Выяснена особенность: обращение к элементам вектора происходит не через десятичное число (индекс), например, как в массивах, а через конструкцию vector.begin() + <позиция>

Схема представления в памяти:

00805D98	00805D9C	00805DA0	00805DA0	00805DA8	00805DAC	00805DB0
0	1	2	3	4	5	6

Вставка.

За вставку отвечают методы .insert() и .push_back(). Они вставляют значение или значения, соответственно увеличивая size и capacity вектора (об их поведении сказано выше). Адреса всех ячеек после (включительно) той, на место которой что-то вставили меняются.

Удаление.

За удаление отвечают методы .erase() и .pop_back(). .erase() удаляет элемент из заданной позиции. .pop_back() удаляет последний элемент. Поведение ячеек при удалении опасно выше. Функция .clear(), очищает все ячейки вектора, очищает (не запоминает) их адреса и очищает capacity.

Сравнение с TVector

- Отсутствие состояния у ячеек
- Адресация происходит по vector.begin() + <позиция>, а не по индексам
- Capacity (доп. память) не выделяется

```
12 12
0(00BA5D98) 1(00BA5D9C) 2(00BA5DA0) 3(00BA5DA4) 4(00BA5DA8) 5(00BA5DAC) 6(00BA5DB0) 7(00BA5DB4) 8(00BA5DB8) 9(00BA5DBC) 10(00BA5DC0) 11(00BA5DC4)

17 17
0(00BAA048) 1(00BAA04C) 2(00BAA050) 3(00BAA054) 4(00BAA058) 5(00BAA05C) 6(00BAA060) 7(00BAA064) 8(00BAA068) 9(00BAA06C) 10(00BAA070) 11(00BAA074) 111(00BAA078) 222(00BAA07C) 333(00BAA080) 444(00BAA084) 555(00BAA088)

16 16
0(00BAA048) 1(00BAA04C) 2(00BAA050) 3(00BAA054) 5(00BAA058) 6(00BAA05C) 7(00BAA060) 8(00BAA064) 9(00BAA068) 10(00BAA06C) 11(00BAA070) 111(00BAA074) 222(00BAA078) 333(00BAA07C) 444(00BAA080) 555(00BAA084)

15 15
0(00BAA048) 1(00BAA04C) 2(00BAA050) 3(00BAA054) 5(00BAA058) 6(00BAA05C) 7(00BAA060) 8(00BAA064) 9(00BAA068) 10(00BAA06C) 11(00BAA070) 111(00BAA074) 222(00BAA078) 333(00BAA07C) 444(00BAA080)

16 16
0(00BAA048) 14(00BAA04C) 1(00BAA050) 2(00BAA054) 3(00BAA058) 5(00BAA05C) 6(00BAA060) 7(00BAA064) 8(00BAA068) 9(00BAA06C) 10(00BAA070) 11(00BAA074) 111(00BAA078) 222(00BAA07C) 333(00BAA080) 444(00BAA084)

18 18
0(00BAA048) 14(00BAA04C) 1(00BAA050) 2(00BAA054) -1(00BAA058) -4(00BAA05C) 3(00BAA060) 5(00BAA064) 6(00BAA068) 7(00BAA06C) 8(00BAA070) 9(00BAA074) 10(00BAA078) 11(00BAA07C) 111(00BAA080) 222(00BAA084) 333(00BAA088) 444(00BAA08C)

17 17
14(00BAA048) 1(00BAA04C) 2(00BAA050) -1(00BAA054) -4(00BAA058) 3(00BAA05C) 5(00BAA060) 6(00BAA064) 7(00BAA068) 8(00BAA06C) 9(00BAA070) 10(00BAA074) 11(00BAA078) 111(00BAA07C) 222(00BAA080) 333(00BAA084) 444(00BAA088)

Для продолжения нажмите любую клавишу . . .
```

Приложение А: проведение эксперимента

```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;
	std::cout << std::endl;
}

int main() {
	std::vector<int> vec(12);
	for (int i = 0; i < vec.size(); i++) vec[i] = i;
	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);
	vec.pop_back();
	print_vector_info(vec);
	vec.insert(vec.begin() + 1, 14);
	print_vector_info(vec);
	vec.insert(vec.begin() + 4, {-1, -4});
	print_vector_info(vec);
	vec.erase(vec.begin());
	print_vector_info(vec);
	return 0;
}
```