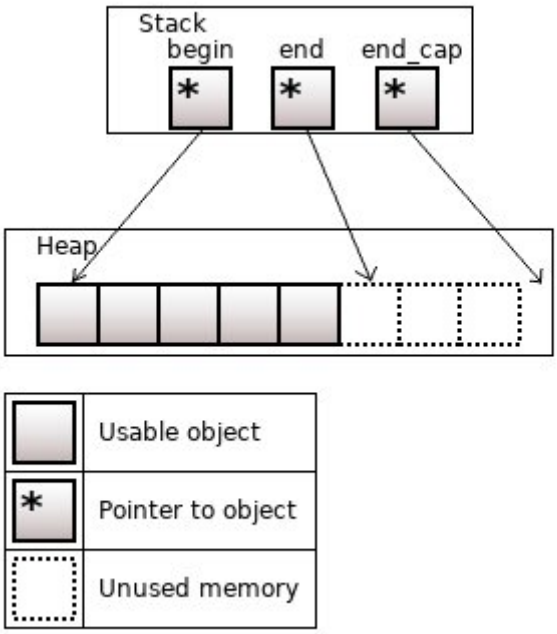
Описание принципов работы std::vector

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

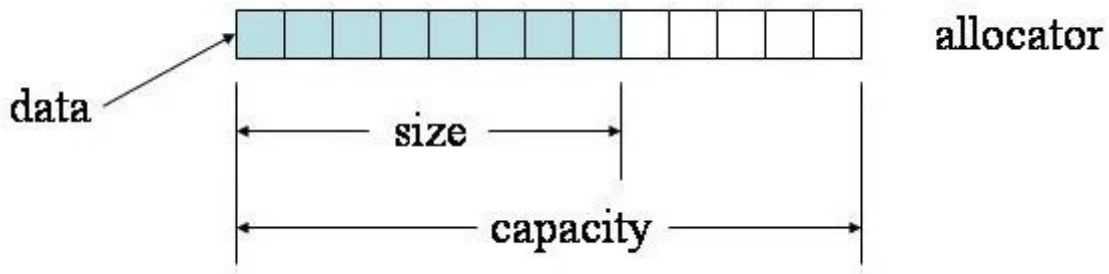
Представление в памяти:

Элементы вектора хранятся в непрерывном блоке памяти аналогично обычному массиву. Вектор представлен в памяти как три указателя: T* begin (указатель на начало выделенной памяти), T* end (указатель на элемент, следующий после последнего хранимого), T* capacity_end (указатель на конец выделенной памяти).



Методы:
1) size() = end* - begin*.
Представляет собой текущее хранимое количество элементов.
2) capacity() = capacity_end* - begin*.
Это максимальное количество элементов, которое вектор может хранить без перевыделения памяти. Значение capacity всегда >= size.
3) reserve(n)
Метод позволяет выделить память под n элементов

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



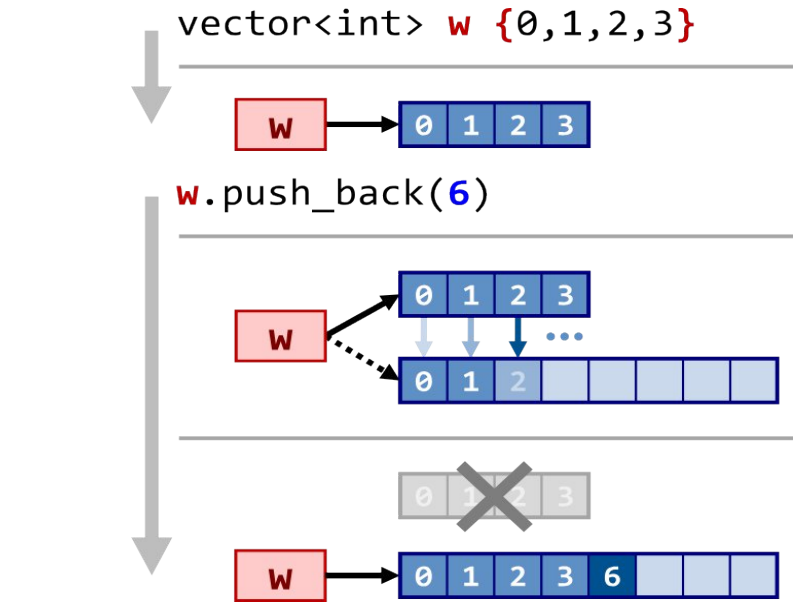
Вставка:

Методы:

1) push_back – добавляет элемент в конец.

Когда элемент вставляется, он копируется в конец массива в выделенную память. End увеличивается на 1. Сложность O(1).

Мы можем продолжать вставлять элементы таким образом до тех пор, пока размер не сравняется с емкостью (size == capacity), а это значит, что вектор заполнен. Чтобы вставить больше элементов, необходимо выполнить перераспределение. Выделяется новый блок памяти, существующие элементы копируются в новый блок, добавляется новый элемент, старая память освобождается. В C++ нет фиксированного коэффициента роста. Реализация зависит от компилятора. В Microsoft VS, в которой я работаю, он равен 1,5. (округление вниз)




```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 13. 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 750, но оно равно 505).

```
std::vector<int> vec(500);    505 505
```

Когда функция принимает аргумент по значению, создаётся копия исходного вектора. Когда вектор копируется, его копия выделяет новую память, достаточную для хранения текущих элементов исходного вектора. Поэтому capacity == size. Это стандартное поведение конструктора копирования в C++.

Когда функция принимает вектор по константной ссылке, копирования не происходит. Функция работает с оригинальным вектором, поэтому в таком случае выводится актуальный capacity.

```
void print_vector_info(const std::vector<int> &vec)
```

```
6 6
0(00000
11 13
```

```
std::vector<int> vec(500);    505 750
```

2) emplace_back
Конструирует элемент на месте, избегая лишний копирований. Функция аналогична push_back, но эффективнее для объектов с дорогим копированием.

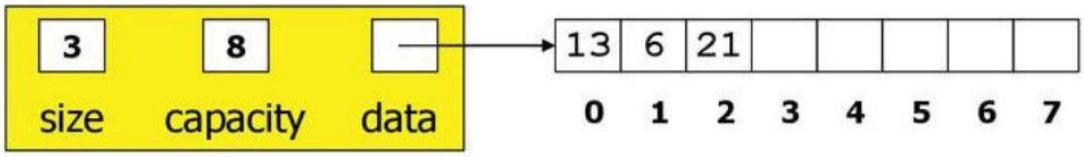
3) insert
Вставляет элемент перед определённой позицией. Элементы после этой позиции сдвигаются вправо, новый элемент копируется и перемещается на заданную позицию

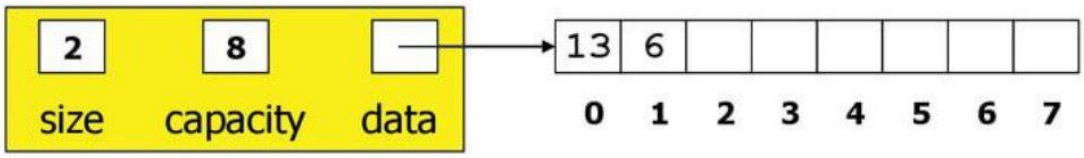
Удаление:

Методы:

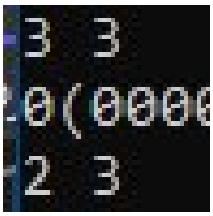
1) pop_back

Удаляет последний элемент, деструктор вызывается немедленно. Удаление элементов не приводит к перераспределению. Удаленный объект будет уничтожен, но память останется принадлежать вектору. Сложность O(1).






```
std::vector<int> vec(3);
print_vector_info(vec);
vec.pop_back();
```



2) erase

Удаляет элемент на определённой позиции, после которой все сдвигается влево.

Сложность O(n) из-за сдвига.

```
vec.erase(vec.begin()+4);
print_vector_info(vec);
```

```
6 6
1(000001B7C8FF04F0) 2(000001B7C8FF04F4) 3(000001B7C8FF04F8) 4(000001B7C8FF04FC) 5(000001B7C8FF0500) 6(000001B7C8FF0504)
5 6
1(000001B7C8FF04F0) 2(000001B7C8FF04F4) 3(000001B7C8FF04F8) 4(000001B7C8FF04FC) 6(000001B7C8FF0500)
```

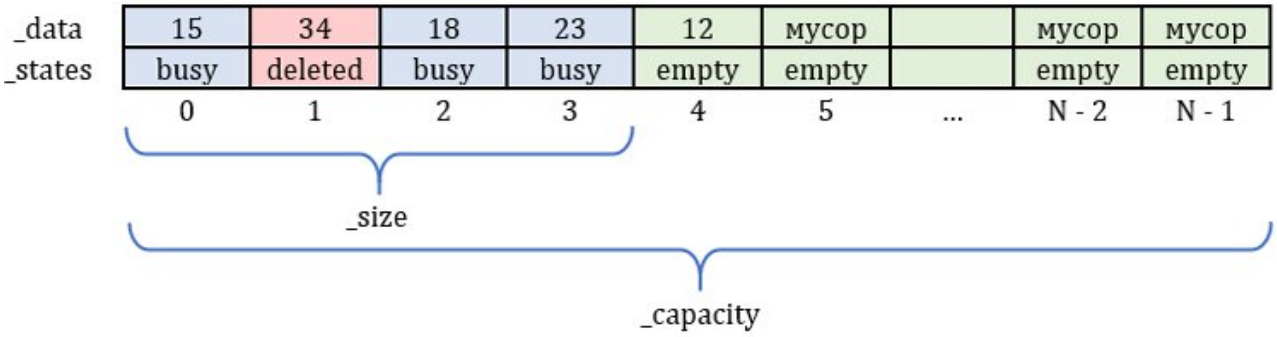
3) clear

Удаляет все элементы, не изменяет capacity. Сложность O(n).

Сравнение с TVector:

Представление в памяти:

TVector каждый раз по необходимости выделяет новую память с запасом на фиксированную величину. Такой подход позволяет не выделить лишнюю память просто так.



Вставка:

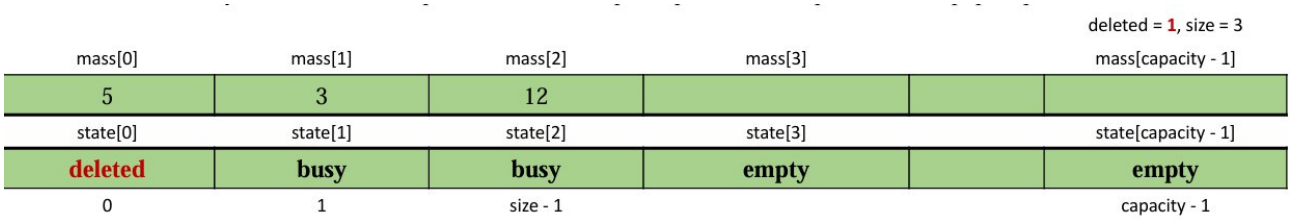
При создании массива выделяется память под хранение элементов с запасом, по мере переполнения памяти добавляется память также с запасом. Это позволяет сильно уменьшить количество перераспределений памяти при небольших size.

TVector позволяет применять вставку с начала и середины вектора. В таком случае время возрастает до O(n).

Временная сложность: O(1) / O(N).

Удаление:

Вводятся «статусы» для элементов (свободен, занят, удалён), при удалении элемента лишь помечать элемент удаленным без реального смещения; устанавливается пороговое значение процента удаленных элементов при котором следует автоматически перераспределить память



TVector имеет имеет функции поиска и сортировки, не требуя подключения сторонних библиотек.