

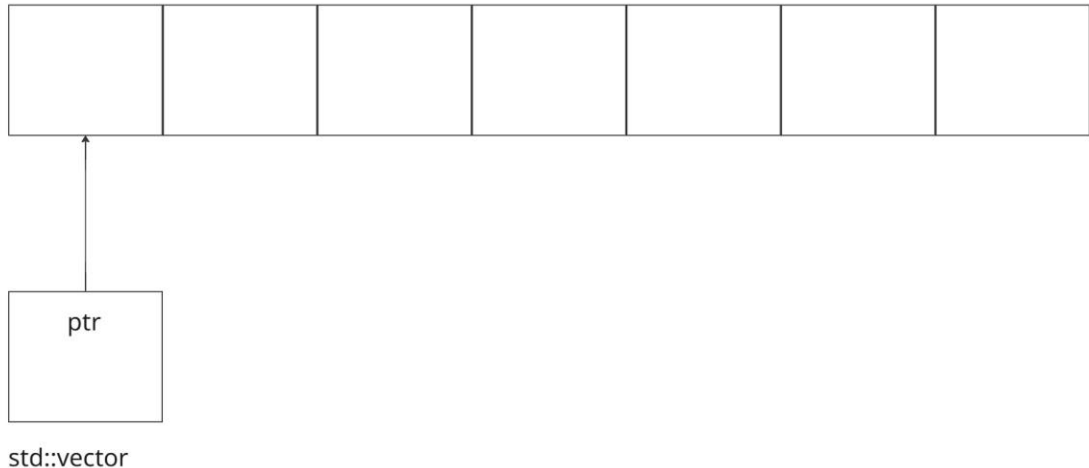
Принципы работы контейнера `std::vector` из библиотеки `<vector>`

Представление в памяти

При создании вектора выделяется блок памяти фиксированного размера, все элементы вектора хранятся в этом блоке непрерывно. Сам вектор содержит указатель на выделенный в куче буфер.

Схема представления вектора в памяти:

Выделенный в куче буфер:



Основные методы вектора для работы с памятью:

- `.size()` - это количество реальных объектов (т. е размер)
- `.capacity()` - это количество объектов, под которые зарезервирована память (т. е емкость).
- `.reserve(n)` - выделит память под `n` элементов (но `size` при этом будет равно 0).
- `.shrink_to_fit()` – уменьшает `capacity` до `size`.

Вставка

При вставке элемента он копируется в выделенную память, и `size` увеличивается. Мы можем продолжать вставлять элементы до тех пор, пока размер не станет равен ёмкости, что означает, что вектор заполнен. Поэтому пока размер меньше емкости временная сложность составляет $O(1)$ при вставке в конец(при других вставках должен учитываться сдвиг элементов), а когда он равен емкости, происходит перераспределение. При перераспределении весь блок копируется в новый блок большей емкости , что занимает $O(n)$.

Основные методы вставки в векторе:

- `.push_back()` - добавляет новый элемент в конец вектора.
- `.insert()` - вставляет новые элементы в заданную позицию в векторе.
- `.emplace()` - вставляет элементы в заданную позицию в векторе, создавая их на месте внутри вектора.

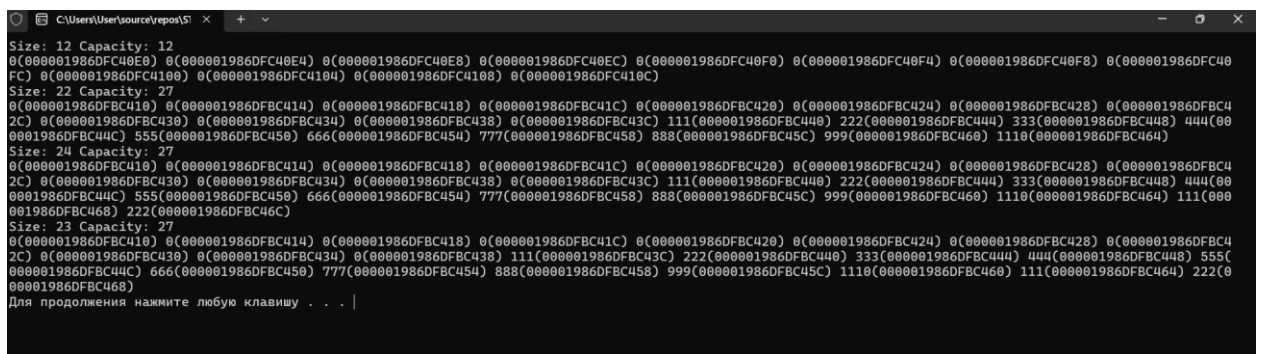
Удаление

Удаление элементов не приводит к перераспределению памяти. Удаляемый объект будет уничтожен, но память по-прежнему будет принадлежать вектору (так как `saracity` изменяться не будет, а `size` будет). Поэтому временная сложность удаления одного элемента с конца - это $O(1)$ (при других способах удаления учитывается сдвиг элементов).

Основные методы удаления в векторе:

- `.clear()` - удаляет все элементы из вектора, эффективно опустошая его.
- `.pop_back()` - удаляет последний элемент вектора.
- `.erase()` - удаляет элемент или диапазон элементов в векторе из заданных позиций.

Сравнение с TVector



```
C:\Users\User\source\repos\SI x + v
Size: 12 Capacity: 12
0(000001986DFC40E0) 0(000001986DFC40E4) 0(000001986DFC40E8) 0(000001986DFC40EC) 0(000001986DFC40F0) 0(000001986DFC40F4) 0(000001986DFC40F8) 0(000001986DFC40FC) 0(000001986DFC4100) 0(000001986DFC4104) 0(000001986DFC4108) 0(000001986DFC410C)
Size: 22 Capacity: 27
0(000001986DFC4110) 0(000001986DFC4114) 0(000001986DFC4118) 0(000001986DFC411C) 0(000001986DFC4120) 0(000001986DFC4124) 0(000001986DFC4128) 0(000001986DFC412C) 0(000001986DFC4130) 0(000001986DFC4134) 0(000001986DFC4138) 0(000001986DFC413C) 111(000001986DFC4140) 222(000001986DFC4144) 333(000001986DFC4148) 444(000001986DFC414C) 555(000001986DFC4150) 666(000001986DFC4154) 777(000001986DFC4158) 888(000001986DFC415C) 999(000001986DFC4160) 1110(000001986DFC4164)
Size: 24 Capacity: 27
0(000001986DFC4110) 0(000001986DFC4114) 0(000001986DFC4118) 0(000001986DFC411C) 0(000001986DFC4120) 0(000001986DFC4124) 0(000001986DFC4128) 0(000001986DFC412C) 0(000001986DFC4130) 0(000001986DFC4134) 0(000001986DFC4138) 0(000001986DFC413C) 111(000001986DFC4140) 222(000001986DFC4144) 333(000001986DFC4148) 444(000001986DFC414C) 555(000001986DFC4150) 666(000001986DFC4154) 777(000001986DFC4158) 888(000001986DFC415C) 999(000001986DFC4160) 1110(000001986DFC4164) 111(000001986DFC4168) 222(000001986DFC416C)
Size: 23 Capacity: 27
0(000001986DFC4110) 0(000001986DFC4114) 0(000001986DFC4118) 0(000001986DFC411C) 0(000001986DFC4120) 0(000001986DFC4124) 0(000001986DFC4128) 0(000001986DFC412C) 0(000001986DFC4130) 0(000001986DFC4134) 0(000001986DFC4138) 111(000001986DFC413C) 222(000001986DFC4140) 333(000001986DFC4144) 444(000001986DFC4148) 555(000001986DFC414C) 666(000001986DFC4150) 777(000001986DFC4154) 888(000001986DFC4158) 999(000001986DFC415C) 1110(000001986DFC4160) 111(000001986DFC4164) 222(000001986DFC4168)
Для продолжения нажмите любую клавишу . . . |
```

Рис.1 - Запуск Тестовой Программы

Приложение А: проведение эксперимента

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

void print_vector_info(std::vector<int> &vec) {
    std::cout << "Size: " << vec.size() << " " << "Capacity: " << 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 < 10; i++) { vec.push_back(111 * (i + 1)); }
    print_vector_info(vec);
    for (int i = 0; i < 2; i++) { vec.push_back(111 * (i + 1)); }
    print_vector_info(vec);
    vec.erase(vec.begin() + 4);
    print_vector_info(vec);
    system("pause");
    return 0;
}
```

Итог эксперимента

Запуск тестовой программы показал нам ожидаемые результаты. Важно заметить, что в начале(при инициализации) $size = capacity$, поэтому при любой дальнейшей вставке произойдет перераспределение, и адреса у элементов вектора изменятся. То что при любой вставке не происходит перераспределение доказывает вторая вставка(адреса у элементов не изменились из-за маленького количества вставляемых). Удаление также сработало правильно, удалив элемент и сдвинув все остальные, что можно посмотреть по адресам. Также интересно изменение $capacity$ с увеличением количества элементов: до 5 элементов $size = capacity$, далее оно всегда превосходит $size$ в 1,5 – 2 раза.