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

Представление в памяти. У вектора есть метод size(), который возвращает количество элементов в векторе; и метод capacity(), который возвращает количество элементов, под которые нужно зарезервировать память. Резервирование памяти осуществляется с помощью метода reserve(), в который передают новую вместимость. Изменение размера происходит с помощью метода resize(), в который передают либо только количество элементов, которое должен содержать вектор, либо передают ещё значение, которым будут заполняться новые ячейки, если переданное количество элементов будет больше текущего размера вектора. Если же текущий размер больше переданного количества элементов, то контейнер уменьшается до этого количества элементов. Метод shrink_to_fit() уменьшает вместимость до размера контейнера. Этот метод ничего не принимает и не возвращает.

F001	F002	F003	F004	F005	F006	F007
1	2	3	4	5782	4848	1654

vector.size() = 4; vector.capacity() = 7;

Рис. 1. Схема представления вектора в памяти.

```
8 8
1(00000183507B1FB0)2(00000183507B1FB4)3(00000183507B1FB8)4(00000183507B1FBC)5(00000183507B1FC0)6(00000183507B1FC4)7(0000
0183507B1FC8)8(00000183507B1FCC)
После использования vec.reserve(15)
8 15
1(00000183507CD910)2(00000183507CD914)3(00000183507CD918)4(00000183507CD91C)5(00000183507CD920)6(00000183507CD924)7(0000
0183507CD928)8(00000183507CD92C)
После использования vec.resize(8)
8 15
1(00000183507CD910)2(00000183507CD914)3(00000183507CD918)4(00000183507CD91C)5(00000183507CD920)6(00000183507CD924)7(0000
0183507CD928)8(00000183507CD92C)
После использования vec.resize(10, 22)
10 15
1(00000183507CD910)2(00000183507CD914)3(00000183507CD918)4(00000183507CD91C)5(00000183507CD920)6(00000183507CD924)7(0000
0183507CD928)8(00000183507CD92C)22(00000183507CD930)22(00000183507CD934)
После использования vec.shrink_to_fit()
10 10
1(00000183507CF470)2(00000183507CF474)3(00000183507CF478)4(00000183507CF47C)5(00000183507CF480)6(00000183507CF484)7(0000
0183507CF488)8(00000183507CF48C)22(00000183507CF490)22(00000183507CF494)
Для продолжения нажмите любую клавишу . . .
```

Рис. 2. Результат запуска тестовой программы с методами, которые работают с памятью.

На рис. 2. видно, что сначала у нас размер равен вместимости, после вызова метода reserve(15) у нас изменился размер вместимости и произошло перевыделение памяти (это можно заменить благодаря изменившимся адресам). Затем после вызова метода resize(8) у нас изменился размер вектора, но вместимость не поменялась и перевыделения памяти не было. После вызова метода resize(10, 22) у нас увеличился размер на 2 ячейки и новые два элемента приняли значение 22, вместимость не изменилась. Затем происходил вызов функции shrink_to_fit() и после этого у нас вместимость стала равна размеру и так как вместимость изменилась, произошло перевыделение памяти. После вызова метода clear() у нас обнулился размер.

Вставка. Вставка в середину/начало (insert()) сдвигает все элементы вправо и вставляет элемент/элементы. Например, если в начало вставить элемент со значением 0, то в F001 (на рис. 1.) поместится значение 0, а в F002 перейдёт значение 1 и последующие элементы также сдвинутся на одну ячейку.

Рассмотрим три объявления метода insert. Этот метод во всех объявлениях возвращает указатель на вставленный элемент. В первом варианте принимает позицию, куда вставлять и константную ссылку на значение. Во втором случае принимает позицию, количество вставляемых элементов, константную ссылку на значение. В третьем варианте принимает позицию и список вставляемых значений. Позицию передают в формате begin() + pos/end() - pos, где pos - индекс для вставки относительно начала/конца вектора. Метод возвращает итератор, указывающий на вставленный элемент. Если значения вместимости не хватает для вставки элементов, то происходит автоматическое перевыделение памяти.

Вставка в конец (push_back()) добавляет элемент в «хвост» вектора (на рис.1. «хвост» вектора – ячейки F005-F007). Например, если вставить элемент со значением 5, то в ячейку F005 (на рис. 1.) поместится значение 5.

Этот метод принимает константную ссылку на значение элемента для вставки и ничего не возвращает. Если значения вместимости не хватает для вставки элементов, то происходит автоматическое перевыделение памяти.

При вставке изменяется размер.

```
8 8
1(000001CB9F952250)2(000001CB9F952254)3(000001CB9F952258)4(000001CB9F95225C)5(000001CB9F952260)6(000001CB9F952264)7(000001CB9F952268)8(000001CB9F95226C)
После использования vec.insert(vec.begin() + 1, 55)
9 12
1(000001CB9F96F550)55(000001CB9F96F554)2(000001CB9F96F558)3(000001CB9F96F55C)4(000001CB9F96F560)5(000001CB9F96F564)6(000001CB9F96F568)7(000001CB9F96F56C)8(000001CB9F96F570)
После использования vec.insert(vec.begin(), { 66, 77, 88, 99 })
13 18
66(000001CB9F972E20)77(000001CB9F972E24)88(000001CB9F972E28)99(000001CB9F972E2C)1(000001CB9F972E30)55(000001CB9F972E34)2(000001CB9F972E38)3(000001CB9F972E3C)4(000001CB9F972E40)5(000001CB9F972E44)6(000001CB9F972E48)7(000001CB9F972E4C)8(000001CB9F972E50)
После использования vec.push_back(111 * (i + 1)) 3 раза
16 18
66(000001CB9F972E20)77(000001CB9F972E24)88(000001CB9F972E28)99(000001CB9F972E2C)1(000001CB9F972E30)55(000001CB9F972E34)2(000001CB9F972E38)3(000001CB9F972E3C)4(000001CB9F972E40)5(000001CB9F972E44)6(000001CB9F972E48)7(000001CB9F972E4C)8(000001CB9F972E50)111(000001CB9F972E54)222(000001CB9F972E58)333(000001CB9F972E5C)
Для продолжения нажмите любую клавишу . . .
```

Рис. 3. Результат запуска тестовой программы с методами, которые работают для вставки.

На рис. 3. видно, что сначала у нас создался вектор с размеров и вместимостью равными 8, затем после вставки элемента (insert(vec.begin()+1, 55) у нас произошло перевыделение памяти (заметно благодаря изменившимся адресам), после вызова метода для вставки значений из массива (insert(vec.begin(), {66, 77, 88, 99}) у нас новый размер опять превысил вместимость, поэтому произошло перевыделение памяти. Затем после вставки в конец трёх элементов перевыделения памяти не произошло, так как размер не превысил вместимость.

Удаление. Удаление в середине/начале (erase()) происходит сдвигом элементов, так как все элементы после удалённого, сдвигаются на один по адресу. Например, если удалить элемент со значением 3, то тогда в ячейку F003 (на рис. 1.) перейдет значение 4 и т.д.

Этот метод принимает либо указатель на позицию или константный указатель на позицию, где удалить, либо же два указателя на позиции или два константных указателя, т. е. диапазон, в котором нужно удалить элементы. Позицию передают в формате begin() + pos/end() - pos, где pos - индекс для удаления относительно начала/конца вектора. Возвращает итератор, указывающий на следующий элемент после удалённого. Если удалён последний элемент итератор будет указывать на конец вектора.

Удаление в конце (pop_back()) удаляет последний элемент и эта ячейка переходит в «хвост» вектора (на рис.1. «хвост» вектора – ячейки F005-F007). Например, если удалить элемент со значением 4, то в ячейка F004 (на рис. 1.) перейдёт в «хвост» вектора.

Этот метод ничего не принимает и ничего не возвращает.

Метод clear() удаляет все элементы вектора. Размер обнуляется, перевыделение памяти не происходит. Этот метод ничего не принимает и не возвращает.

При удалении изменяется размер.

Рис. 4. Результат запуска тестовой программы с методами, которые работают с удалением.

На рис. 4. видно, что перевыделения памяти не происходило, размер уменьшается на количество удалённых элементов. Всегда при удалении из начала/середины элементы сдвигаются влево, при удалении из конца, ячейка переходит в «хвост» вектора.
Метод .clear() удаляет все элементы контейнера, при этом не перевыделяя память, это можно заметить по адресу первого элемента после вставки.

Другие методы. Метод empty() возвращает true, если в контейнере нет элементов, т.е. размер равен нулю; иначе возвращает false.

Метод assign() имеет две реализации. В первой принимает количество элементов и значение для присваивания. Во второй принимает список значений для присваивания. В обоих случаев ничего не возвращает.

Метод at() имеет тоже две реализации. В обоих случаях принимает позицию. Отличаются случаи тем, что один возвращает ссылку, а другой константную ссылку.

Сравнение с TVector.

Представление в памяти. В TVector представление в памяти будет иметь похожий вид. Будут представлены методы size(), capacity(), reserve(), resize(), shrink_to_fit(). Отличие будет состоять в том, что в TVector вместимость будет рассчитываться по специальной формуле.

Удаление. В TVector будут также представлены методы pop_back(), erase() и появится метод pop_front(). Также в TVector у ячеек будут статусы их состояния и если значение в ячейке будет удалено, то смещения элементов происходить не будет, только лишь измениться статус ячейки с busy на delete. В отличие от std::vector, где при удаление все элементы сдвигались влево. Также будет происходить перевыделение памяти, если процент удаленных достигнет 10%.

Вставка. В TVector также будут представлены методы push_back(), insert() и появится метод push_front. Сама вставка будет осуществляться с учётом статусов ячеек.

Другие методы. Методы assign(), at() будут реализованы также, как и в std::vector.

Приложение А: проведение эксперимента с методами, которые работают с памятью.

```
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() {
    setlocale(LC_ALL, "rus");
    std::vector<int> vec({ 1, 2, 3, 4, 5, 6, 7, 8 });
    print_vector_info(vec);
    vec.reserve(15);
    std::cout << "После использования vec.reserve(15)\n";
    print_vector_info(vec);
    vec.resize(8);
    std::cout << "После использования vec.resize(8)\n";
    print_vector_info(vec);
    vec.resize(10, 22);
    std::cout << "После использования vec.resize(10, 22)\n";
    print_vector_info(vec);
    vec.shrink_to_fit();
    std::cout << "После использования vec.shrink_to_fit()\n";
    print_vector_info(vec);
    system("pause");
    return 0;
}
```

Приложение В: проведение эксперимента с методами, которые работают со вставкой.

```
int main() {
    setlocale(LC_ALL, "rus");
    std::vector<int> vec({ 1, 2, 3, 4, 5, 6, 7, 8 });
    print_vector_info(vec);
    vec.insert(vec.begin() + 1, 55);
    std::cout << "После использования vec.insert(vec.begin() + 1, 55)\n";
    print_vector_info(vec);
    vec.insert(vec.begin(), { 66, 77, 88, 99 });
    std::cout << "После использования vec.insert(vec.begin(), { 66, 77, 88, 99 })\n";
    print_vector_info(vec);
    for (int i = 0; i < 3; i++) {
        vec.push_back(111 * (i + 1));
    }
    std::cout << "После использования vec.push_back(111 * (i + 1)) 3 раза\n";
    print_vector_info(vec);
    system("pause");
    return 0;
}
```

Приложение С: проведение эксперимента с методами, которые работают с удалением.

```
int main() {
    setlocale(LC_ALL, "rus");
    std::vector<int> vec({ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 27, 18, 19, 20,
        21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 1, 2, 3});
    print_vector_info(vec);
    vec.erase(vec.begin());
    std::cout << "После использования vec.erase(vec.begin())\n";
    print_vector_info(vec);
    vec.erase(vec.begin() + 1, vec.end() - 9);
    std::cout << "После использования vec.erase(vec.begin() + 1, vec.end() - 9)\n";
    print_vector_info(vec);
    for (int i = 0; i < 2; i++) {
        vec.pop_back();
    }
    std::cout << "После использования vec.pop_back() 2 раза\n";
    print_vector_info(vec);
    vec.clear();
    std::cout << "После использования vec.clear()\n";
    print_vector_info(vec);
    system("pause");
    return 0;
}
```