**Мой класс TVector имеет функциональные возможности:**

1. Возможность создания:

a) пустого вектора по умолчанию,

b) вектора заданного размера,

c) вектора, созданного на основе переданного массива данных,

d) конструктор копирования + иные необходимые на ваш взгляд способы создания вектора; а также уничтожения.

2. Функции: data(), size(), capacity(), front(), back() + функции begin(), end().

3. Функция проверки на пустоту is\_empty().

4. Стандартные функции вставки: push\_front(), push\_back(), insert().

5. Стандартные функции удаления: pop\_front(), pop\_back(), erase().

6. Функция замены значения emplace().

7. Функция присваивания assign(), функция обращения по индексу at() с проверкой допустимости индекса, функция очистки вектора clear().

8. Функция shrink\_to\_fit() для возврата дополнительной памяти в систему. Функция резервирования памяти заданного размера reserve(); если меньше текущей памяти – ничего не происходит. Функция изменения размера массива resize() с заполнителем и без (работает как на расширение памяти, так и на сжатие).

9. Должны быть перегружены операторы: присваивания =, сравнения !=, ==, оператор [].

Служебный функционал представляет собой:

1. Функции работы с памятью в массиве;

2. Функция проверки на заполненность is\_full().

Вне класса (дружественные функции):

1. Функцию перемешивания через, например, алгоритм Фишера-Йейтса.

2. Эффективный способ сортировки массива.

3. Поиск элементов с заданными свойствами: первого, последнего, всех.

**Разработка эффективных методов класса TVector:**

|  |  |  |
| --- | --- | --- |
| СОЗДАНИЕ | Создавать запас памяти для обработки будущего роста (например, 15 элементов). | без переноса данных – O(1) с переносом – O(n) |
| ВСТАВКА | При вставке элемент в начало – произвести сдвиг вправо всех данных, вставить элемент на освободившуюся позицию. При вставке элемент в середину – произвести сдвиг вправо до заданной позиции, вставить элемент на освободившуюся позицию. При вставки в конец, добавить элемент, изменить счётчик; при нехватки памяти для вставки – расширить память также с запасом для роста. | в начало – O(n) в середину – O(n) в конец – O(1) с перевыделением памяти – O(n) |
| УДАЛЕНИЕ | При удалении элемента из конца – пометить его empty. При удалении элемента с другой позиции – пометить его deleted. При наличии в массиве большого числа удалённых элементов (например, процент и ли прогрессивная шкала процентов) – произвести перевыделение памяти меньшего размера с исключением удаленных элементов с выделением запаса для роста. | с перевыделением памяти – O(n) иначе – O(1) |
| ПОИСК | Поиск осуществлять только по элементам в состоянии busy. Требуется перерасчет найденных индексов для пользователя (он не видит удаленные элементы). | O(n) |
| ПЕРЕМЕШИВАНИЕ | Алгоритм Фишера-Йейтса (Кнута) | O(n) |
| СОРТИРОВКА | Алгоритм быстрой сортировки (сортировка Хоара) | O(n log(n)) |
| ВЫВОД НА ЭКРАН | При стандартном выводе не показывать элементы со статусом deleted | O(n) |

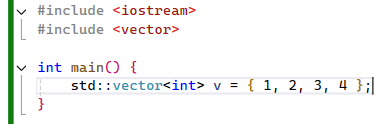
**Принципы работы контейнера std::vector из библиотеки <vector>**

**Представление в памяти:** std::vector использует **динамический массив**, но с оптимизациями. **size()** – количество реально существующих элементов в векторе. **capacity()** – количество элементов, под которые уже выделена память (но они еще не инициализированы).

Вектор состоит из трех частей:

1. Указатель на выделенную память (динамический массив).
2. Переменная size, указывающая на **количество активных элементов**.
3. Переменная capacity, указывающая на **размер выделенной памяти**.

**Пример** представления std::vector<int>:



В памяти:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| индекс | 0 | 1 | 2 | 3 | - | - | - | - |
| значение | 1 | 2 | 3 | 4 | мусор | мусор | мусор | мусор |
| состояние | занят | занят | занят | занят | свободно | свободно | свободно | свободно |

#include <iostream>

#include <vector>

void print\_vector\_info(const std::vector<int>& vec, const std::string& msg = "") {

std::cout << msg << "\n";

std::cout << "Size: " << vec.size() << ", Capacity: " << vec.capacity() << "\n";

std::cout << "Elements: ";

for (size\_t i = 0; i < vec.size(); i++) {

std::cout << vec[i] << "(" << &vec[i] << ") ";

}

std::cout << "\n\n";

}

int main() {

// Создание вектора, работа с памятью

std::vector<int> vec;

print\_vector\_info(vec, "Empty vector");

vec.reserve(15); // Резерв память

print\_vector\_info(vec, "After reserve(15)");

// Добавляю элемент push\_back()

for (int i = 0; i < 10; i++) vec.push\_back(i \* 10);

print\_vector\_info(vec, "After push\_back 10 elements");

// Удаляю элемента erase()

vec.erase(vec.begin() + 3);

print\_vector\_info(vec, "After erase(3rd element)");

// Вставка в начало insert()

vec.insert(vec.begin(), 999);

print\_vector\_info(vec, "After insert(999 at begin)");

// front(), back(), data()

std::cout << "Front: " << vec.front() << ", Back: " << vec.back() << "\n";

std::cout << "Data pointer: " << vec.data() << "\n\n";

// Очистка вектора clear()

vec.clear();

print\_vector\_info(vec, "After clear()");

return 0;

}

