**Конспект: Квиз по векторам**

**Определение.** Вектор - это коллекция элементов, сохраненных в массиве, изменяющегося по мере необходимости размера (обычно, увеличивающегося). Грубо говоря, вектор - это улучшенный динамический массив.

Для работы с вектором необходимо подключить библиотеку <vector>:

#include <vector>

При объявлении вектора нужно указывать тип данных элементов, которые будут содержаться в этом векторе:

std::vector<int> vec; // создание пустого вектора

std::vector<char> vec2{‘s’}; // создание вектора, содержащего элемент ‘s’

std::vector<double> vec3(2); // создание пустого вектора из 2-х элементов

std::vector<float> vec4(3) = {1.2, 4.1, 5}; // создание вектора из 3-х элементов, равных 1.2, 4.1 и 5

**Важно.** Один вектор не может содержать элементы разных типов данных.

**Методы.**

* begin() - возвращает итератор на первый элемент вектора (элемент с индексом 0)
* end() - возвращает итератор на последний элемент вектора
* push\_back(n) - добавляет элемент n в конец вектора:

vec.push\_back(10);

* insert(m, n) - вставляет элемент n на позицию, на которую указывает итератор m:

vec.insert(vec.begin(), 0);

* size() - возвращает размер вектора (количество элементов в векторе):

std::cout << vec.size();

* back() - возвращает последний элемент вектора:

std::cout << vec.back();

* front() - возвращает первый элемент вектора:

std::cout << vec.front();

* pop\_back() - удаляет последний элемент вектора:

vec.pop\_back();

* erase(m) - удаляет элемент, на который указывает итератор p:

vec.erase(vec.begin());

* clear() - удаляет все элементы из вектора:

vec.clear();

* empty() - возвращает 1, если вектор пуст, 0 - если в векторе содержатся элементы:

std::cout << vec.empty();

**Доступ к элементам** также можно получить по индексу с помощью квадратных скобок и метода at():

int i1 = vec[0];

int i2 = vec.at(0);

Кроме того, к элементам можно обращаться с помощью итераторов:

cout << \*vec.begin() << endl;

**Для вывода вектора в консоль** нужно воспользоваться циклом.

Вариант цикла, где перебираются элементы вектора:

for (auto elem : v1) { //range based for

cout << elem << endl;

}

Как этот цикл работает:

for (auto p = v1.begin(); p != v1.end(); p++) { //как раскрывается range based for

auto elem = \*p;

cout << elem << endl;

}

Вариант цикла с перебором индексов:

for (int i=0; i < v1.size(); i++) { //как мы привыкли делать

cout << v1[i] << endl;

}

Используйте вектор когда не знаете сколько элементов в нём будет храниться. С этой задачей он справляется идеально и не нужно придумывать “велосипед” с постоянным выделением памяти под новые динамические массивы, ведь вектор в своей реализации уже все сделал за вас.

Если элементов действительно очень много, то вам как программисту достаточно указать приблизительный начальный размер вектора, чтобы не так сильно терять в скорости по сравнению с массивами.

Вектор в C++ может хранить те же данные, что и обычный массив. Вектор представляет собой динамический массив и обеспечивает удобные методы для работы с данными, такие как добавление, удаление и доступ к элементам. Он предоставляет аналогичный интерфейс для работы с данными, как и обычный массив, но с возможностью динамического изменения размера. Поэтому вектор может хранить элементы любых типов данных, таких как целые числа, числа с плавающей запятой, символы, строки, пользовательские классы и так далее.