Массив переменной длины в языке C++ реализован с помощью контейнера std::vector. Он представляет собой динамический массив, который может менять свою длину во время выполнения программы.

Изменение длины массива std::vector осуществляется с помощью различных методов и функций. Наиболее распространенными из них являются resize, reserve и push\_back.

Метод resize изменяет длину вектора, при этом, если новая длина больше предыдущей, добавляются новые элементы, а если меньше, лишние элементы удаляются.

std::vector<int> vec;

vec.resize(5); // установит длину вектора равной 5

Метод reserve выделяет память для заданного числа элементов, но фактически не меняет длину вектора.

std::vector<int> vec;

vec.reserve(10); // выделит память для 10 элементов, но длина вектора остается неизменной

Метод push\_back добавляет новый элемент в конец вектора и увеличивает его длину на 1.

std::vector<int> vec;

vec.push\_back(42); // добавит новый элемент со значением 42

В векторе std::vector есть две важные переменные: size и capacity.

Size представляет текущее количество элементов в векторе, тогда как capacity отражает количество элементов, которое в данный момент может вместить вектор без выделения новой памяти. Когда количество элементов в векторе превышает его capacity, происходит выделение нового блока памяти с большей capacity. Благодаря этому std::vector может увеличивать свою длину. Обычно capacity увеличивается в два раза относительно предыдущего значения.

Функция reserve позволяет задавать capacity вручную. Если вектор уже содержит элементы, то reserve увеличивает capacity только если он больше текущего значения, чтобы избежать лишнего выделения памяти.

Функция shrink\_to\_fit запрашивает освобождение ненужной памяти, пытаясь уменьшить capacity до size. Однако, платформа может не поддерживать эту операцию, и в этом случае shrink\_to\_fit ничего не делает.

Когда при создании std::vector используются знаки < и >, это указывает на тип элементов, которыми будет заполнен вектор. Например:

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

std::vector<std::string> vec2; // создание вектора типа std::string

Функция erase позволяет удалить один или несколько элементов из вектора по указанной позиции или диапазону. Например:

std::vector<int> vec = {1, 2, 3, 4, 5};

vec.erase(vec.begin() + 2); // удалит элемент с индексом 2 (третий элемент)

Функция insert позволяет вставить элементы в вектор по указанной позиции или диапазону. Например:

std::vector<int> vec = {1, 2, 3};

vec.insert(vec.begin() + 1, 42); // вставит число 42 перед вторым элементом

Функция clear очищает вектор, удаляя все его элементы, но сохраняет capacity:

std::vector<int> vec = {1, 2, 3};

vec.clear(); // удалит все элементы из вектора

Вектор std::vector предоставляет два способа доступа к его элементам: через оператор [] и через функцию at().

Оператор [] позволяет получить доступ к элементу по индексу, но не выполняет проверку выхода за границы вектора. Это может привести к ошибкам, если доступиться к несуществующему элементу.

std::vector<int> vec = {1, 2, 3};

int x = vec[1]; // получит второй элемент - значение 2

Функция at() также позволяет получить доступ к элементу по индексу, но выполняет проверку границ вектора. Если индекс выходит за пределы вектора, выбрасывается исключение std::out\_of\_range.

std::vector<int> vec = {1, 2, 3};

int x = vec.at(1); // получит второй элемент - значение 2

Различие между [] и at() заключается в том, что оператор [] быстрее, но не проверяет границы, в то время как функция at() является безопасной и выполняет проверку границ каждый раз при доступе к элементу. В C++ сделали именно так, чтобы разработчик сам решал, какой вариант использовать в зависимости от требований к быстродействию и безопасности.

Для демонстрации работы с std::vector можно использовать следующий код:

#include <iostream>

#include <vector>

int main() {

std::vector<int> vec = {1, 2, 3, 4, 5};

// изменение длины вектора

vec.resize(7); // увеличение длины до 7

vec.resize(3); // уменьшение длины до 3

// вывод элементов вектора

for (int i : vec) {

std::cout << i << " ";

}

std::cout << std::endl;

// показатель размера и емкости

std::cout << "Size: " << vec.size() << std::endl; // выводит 3

std::cout << "Capacity: " << vec.capacity() << std::endl; // выводит 5 (или больше в зависимости от реализации)

// добавление элементов в конец вектора

vec.push\_back(6);

vec.push\_back(7);

// удаление элемента по индексу

vec.erase(vec.begin() + 1);

// вставка элемента по индексу

vec.insert(vec.begin() + 2, 42);

// очистка вектора

vec.clear();

return 0;

}