std::vector в C++

Denis Bakin

Мотивация

- Иногда количество данных заранее неизвестно
- Нужно где-то хранить последовательность элементов
- Пример: сохранить цифры числа, чтобы потом вывести в правильном порядке
- Для этого удобно использовать контейнер

std::vector

Определение

- Динамический массив
- Поддерживает изменение размера
- Быстро добавляет элементы в конец

```
#include <vector>

std::vector<int> a;              // пустой вектор
std::vector<int> b(10);          // 10 элементов (не инициализированы)
std::vector<int> c(10, -1);      // 10 элементов, все -1
std::vector<int> d = {1, 2, 3};  // список инициализации
```

Итерация по вектору

```cpp
#include <iostream>
#include <vector>
#include <string>

int main() {
    std::vector<std::string> words = {"Just", "some", "random", "words"};

    for (std::string &w: words) {
        std::cout << w << ' ';
    }
    std::cout << '\n';
}
```

- Range-based for упрощает работу
- Тип элемента указывается внутри <>

Итерация по индексам

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

int main() {
    std::vector<int> data = {1, 2, 3, 4};

    for (size_t i = 0; i < data.size(); ++i) {
        std::cout << data[i] << ' ';
    }
    std::cout << '\n';
}
```

- Метод `.size()` возвращает количество элементов
- Тип `size_t` — беззнаковое целое для индексов

Основные методы

```
data.size();       // количество элементов
data.empty();      // true, если пуст
data.front();      // первый элемент
data.back();       // последний элемент

data.push_back(x); // добавить в конец
data.erase(it);    // удалить по итератору
```

Пример: цифры числа

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

int main() {
    int num, d;
    std::cin >> num >> d;

    std::vector<int> digits;
    while (num) {
        digits.push_back(num % d);
        num /= d;
    }

    for (size_t i = digits.size(); i > 0; --i) {
        std::cout << digits[i - 1];
    }
    std::cout << '\n';
}
```

- Запоминаем цифры с помощью `push_back`
- Выводим в обратном порядке

Пример задачи

Найти количество чётных элементов и максимальный из них

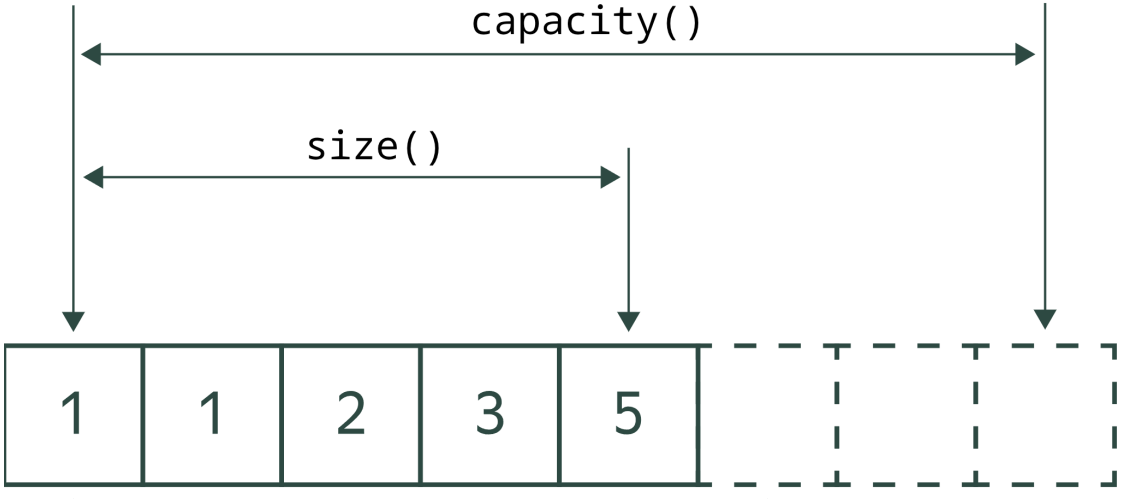
```cpp
#include <iostream>
#include <vector>
int main() {
    int n;
    std::cin >> n;
    std::vector<int> data(n);

    for (size_t i = 0; i < n; ++i) {
        std::cin >> data[i];
    }

    int cnt = 0, maxm = 0;
    for (int x: data) {
        if (x % 2 == 0) {
            ++cnt;
            if (cnt == 1 || x > maxm) maxm = x;
        }
    }
    std::cout << cnt;
    if (cnt) std::cout << " " << maxm;
```

Size и Capacity

- size — количество элементов
- capacity — сколько памяти выделено
- При переполнении capacity память удваивается



Size и Capacity

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

int main() {
    std::vector<int> v = {1, 2};
    std::cout << v.size() << "\t" << v.capacity() << "\n";

    v.push_back(3);
    std::cout << v.size() << "\t" << v.capacity() << "\n";
}
```

Оптимизация с reserve

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

int main() {
    int n;
    std::cin >> n;

    std::vector<int> v;
    v.reserve(n); // выделяем память заранее

    for (int i = 0; i < n; ++i) {
        int x;
        std::cin >> x;
        v.push_back(x);
    }
}
```

- `reserve` экономит время
- Сохраняет амортизированную сложность $O(1)$