V значит вектор — конспект темы

Создание и применение вектора

Контейнер — объект, который хранит массив значений. Вектор — стандартный контейнер C++.

Чтобы работать с вектором, нужно подключить библиотеку `<vector>`. При объявлении переменной в угловых скобках указывают тип элементов, которые будут в контейнере. Содержимое вектора пишется в фигурных скобках:

```
#include <vector>

using namespace std;

int main() {
    vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
}
```

У вектора есть несколько полезных методов.

- `size` позволяет выяснить размер вектора:

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

using namespace std;

int main() {
    vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    cout << month_lengths.size() << endl;
}
```

Элементы вектора нумеруются с нуля. Чтобы обратиться к конкретному элементу, нужно написать индекс в квадратных скобках:

```
int month_index;
cin >> month_index;
```

```
// Обращаемся к элементу вектора month_lengths с индексом month_index
cout << "There are "s << month_lengths[month_index] << " days"s << endl;
```

- `back` позволяет вычислить индекс последнего элемента, если вектор не пустой:

```
cout << month_lengths[month_lengths.size() - 1] << endl;  // количество дней в декабре
cout << month_lengths.back() << endl;                     // то же, но короче
```

- `empty` позволяет проверить вектор на пустоту:

```
vector<string> lost_pets = {"Oscar"s, "Pluto"s, "Tom"s};

if (lost_pets.empty()) {
    cout << "Wonderful! No pets are lost!"s << endl;
} else {
    cout << "Recent lost pet's name - "s << lost_pets.back() << endl;
}

// Программа выведет такой текст:
// Recent lost pet's name - Tom
```

Другой способ проверить вектор на пустоту — вызвать `size` и сравнить результат с 0.

Обращаться по индексу в векторе нужно осторожно. Корректный индекс должен быть неотрицателен и строго меньше размера вектора.

Лаконичный for и добавление элементов в вектор

Чтобы перебрать элементы вектора, используют цикл `for` и обращение по индексу:

```
for (int month_index = 0; month_index < month_lengths.size(); ++month_index) {
    cout << "There are "s << month_lengths[month_index] << " days "s
         << " in month "s << (month_index + 1) << endl;
}
```

Вывод:

```
There 31 days in month 1
...
There 31 days in month 12
```

`range-based for` — специальный цикл, который применяют, чтобы выполнить действия для каждого элемента контейнера.

```
cout << "Month lengths are:"s;

// Значениями переменной length поочерёдно станут все элементы вектора month_length.
for (int length : month_lengths) {
    cout << " "s << length;
}
cout << endl;
```

Вывод:

```
Month lengths are: 31 28 31 30 31 30 31 31 30 31 30 31
```

Строка — это тоже контейнер, содержащий символы, к ней примени́м `range-based for`:

```
string str = "Tiger, tiger, burning bright, in the forests of the night"s;

// посчитаем количество пробелов в строке
int spaces = 0;
for (char c : str) {
    // если очередной символ - пробел, добавим единицу к переменной
    if (c == ' ') {
        spaces += 1;
    }
}

cout << "There are "s << spaces << " spaces"s << endl;
// вывод: There are 9 spaces
```

Есть специальные методы, которые позволяют добавить элементы в вектор.

- `push_back` вставляет элемент в конец вектора:

```
// Читаем количество питомцев из cin.
int pet_count;
cin >> pet_count;

// Создаём пустой вектор.
vector<int> lost_pet_ages;

// Вектор ещё не сформирован. Используем обычный for, чтобы прочитать
// несколько чисел из cin и добавить их в вектор lost_pet_ages.
for (int i = 0; i < pet_count; ++i) {
    int age;
    cin >> age;
    lost_pet_ages.push_back(age);
}
```

- `insert` добавляет элемент в начало или в середину вектора. Для этого нужно создать итератор, используя конструкцию `month_lengths.begin()`:

```
vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
// Применяем метод insert с указанием места вставки - начала вектора.
month_lengths.insert(month_lengths.begin(), 0);
cout << month_lengths[2] << endl;  // выведет 28 — количество дней в феврале
// Так удобно делать чтобы месяцы нумеровались в векторе с единицы.
```

Итератор задаёт позицию в контейнере.

Меняем размер вектора

Изменить размер вектора можно методом `resize`:

```
vector<int> car_velocities = {60, 53, 67, 19, 77, 59};
car_velocities.resize(4);
// Теперь размер вектора — 4, содержимое: 60, 53, 67, 19
```

Если новый размер больше текущего, в конец вектора добавятся недостающие элементы — «пустые» объекты соответствующего типа.

Чтобы новые элементы имели конкретное значение, его передают в качестве второго аргумента:

```
vector<int> lost_pet_ages = {1, 8, 2, 1, 3, 10};
lost_pet_ages.resize(10, -1);  // Читается так: вектор должен иметь размер 10.
```

```
                                // Если нужно добавить новые элементы,
                                // это должны быть числа -1.
// Теперь размер вектора — 10, содержимое: 1, 8, 2, 1, 3, 10, -1, -1, -1, -1.
```

Чтобы создать вектор, состоящий из некоторого количества одинаковых элементов, нужно указать размер:

```
size_t size;
cin >> size;
vector<string> lost_pet_names(size, "N/A"s);
// Вектор из введённого количества строк N/A.
// Эта аббревиатура используется для замещения отсутствующих данных.
// Задать их можно позже.
```

Если не указывать дополнительных параметров, можно создать пустой вектор:

```
vector<string> elephant_wings;

cout << "Elephant has "s << elephant_wings.size() << " wings."s << endl;
// Вывод: Elephant has 0 wings.
```