Базовые алгоритмы — конспект темы

Алгоритмы count и count_if

В библиотеку `<algorithm>` входят **стандартные алгоритмы** — функции, выполняющие распространённые операции с контейнерами.

Стандартный алгоритм `count` позволяет посчитать, сколько раз конкретный элемент встречается в контейнере:

```
const vector<int> xs = {1, 2, 1, 1, 5};
cout << count(xs.begin(), xs.end(), 1) << endl; // 3
```

Функции `begin` и `end` значат, что операция проводится над всем содержимым вектора. Алгоритм `count` работает с векторами любых типов:

```
const vector<string> escapees = {
    "cat"s, "dog"s, "parrot"s, "cat"s, "rat"s, "parrot"s, "cat"s, "dog"s
};
cout << count(escapees.begin(), escapees.end(), "parrot"s); // 2 попугая
```

Стандартный алгоритм `count_if` принимает третьим аргументом не значение, а ссылку на функцию и считает значения в контейнере, для которых эта функция возвращает `true`:

```
bool IsPositive(int x) {
    return x > 0;
}

int main() {
    // ...
    int num_positive = count_if(xs.begin(), xs.end(), IsPositive);
}
```

Алгоритмы sort и reverse

Алгоритм `sort` из библиотеки `<algorithm>` сортирует указанный контейнер или обычный массив по неубыванию:

```
sort(xs.begin(), xs.end()); // теперь элементы xs будут в возрастающем порядке
```

`sort` умеет сравнивать любые типы данных — строки, числа, пары строк и чисел. Числа сортируются по возрастанию, строки — по алфавиту, пары — сначала по первому элементу, затем по второму.

Алгоритм `reverse` разворачивает отсортированный функцией `sort` вектор:

```
reverse(xs.begin(), xs.end());
```

Алгоритм accumulate

В библиотеку `<numeric>` входят **стандартные алгоритмы** для числовых операций. Например, алгоритм `accumulate`, который складывает все элементы вектора:

```
int sum = accumulate(xs.begin(), xs.end(), 0);
```

`accumulate` выполняет операции слева направо.

По умолчанию `accumulate` суммирует все элементы вектора. Если нужна сумма элементов на конкретном участке, нужно указать начальную и конечную точки и значение, с которого начинается счёт:

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

using namespace std;

int main() {
    int size;
    cin >> size;

    vector<int> values;
    for (int i = 0; i < size; ++i) {
        int value;
        cin >> value;
```

```
        values.push_back(value);
    }

    int sum = accumulate(values.begin(), values.end(), 0);
    cout << sum << endl;
}
```

Встроенные алгоритмы

У некоторых контейнеров есть встроенные версии алгоритмов. Они эффективнее стандартных, так как учитывают внутреннее устройство своего контейнера.

Вот как работает встроенный метод `count` контейнера `set`:

```
const set<string> menagerie = {"cat"s, "dog"s, "rat"s};
cout << menagerie.count("cat"s) << endl;
cout << count(menagerie.begin(), menagerie.end(), "cat"s) << endl;
// Вывод:
// 1
// 1
```