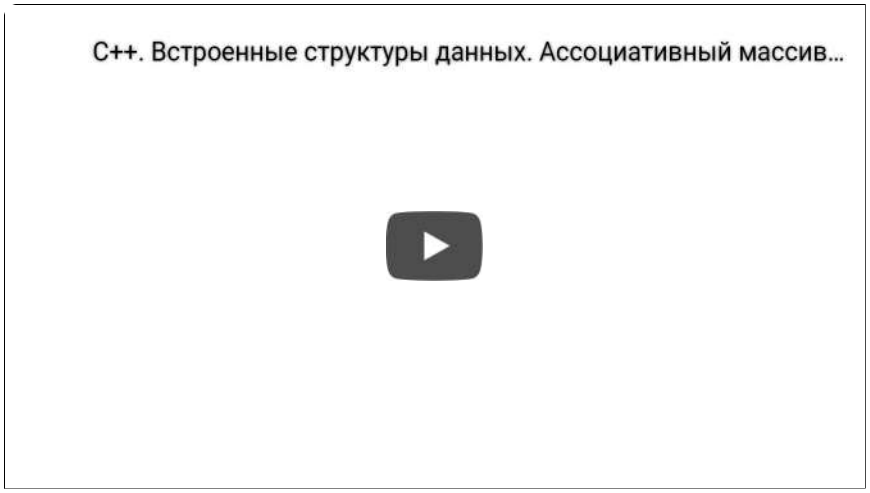
Ассоциативный массив (map)



Рассмотрим следующую задачу. Пусть задана некоторая дата в виде строки, состоящей из числа и названия месяца. Необходимо представить дату в виде "dd.mm" (число и номер месяца, разделённые точкой). Например, строку "December 31" необходимо преобразовать в "31.12".

При решении данной задачи мы можем столкнуться с необходимостью сопоставления названия месяца его номеру. Для этого полезен контейнер `map` (ассоциативный массив). Он позволяет использовать в качестве идентификатора не только числовые индексы, как в массивах, но и другие структуры данных. Идентификатор в ассоциативном массиве называют ключом. Все ключи в ассоциативном массиве уникальны: внутри одного ассоциативного массива невозможно создать два элемента с одинаковым ключом.

Рассмотрим простой пример использования ассоциативного массива:

```
#include <map>
#include <iostream>
using namespace std;
int main()
{
    map <string, int> M;       //создадим пустой ассоциативный массив
    M["January"] = 1;          // создадим элемент с ключом "January" и значением 1
    M["February"] = 2;         // создадим элемент с ключом "February" и значением 2
    M["March"] = 3;            // создадим элемент с ключом "March" и значением 3
    M["March"] = 33;           // присвоим уже существующему элементу с ключом "March" значение
    cout << M.size() << endl; // выведем размер ассоциативного массива
    return 0;
}
```

Важно аккуратно работать с операцией обращения по ключу к элементу ассоциативного массива: любое обращение к элементу по ключу создаёт такой элемент. Например:

```
map<string, int> M;
M["abcaba"] = 1;
cout << M["ab"] << endl;
```

Однако при этом в ассоциативном массиве создастся лишний элемент с ключом "ab" и значением 0. При достаточно больших входных данных программа, которая будет проверять наличие элемента таким образом, может выполняться слишком долго или превысить ограничение по памяти.

Проверить принадлежность элемента ассоциативному массиву можно с помощью метода `find`:

```
if (M.find("ab") != M.end())
```

Удалить элементы из ассоциативного массива можно двумя способами: по ключу — `M.erase("abcaba");` или по итератору — `M.erase(M.begin());`.

В языке C++ элементы ассоциативного массива хранятся в порядке возрастания значений ключей. Пользуясь этим фактом, можно получить, например, значение минимального ключа в ассоциативном массиве: `(*M.begin()).first`, или значение элемента по этому ключу: `(*M.begin()).second`. Данную запись можно упростить: `M.begin()->second`.

Напишем перебор всех элементов ассоциативного массива:

```
for (auto it = M.begin(); it != M.end(); ++it)
    cout << it->first << " " << it->second << endl;
```

Можно написать это иначе:

```
for (auto elem: M)
    cout << elem.first << " " << elem.second << endl;
```

Как и сет, ассоциативные массивы в C++ реализованы с помощью деревьев поиска. Поэтому сложность обращения к элементу словаря по ключу, поиска элемента по ключу, удаления элемента — $O(\log(n))$.

Задача "Голосование"

На телевизионном шоу зрители голосуют за участников шоу, отправляя SMS-сообщение с именем участника. Определите победителя шоу на основе присланных SMS-сообщений.

Решение

```
#include <map>
#include <iostream>
using namespace std;
int main()
{
    map <string, int> count;
    int n, ans = 0;
    string S;
    cin >> n;
    for (int i = 0; i < n; ++i){
        cin >> S;
        count[S]++;
    }
    for (auto elem: count)
        ans = max(ans, elem.second);
    for (auto elem: count)
        if (elem.second == ans)
            cout << elem.first << endl;
    return 0;
}
```