Программирование

Лекция 7. Словари и множества.
Стандартные алгоритмы STL

Множества

- Множества — это математические структуры, которые могут хранить в себе уникальные элементы (то есть, каждый элемент может входить в множество только один раз).
- #include <set>

Множества

- Множества создаются по аналогии с векторами.
- Пишем ключевое слово set, за ним — название типа каждого из элементов множества (в треугольных скобках), а после этого указываем имя для нового множества. Так мы получаем пустое множество.
- Добавление элементов в него происходит с помощью метода insert.
- Чтобы проверить, входит ли элемент во множество, используется метод find. Если элемент в множестве не нашелся, то он выдает то же значение, что и метод end.
- Удаление отдельного элемента из множества выполняется с помощью метода erase.

Решим следующую задачу

- Даны N запросов трёх типов:

1. добавить элемент во множество;
2. проверить, входит ли элемент во множество;
3. удалить элемент из множества.

- Сначала задается число N, а затем сами запросы. Каждый из них состоит из двух чисел. Первое обозначает тип запроса, а второе — элемент, с которым нужно произвести операцию.

```cpp
#include <iostream>
#include <set>

using namespace std;

int main() {
    set <int> s;
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        int type, x;
        cin >> type >> x;
        if (type == 1) { // добавление
            s.insert(x);
        } else if (type == 2) { // проверка
            if (s.find(x) == s.end()) {
                cout << "NO\n";
            } else {
                cout << "YES\n";
            }
        } else { // удаление
            s.erase(x);
        }
    }
    return 0;
}
```

Число n – ск-ко запросов

В строке метод find() возвращал позицию подстроки, «-1» - если не найдено

Если элемент не нашелся, то вернется указатель на конец множ-ва

Ввод элементов множества

```
set <int> s;
int n;
cin >> n;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    s.insert(x);
}
```

Вывод всех элементов множества

Первый способ:

```
for (auto now = s.begin(); now != s.end(); now++) {
    cout << *now << ' ';
}
```

В данном случае **now** — это не очередной элемент, а указатель на него. Метод **begin** возвращает указатель на самый маленький элемент множества, **end** — это конец множества (он идёт после самого большого элемента), а операция ++ осуществляет переход к указателю на следующий элемент. Чтобы посмотреть, что за элемент хранится по указателю, нужно перед его именем написать символ *.

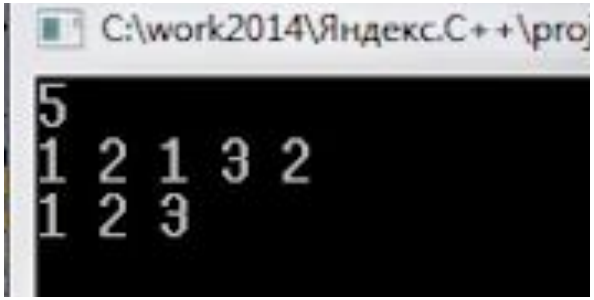
Второй способ позволяет выводить элементы множества аналогично выводу всех элементов в векторе:

```
for (auto now : s) {
    cout << now << ' ';
}
```

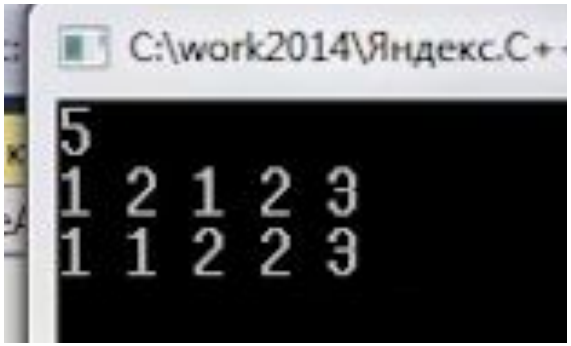
Если ввести одинаковые элементы, то каждый из них окажется во множестве (и будет выведен) только один раз!

В обоих случаях вывод по возрастанию!

Сортировка с помощью множества



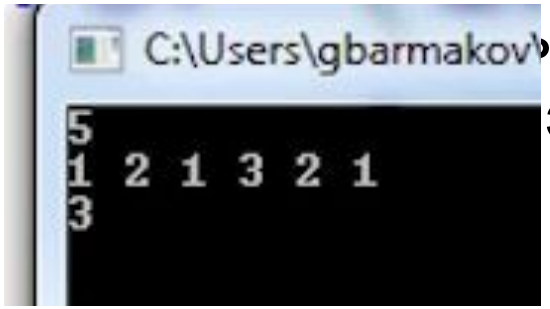
Поскольку проход по элементам множества осуществляется в возрастающем порядке, то велик соблазн использовать его для сортировки последовательностей. Увы, всё портит тот факт, что с одинаковыми элементами множество работает иначе.



Тем не менее, задачу сортировки решить можно и с их помощью. В C++ есть структура **multiset**, которая может хранить в себе одинаковые элементы. Multiset умеет все то же самое, что и обычный set, и лежит в той же библиотеке. Если в предыдущей программе вывода всех элементов заменить set на multiset, то мы как раз и получим элементы по возрастанию (с учетом повторяющихся).

Количество разных элементов

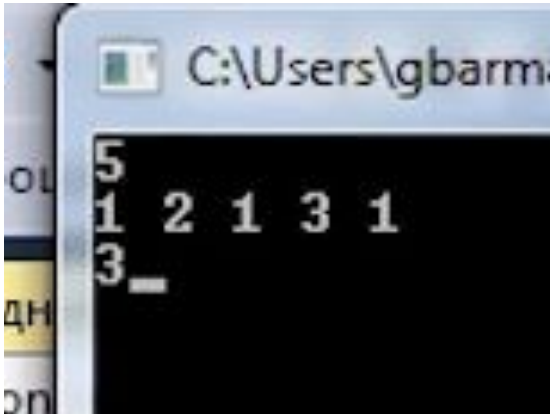
- С помощью set очень легко подсчитать число различных элементов в последовательности. Для этого нужно просто добавить все элементы последовательности во множество, а затем посмотреть на его размер.
- При обработке очередного элемента последовательности нужно сначала проверить, есть ли элемент во множестве. Если его там нет — увеличиваем счётчик на единицу, а затем добавляем элемент во множество. Но есть и более простой способ. У set есть метод **size()**, который возвращает количество элементов во множестве. Достаточно



в него все элементы подряд, а затем этого множества.

Подсчет количества вхождений элемента в последовательность

- Поскольку во множестве все элементы упорядочены, с его помощью легко подсчитать количество вхождений элемента в последовательность. Например, мы хотим посчитать, сколько раз встречается единица. Для решения этой задачи мы будем использовать multiset.



2 новых метода: **lower_bound** и **upper_bound**. l**ower_bound** возвращает указатель на первый элемент, значение которого больше либо равно переданному параметру. u**pper_bound** — на первый элемент, который строго больше. Так мы пробежим от первой единицы до первого элемента (или end'а нашего set'а), на каждом шаге увеличивая значение счётчика вхождений. Если ни одной единицы в последовательности нет, оба метода вернут указатели на больший элемент; будет выполнено 0 шагов.

```
multiset <int> s;
int n;
cin >> n;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    s.insert(x);
}
int cnt = 0;
for (auto now = s.lower_bound(1); now != s.upper_bound(1); now++) {
    cnt++;
}
cout << cnt;
```

Задача 1

Дан список целых чисел, который может содержать до 100000 чисел. Определите, сколько в нем встречается различных чисел.

Входные данные

Вводится число N - количество элементов списка, а затем N чисел.

Выходные данные

Выведите ответ на задачу.

Sample Input:

5

1 2 3 2 1

Sample Output:

3

Задача 2

Во входной строке записана последовательность чисел через пробел. Для каждого числа выведите слово YES (в отдельной строке), если это число ранее встречалось в последовательности или NO, если не встречалось.

Входные данные

Вводится число N - количество элементов списка, а затем N чисел.

Выходные данные

Выведите ответ на задачу.

Sample Input:

6

1 2 3 2 3 4

Sample Output:

NO NO NO YES YES NO

Задача 3

Даны два списка чисел, которые могут содержать до 100000 чисел каждый. Посчитайте, сколько чисел содержится одновременно как в первом списке, так и во втором.

Входные данные

Вводится число N – количество элементов первого списка, а затем N чисел первого списка.

Затем вводится число M - количество элементов второго списка, а затем M чисел второго списка.

Выходные данные

Выведите ответ на задачу.

Sample Input:

3

1 3 2

3

4 3 2

Sample Output:

2

Задача 4

Даны два списка чисел, которые могут содержать до 100000 чисел каждый. Выведите все числа, которые входят как в первый, так и во второй список в порядке возрастания.

Входные данные

Вводится число N – количество элементов первого списка, а затем N чисел первого списка.

Затем вводится число M – количество элементов второго списка, а затем M чисел второго списка.

Выходные данные

Выведите ответ на задачу.

Sample Input:

3

1 3 2

3

4 3 2

Sample Output:

2 3

Словари

- В C++ есть ещё одна структура, похожая на множество, которая называется «словарь». Она ставит в соответствие ключу значение, совсем как в обычном словаре, где каждому русскому слову ставится в соответствие иностранное.
- Словарь в C++ называется **map** (карта). Чтобы пользоваться словарями, нужно подключить библиотеку map.
- Чтобы создать словарь, мы пишем **map**, затем в треугольных скобках через запятую указываем **тип ключа** и **значения**. После этого идёт имя нового словаря.
- Создавать элементы в словаре очень просто. Достаточно написать имя словаря, а затем в квадратных скобках указать ключ. Нужно сразу приравнять этот элемент к какому-либо значению. Тогда к нему можно будет обращаться, просто указав имя словаря и ключ в квадратных скобках.
- Проверка существования элемента делается с помощью метода **find**, как и во множествах.

Словари

- Рассмотрим такую задачу: на телефон поступает входящий звонок с известного номера телефона. Нужно проверить, есть ли он в телефонной книге. Для этого понадобится словарь, в котором числу (номеру) соответствует строка (имя абонента).

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main() {
    map <int, string> s;
    s[112] = "sos";
    if (s.find(112) != s.end()) {
        cout << "YES\n";
    }
    return 0;
}
```

В соответствие числу ставится строка

Проход по элементам словаря

Проход по всем элементам в словаре делается почти так же, как и во множестве.

```
map <int, string> s;
s[112] = "sos";
s[102] = "emergency";
for (auto now : s) {
   cout << now.first << " " << now.second << "\n";
}
```

- В отличие от множества, в словаре на место **now** подставляются пары «ключ-значение». Пара — это особый тип данных, состоящий из двух элементов. Обратиться к первому из них можно как к **now.first** (где now — название пары), а ко второму — **now.second**. Это обращение к полям очень похоже на вызов метода, но после него не нужно писать скобок. Отдельные элементы пары также можно менять как обычные переменные.

- Как и во множестве, ключи в словаре упорядочены по возрастанию. Для поиска ключей можно также пользоваться методами find, lower_bound и upper_bound.

Сопоставление нескольких значений

- Часто требуется сопоставить одному ключу несколько значений. Например, в словаре иностранных слов может быть несколько переводов для одного слова, а в телефонной книге — несколько номеров у одного и того же человека. Чтобы решить задачу такого сопоставления, мы будем использовать в качестве значения вектор.
- Вернёмся к примеру с телефонной книгой. Допустим, нам нужно сохранить для одного абонента все телефонные номера (их мы тоже будем хранить в строке).

```
#include <iostream>
#include <map>
#include <string>
#include <vector>
using namespace std;

int main() {
    map <string, vector <string>> s;
    s["Vasya"] = { "112133", "12341" };
    for (auto now : s["Vasya"]) {
        cout << now << " ";
    }
    return 0;
}
```

Подключим библиотеку для векторов

Имя и номера телефонов

- В этой программе мы сразу инициализировали вектор конкретными значениями, используя фигурные скобки. В принципе, можно создать пустой вектор и добавлять в него элементы с помощью метода push_back. Это удобно в том случае, если мы не знаем заранее, сколько номеров в нашей телефонной книге может быть сопоставлено человеку.
- Если ключ ещё не встречался, нужно создать пустой вектор, а при каждом его обнаружении просто добавлять элемент в вектор.

Стандартные алгоритмы STL

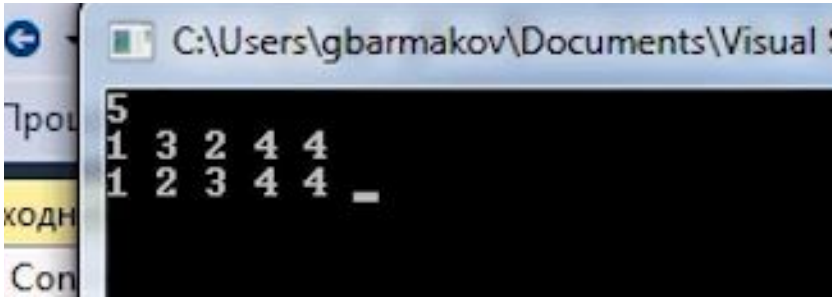
- Сегодня мы будем изучать разные алгоритмы, которые есть в стандартной библиотеке C++. Они помогают быстрее писать программы. В основном мы будем использовать библиотеку algorithm.

Сортировка

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

using namespace std;

int main() {
    int n;
    cin >> n;
    vector <int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    sort(a.begin(), a.end());
    for (auto now : a) {

        cout << now << " ";
    }
    return 0;
}
```

Возьмём последовательность чисел, которую нужно считать, упорядочить и вывести.

Функция sort принимает на вход два параметра: начало и конец сортируемой области. В нашем случае это начало вектора (begin) и его конец



Структуры

- Чтобы описывать объекты, которые характеризуются несколькими разными значениями, нужны структуры. Фактически, структура — это новый тип переменных.
- Сначала нужно описать структуру, а после этого можно создавать переменные, вектора и прочие элементы такого типа.
- Описание структуры должно следовать сразу после using namespace std и находиться вне функций.
- Например, мы хотим описать человека при помощи двух характеристик: рост и имя.

```
struct man {
   int height;
   string name;
};
```

Обратите внимание на точку с запятой после фигурной скобки — она обязательно нужна.

```
int main() {
    int n;
    cin >> n;
    vector <man> a(n);
    for (int i = 0; i < n; i++) {
        int temp;
        string s_temp;
        cin >> temp >> s_temp;
        man struct_temp; // временная структура
        struct_temp.height = temp;
        struct_temp.name = s_temp;
        a[i] = struct_temp; // создание пары значение - номер
    }
    sort(a.begin(), a.end(), cmp);
    for (auto now : a) {
        cout << now.name << endl;
    }
    return 0;
}
```

Устойчивая сортировка

- Сортировка называется устойчивой, если она сохраняет взаимный порядок одинаковых элементов. Если Вася и Петя одного роста, но в исходной последовательности Вася стоял раньше Пети, то после устойчивой сортировки Вася также должен идти перед Петей.
- При использовании sort взаимный порядок одинаковых элементов может нарушиться (Петя окажется перед Васей). Чтобы этого не произошло, нужно использовать функцию устойчивой сортировки **stable_sort**. Она принимает те же параметры, что и sort, но работает немного медленнее.

Задача 5

Отсортируйте массив.

Входные данные

Первая строка входных данных содержит количество элементов в массиве $N \leq 10^5$. Далее идет N целых чисел, не превосходящих по абсолютной величине 10^9.

Выходные данные

Выведите эти числа в порядке неубывания.

Sample Input:

5

5 4 3 2 1

Sample Output:

1 2 3 4 5

Ответ на задачу 1

```
#include <iostream>
#include <set>

using namespace std;

int main() {
  set <int> s;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    s.insert(x);
  }
  cout << s.size();
  return 0;
}
```

Ответ на задачу 2

```
int main() {
  set <int> s;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;

    if (s.find(x) == s.end()) {
      cout << "NO\n";
    } else {
      cout << "YES\n";
    }
    s.insert(x);
  }

  return 0;
}
```

Ответ на задачу 3

```
set <int> s;
set <int> s1;
set <int> s2;
int n1;
cin >> n1;
for (int i = 0; i < n1; i++) {
  int x;
  cin >> x;
  s1.insert(x);
  s.insert(x);
}
int n2;
cin >> n2;
for (int i = 0; i < n2; i++) {
  int x;
  cin >> x;
  s2.insert(x);
  s.insert(x);
}
cout << s2.size() - (s.size() - s1.size());
```

Ответ на задачу 4

```
set <int> s1;
set <int> s2;
set <int> s;
int n1;
cin >> n1;
for (int i = 0; i < n1; i++) {
  int x;
  cin >> x;
  s1.insert(x);
}
int n2;
cin >> n2;
for (int i = 0; i < n2; i++) {
  int x;
  cin >> x;
  s2.insert(x);
  if (s1.find(x) != s1.end()) {
    s.insert(x);
  }
}

for (auto now : s) {
  cout << now << " ";
```

Ответ на задачу 5

```
set <int> s1;
set <int> s2;
set <int> s;
int n1;
cin >> n1;
for (int i = 0; i < n1; i++) {
  int x;
  cin >> x;
  s1.insert(x);
}
int n2;
cin >> n2;
for (int i = 0; i < n2; i++) {
  int x;
  cin >> x;
  s2.insert(x);
  if (s1.find(x) != s1.end()) {
    s.insert(x);
  }
}

for (auto now : s) {
  cout << now << " ";
```

Ответ на задачу 4

```
set <int> s1;
set <int> s2;
set <int> s;
int n1;
cin >> n1;
for (int i = 0; i < n1; i++) {
  int x;
  cin >> x;
  s1.insert(x);
}
int n2;
cin >> n2;
for (int i = 0; i < n2; i++) {
  int x;
  cin >> x;
  s2.insert(x);
  if (s1.find(x) != s1.end()) {
    s.insert(x);
  }
}

for (auto now : s) {
  cout << now << " ";
```

Ответ на задачу 4

```
set <int> s1;
set <int> s2;
set <int> s;
int n1;
cin >> n1;
for (int i = 0; i < n1; i++) {
  int x;
  cin >> x;
  s1.insert(x);
}
int n2;
cin >> n2;
for (int i = 0; i < n2; i++) {
  int x;
  cin >> x;
  s2.insert(x);
  if (s1.find(x) != s1.end()) {
    s.insert(x);
  }
}

for (auto now : s) {
  cout << now << " ";
```

Ответ на задачу 5

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

using namespace std;


int main() {
  int n;
    cin >> n;
    vector <int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    sort(a.begin(), a.end());
    for (auto now : a) {

        cout << now << " ";
    }
    return 0;
}
```