Пары и кортежи — конспект темы

Улучшаем сравнение

Сортировка объектов по одному параметру

Информация очевидцев о потерянном животном хранится в структуре:

```
struct AnimalObservation {
    string name;   // кличка
    int days_ago;  // сколько дней назад видели
};
```

Напишите вектор структур и отсортируйте объекты компаратором по `days_ago`:

```
vector<AnimalObservation> observations = {{"Мурка"s, 3}, {"Рюрик"s, 1}, {"Веня"s, 2}};
sort(observations.begin(), observations.end(),
     [](const AnimalObservation& lhs, const AnimalObservation& rhs) {
         return lhs.days_ago < rhs.days_ago;
     });
// получим: {"Рюрик"s, 1}, {"Веня"s, 2}, {"Мурка"s, 3}
```

Сортировка объектов по двум параметрам

Информация очевидцев о потерянном животном и состоянии его здоровья хранится в структуре:

```
sort(observations.begin(), observations.end(),
     [](const AnimalObservation& lhs, const AnimalObservation& rhs) {
         if (lhs.days_ago == rhs.days_ago) {
             return lhs.health_level < rhs.health_level;
         } else {
             return lhs.days_ago < rhs.days_ago;
         }
     });
```

Этот компаратор можно записать в виде сложного логического выражения:

```
sort(observations.begin(), observations.end(),
     [](const AnimalObservation& lhs, const AnimalObservation& rhs) {
         // благодаря приоритету операций скобки в выражении необязательны
         return lhs.days_ago < rhs.days_ago
                || (lhs.days_ago == rhs.days_ago
                    && lhs.health_level < rhs.health_level);
     });
```

Если `lhs.days_ago < rhs.days_ago`, первый параметр будет считаться меньше второго, так как первый аргумент операции `||` — истинный.

Если `lhs.days_ago == rhs.days_ago`, первый параметр будет меньше только при `lhs.health_level < rhs.health_level`.

Пары в компараторах

Задачу сравнить объекты по двум параметрам можно автоматизировать:

```
sort(observations.begin(), observations.end(),
     [](const AnimalObservation& lhs, const AnimalObservation& rhs) {
         return pair(lhs.days_ago, lhs.health_level)
              < pair(rhs.days_ago, rhs.health_level);
     });
```

Пары сравниваются сначала по возрастанию первой компоненты, а при её равенстве — по возрастанию второй. Такой порядок называется **лексикографическим**. Строки и векторы упорядочиваются сначала по первой букве, затем по второй и по третьей.

Кортежи. Начало

Для сортировки по двум параметрам подходит пара. Но когда параметров больше, применяют `tuple` — **кортеж**. В отличие от пары, он хранит неограниченное количество объектов произвольного размера.

От вектора кортежи отличаются тем, что хранят объекты разного типа. Кортежи сравниваются лексикографически:

```
const AnimalObservation lhs = {"Степан"s, 2, 8};
const AnimalObservation rhs = {"Захар"s,  2, 8};
cout << (tuple(lhs.days_ago, lhs.health_level, lhs.name)
```

```
          < tuple(rhs.days_ago, rhs.health_level, rhs.name)) << endl;
// выведет 0, так как "Степан"s > "Захар"s
```

Если компилятор старый, и код не компилируется, используйте функцию `make_tuple`.

Создание кортежей

При создании кортежа объекты копируются в него:

```
string name = "Василий"s;
const tuple animal_info(name, 5, 4.1);
name = "Вася"s;  // в animal_info остался Василий
```

Чтобы избавиться от лишнего копирования, примените функцию `tie` из библиотеки `<tuple>`:

```
auto MakeKey() const {
    return tie(days_ago, health_level, name);
}
```

Функция `tie` вернула кортеж, хранящий не сами объекты, а ссылки на них: `const int&`, `const int&` и `const string&`.

Возврат нескольких значений из функции

Получить доступ к отдельным элементам кортежа можно так:

```
const tuple animal_info("Василий"s, 5, 4.1);
cout << "Пациент "s << get<0>(animal_info)
     << ", "s << get<1>(animal_info) << " лет"s
     << ", "s << get<2>(animal_info) << " кг"s << endl;
// Пациент Василий, 5 лет, 4.1 кг
```

Другой способ — обратиться к полю по его типу вместо индекса, если это единственное поле указанного типа:

```
const tuple animal_info("Василий"s, 5, 4.1);
cout << "Пациент "s << get<string>(animal_info)
```

```
     << ", "s << get<int>(animal_info) << " лет"s
     << ", "s << get<double>(animal_info) << " кг"s << endl;
// Пациент Василий, 5 лет, 4.1 кг
```

В обоих случаях неясно, что лежит в месте использования. Больше подойдёт структура с полями `name`, `age` и `weight`.

Компактный способ сохранить несколько значений из функции — **распаковка**:

```
class SearchServer {
public:
    tuple<vector<string>, DocumentStatus> MatchDocument(const string& raw_query, int docum
ent_id) const {
        // ...
    }

    // ...
};

// ...

const auto [words, status] = search_server.MatchDocument("белый кот"s, 2);
```

Из метода возвращаются два объекта: `vector<string>` и `DocumentStatus`. У них нет самостоятельных названий. Догадаться о смысле объектов можно по типам и названию метода. Первый объект — это вектор слов запроса, которые нашлись в документе `document_id`, а второй объект — статус документа.

Вещественные числа и задача о задачах

Релевантность измеряется вещественными числами.

Память, отводимая под переменную типа `double`, ограничена, а числа хранятся в двоичной записи. Их точность достаточно высока, но неидеальна. Поэтому не сравнивайте вещественные числа на равенство операторами `==`, `!=`, `<=` и `>=`. Если сделать это нужно, вместо применения `==` вычислите разность чисел и проверьте, укладывается ли она в погрешность.

Функция `abs` из библиотеки `<math>` вычисляет модуль числа.