

Задание по C++ №7, осень 2018 года

Реализуйте шаблонную версию класса `Array`, не полагаясь на то, что для типа `T` определен оператор присваивания и конструктор по умолчанию. Конструктор копирования у типа `T` есть.

```
// Список операций:
//
// explicit Array(size_t size = 0, const T& value = T())
//   конструктор класса, который создает
//   Array размера size, заполненный значениями
//   value типа T. Считайте что у типа T есть
//   конструктор, который можно вызвать без
//   без параметров, либо он ему не нужен.
//
// Array(const Array &)
//   конструктор копирования, который создает
//   копию параметра. Считайте, что для типа
//   T определен оператор присваивания.
//
// ~Array()
//   деструктор, если он вам необходим.
//
// Array& operator=(...)
//   оператор присваивания.
//
// size_t size() const
//   возвращает размер массива (количество
//   элементов).
//
// T& operator[](size_t)
// const T& operator[](size_t) const
//   две версии оператора доступа по индексу.
```

Реализуйте шаблонную функцию `minimum`, которая находит минимальный элемент, который хранится в экземпляре шаблонного класса `Array`, при этом типовой параметр шаблона `Array` может быть произвольным. Чтобы сравнивать объекты произвольного типа, на вход функции также будет передаваться компаратор, в качестве компаратора может выступать функция или объект класса с перегруженным оператором `()`. Примеры вызова функции `minimum`:

```
bool less(int a, int b) { return a < b; }
struct Greater { bool operator()(int a, int b) const { return b < a; } };
```

```
Array<int> ints(3);
ints[0] = 10;
ints[1] = 2;
ints[2] = 15;
int min = minimum(ints, less); // в min должно попасть число 2
int max = minimum(ints, Greater()); // в max должно попасть число 15
```

Задание по C++ №7, осень 2018 года

Шаблонный класс `Array` может хранить объекты любого типа, для которого определён конструктор копирования, в том числе и другой `Array`, например, `Array< Array<int> >`. Глубина вложенности может быть произвольной. Напишите шаблонную функцию (или несколько) `flatten`, которая принимает на вход такой "многомерный" `Array` неизвестной заранее глубины вложенности и выводит в поток `out` через пробел все элементы, хранящиеся на самом нижнем уровне. Примеры работы функции `flatten`:

```
Array<int> ints(2, 0);
ints[0] = 10;
ints[1] = 20;
flatten(ints, std::cout); // выводит на экран строку "10 20"
```

```
Array< Array<int> > array_of_ints(2, ints);
flatten(array_of_ints, std::cout); // выводит на экран строку "10 20 10 20"
```

```
Array<double> doubles(10, 0.0);
flatten(doubles, std::cout); // работать должно не только для типа int
```

Hint: шаблонные функции тоже можно перегружать, из нескольких шаблонных функций будет выбрана наиболее специфичная.