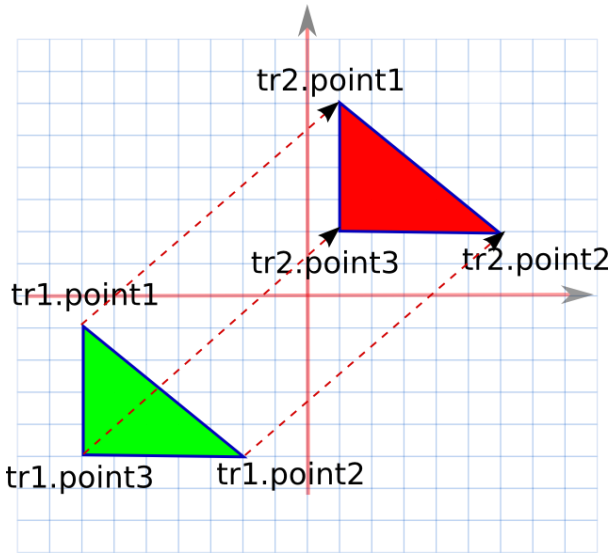
Задача 1 (Прямоугольник и квадрат)

Треугольник задан координатами своих вершин.



Треугольник **a** равен треугольнику **b**, если его можно переместить, при необходимости повернуть и наложить на треугольник **b**.

```
class Coord{
  public:
   int x;
   int y;
};

// Класс треугольник
class Triangle{
public:
  Coord p1,p2,p3;
};
```

Написать функции:

`void getTri(Triangle& tri)`	получает данные треугольника, т.е. считывает со стандартного потока ввода координаты вершин треугольника, заполняет структуру `Triangle` и возвращает её через `tri`
`Coord getPoint()`	вспомогательная функция - считывает со стандартного потока ввода координаты точки и возвращает их через класс `Coord`.
`int cmp(Triangle a, Triangle b)`	сравнивает треугольник **a** с **b**. Если они равны, возвращает 1, если нет, возвращает 0.

Входные данные

координаты вершин двух треугольников (первые два числа - координаты первой точки первого треугольника, следующие два числа - координаты второй точки первого треугольника и т.д.)

Выходные данные

результат **cmp**

Примеры

Вход	Выход
0 0 3 0 0 4 1 1 4 1 1 5	1
10 0 10 3 14 0 -5 -1 -1 -1 -1 -5	0
10 0 10 3 14 0 -5 -1 -1 -1 -1 -4	1

Задача 2 (Queue)

Вам требуется реализовать класс Queue, аналогичный адаптеру std::queue. Он является обёрткой над некоторым стандартным контейнером и реализует интерфейс очереди. Класс должен быть шаблонным. Первый шаблонный параметр T — тип хранимых элементов. Второй шаблонный параметр — контейнер, используемый для хранения элементов (по умолчанию — std::deque<T>):

```
template <typename T, typename Container = std::deque<T>>
class Queue;
```

Предусмотрите в классе следующее:

1. Конструктор по умолчанию, создающий пустую очередь.
2. Константную функцию front, которая возвращает элемент, стоящий в начале очереди.
3. Неконстантную функцию front, которая возвращает по ссылке элемент, стоящий в начале очереди — тем самым давая возможность его изменить.
4. Функцию pop, которая убирает элемент из начала очереди (и ничего не возвращает)
5. Функцию push, которая кладёт переданный элемент в конец очереди.
6. Функцию size, которая возвращает количество элементов.
7. Функцию empty, которая возвращает true тогда и только тогда, когда очередь пуста
8. Операторы == и != для сравнения двух очередей.

Задача 3(Table)

Вам надо написать шаблонный класс Table для электронной таблицы. Для простоты будем считать, что все ячейки таблицы имеют один и тот же тип данных T. Таблица должна уметь менять свой размер по требованию пользователя. Вновь созданные ячейки должны заполняться значениями по умолчанию типа T.

Требования к классу такие:

1. Класс должен называться Table.
2. У класса должен быть шаблонный параметр T — тип элемента в ячейке.
3. У класса должен быть конструктор, получающий на входе два числа типа size_t, — начальные размеры таблицы.
4. У класса должны быть константная и неконстантная версии оператора [], возвращающего нечто такое, к чему снова можно было бы применить оператор []. То есть, должны работать конструкции вида std::cout << table[i][j]; и table[i][j] = value;. Проверять корректность индексов при этом не нужно.
5. У класса должна быть функция resize, получающая на вход два параметра типа size_t и меняющая размер таблицы. Старые данные, умещающиеся в новый размер, должны при этом сохраниться.
6. У класса должна быть функция size, возвращающая std::pair<size_t, size_t> — размер таблицы (в том же порядке, в котором эти аргументы передавались в конструктор).