Contents

1 Non-code things

1.1 Hash

Hash: 9538616d87aa2d06c37c129736430a98

```
tr -d '[:space:]' | md5sum | cut -d ' ' -f 1
```

1.2 Makefile

Hash: 1be30703415446aaf3a1260294222d71

```
CXX = g++
CXXFLAGS = -Wall -Wextra -pedantic -std=c++11 -O2
    ↪ -Wshadow -Wformat=2 -Wfloat-equal
    ↪ -Wconversion -Wlogical-op -Wshift-overflow=2
    ↪ -Wduplicated-cond -Wcast-qual -Wcast-align
DEBUGFLAGS = -D_GLIBCXX_DEBUG
    ↪ -D_GLIBCXX_DEBUG_PEDANTIC -fsanitize=address
    ↪ -fsanitize=undefined
    ↪ -fno-sanitize-recover=all -fstack-protector
    ↪ -D_FORTIFY_SOURCE=2

CXXFLAGS += $(DEBUGFLAGS)

TARGET := $(notdir $(CURDIR))
EXECUTE := ./$(TARGET)

CASES := $(sort $(basename $(wildcard *.in)))
TESTS := $(sort $(basename $(wildcard *.out)))

all: $(TARGET)

clean:
  -rm -rf $(TARGET) *.res

%: %.cpp
  $(LINK.cpp) $< $(LOADLIBES) $(LDLIBS) -o $@

run: $(TARGET)
  time $(EXECUTE)

%.res: $(TARGET) %.in
  time $(EXECUTE) < $*.in > $*.res

%.out: %

test_%: %.res %.out
  diff $*.res $*.out

runs: $(patsubst %,%.res,$(CASES))
test: $(patsubst %,test_%,$(TESTS))

.PHONY: all clean run test test_% runs

.PRECIOUS: %.res
```

1.3 vimrc

Hash: 8f870abf0ba8837fb91734ae9a941ba8

```
set nocp ai bs=2 cul hls ic is lbr ls=2 mouse=a nu
    ↪ ru sc scs smd so=3 sw=4 ts=4
filetype plugin indent on
syntax on

map gA m'ggVG"+y''
```

1.4 nanorc

Hash: 4364dc56fff2b10d5aacd6dc61625802

```
set tabsize 4
set const
set autoindent
```

2 Geometry

2.1 Point

Hash: a1ef04616fa78cdafb4e4425490521b7

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-02-26
 * License: CC0
 * Source: My head with inspiration from tinyKACTL
 * Description: Class to handle points in the plane.
 *  T can be e.g. double or long long. (Avoid int.)
 * Status: Works fine, used a lot
 */
#pragma once

template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) <
      ↪ tie(p.x,p.y); }
  bool operator==(P p) const { return
      ↪ tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return
      ↪ (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return
      ↪ sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes
      ↪ dist()=1
  P perp() const { return P(-y, x); } // rotates +90
      ↪ degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around
      ↪ the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
};
```

3 Java/Python

3.1 Java IO

3.2 Java BigInteger

3.3 Python IO